



Açık kaynak · Türkçe · DO-178C

DO-178C ile Emniyet-Kritik Aviyonik Yazılım

Emniyet-kritik aviyonik yazılım geliştirme, doğrulama ve sertifikasyon

M. Serdar Karaman ve katkıda bulunanlar

DO-178C ile Emniyet-Kritik Aviyonik Yazılım

M. Serdar Karaman ve katkıda bulunanlar

Bu PDF, aviyonikyazilim.com/kitap adresindeki kitabın 7 Ekim 2026 tarihli içeriğinden otomatik olarak üretilmiştir. Kitap yaşayan bir çalışmadır; güncel sürüm her zaman sitededir.

İçerik Creative Commons Atıf-AynıLisanslaPaylaş 4.0 (CC BY-SA 4.0) ile lisanslanmıştır: kaynağı belirterek paylaşabilir, uyarlayabilir ve aynı lisansla yayımlayabilirsiniz.

Bir hata gördüyseniz ya da katkıda bulunmak isterseniz:
github.com/Mavrikant/aviyonikyazilim

İçindekiler

Kitap Hakkında	5
Kısım I – Giriş	9
1. Giriş ve Genel Bakış	10
Kısım II – Emniyet-Kritik Yazılım Geliştirmenin Bağlamı	18
2. Sistem Bağlamında Yazılım	19
3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım	27
Kısım III – DO-178C ile Emniyet-Kritik Yazılım Geliştirme	39
4. DO-178C ve Destekleyici Dokümanlara Genel Bakış	40
5. Yazılım Planlama	51
6. Yazılım Gereksinimleri	64
7. Yazılım Tasarımı	77
8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon	89
9. Yazılım Doğrulama	103
10. Yazılım Konfigürasyon Yönetimi	120
11. Yazılım Kalite Güvencesi	131
12. Sertifikasyon İrtibatı	142
Kısım IV – Araç Kalifikasyonu ve DO-178C Ekleri	152
13. DO-330 ve Yazılım Aracı Kalifikasyonu	153
14. DO-331 ve Model Tabanlı Geliştirme ve Doğrulama	165
15. DO-332 ve Nesne Yönelimli Teknoloji ve İlgili Teknikler	177
16. DO-333 ve Biçimsel Yöntemler	187
Kısım V – Özel Konular	199
17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar	200
18. Sahada Yüklenebilir Yazılım	211
19. Kullanıcı Tarafından Değiştirilebilir Yazılım	225
20. Gerçek Zamanlı İşletim Sistemleri	235
21. Yazılım Bölümlemesi	246

22. Konfigürasyon Verisi	258
23. Havacılık Verileri	268
24. Yazılım Yeniden Kullanımı	279
25. Tersine Mühendislik	290
26. Yazılım Yaşam Döngüsü Faaliyetlerinde Dış Kaynak Kullanımı	300
Ekler	310
Ek A: Örnek Geçiş Kriterleri	311
Ek B: Gerçek Zamanlı İşletim Sistemlerinde Endişe Alanları	317
Ek C: Gerçek Zamanlı İşletim Sistemi Seçiminde Sorulacak Sorular	323
Ek D: Yazılım Servis Geçmişi Soruları	329
Kaynaklar	335
Kısaltmalar	336
SW SOI-1	374
SW SOI-2	379
SW SOI-3	384
SW SOI-4	389

Kitap Hakkında

Bu kitap, DO-178C ekseninde emniyet-kritik aviyonik yazılım geliştirmeyi, doğrulamayı ve sertifikasyon kanıtı üretmeyi Türkçe ve özgün bir dille anlatır. Amaç, standardı yalnızca maddeler hâlinde özetlemek değil; neden bu beklentilerin bulunduğunu, pratikte nasıl uygulandığını ve tipik proje kararlarını nasıl etkilediğini göstermektir.

Kitap, yazılımın sistem ve emniyet değerlendirmesi içindeki yerinden başlar; ardından DO-178C'nin yaşam döngüsü süreçlerini (planlama, gereksinim, tasarım, kodlama ve entegrasyon, doğrulama, konfigürasyon yönetimi, kalite güvencesi, sertifikasyon irtibatı), araç kalifikasyonunu ve teknoloji eklerini (supplement), son olarak da gerçek zamanlı işletim sistemleri, yazılım bölümlenmesi, sahada yüklenebilir yazılım ve yeniden kullanım gibi özel konuları aynı çerçevede ele alır. Böylece okuyucu yalnızca belge isimlerini değil, o belgelerin proje akışındaki yerini, birbirleriyle olan bağıntısını ve emniyet hedefleriyle ilişkisini de görür.

Bu içerik, yeni başlayan bir mühendisin konuyu sistemli biçimde öğrenmesine yardımcı olacak kadar açıklayıcı; deneyimli bir ekip üyesinin ise kavramlar arasındaki ilişkiyi daha net kurmasını sağlayacak kadar ayrıntılı olacak şekilde yazılmıştır.

Kitap nasıl okunmalı?

Kitabı baştan sona sıralı okumak en iyi yaklaşımdır: kısımlar girişten bağlama, DO-178C süreçlerinden araç kalifikasyonu ve teknoloji eklerine, oradan özel konulara ilerler ve ilk kez okuyan biri için en az sürtünmeli rotayı verir. Bu sıra zorunlu bir okuma emri değildir; belirli bir iş üzerindeyseniz aşağıdaki rotalardan birini izleyebilirsiniz.

Durumunuz	Önerilen rota
Konuya yeni başlıyorsunuz	1. Giriş ve Genel Bakış , ardından 4. DO-178C ve Destekleyici Dokümanlara Genel Bakış
Yazılım seviyesinin nereden geldiğini anlamak istiyorsunuz	2. Sistem Bağlamında Yazılım ve 3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım
Proje başlangıcındasınız	5. Yazılım Planlama ve 12. Sertifikasyon İrtibatı
Bir gereksinim seti üzerinde çalışıyorsunuz	6. Yazılım Gereksinimleri ve 7. Yazılım Tasarımı
Kod yazıyor ya da entegrasyon yapıyorsunuz	8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon
Test aşamasındasınız	9. Yazılım Doğrulama , 17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar ve Ek A: Örnek Geçiş Kriterleri
Değişiklik, sürüm ya da süreç denetimiyle uğraşıyorsunuz	10. Yazılım Konfigürasyon Yönetimi ve 11. Yazılım Kalite Güvencesi

Durumunuz	Önerilen rota
Araç kullanımı ya da model tabanlı geliştirme tartışıyorsunuz	13. DO-330 ve Yazılım Aracı Kalifikasyonu ve 14. DO-331 ve Model Tabanlı Geliştirme ve Doğrulama

Bunların dışında:

- **Başvuru arıyorsanız** kitap sonundaki Ek A–D ile Kaynaklar sayfalarını birlikte kullanın: kısaltmalar sözlüğü ve katılım aşaması (Stage of Involvement, SOI) denetimlerine hazırlık için dört kontrol listesi.
- **Bir konuya takıldırıysanız** blog yazılarını destekleyici kısa notlar gibi değerlendirin; çoğu yazı, kitabın daha geniş akışındaki bir kavrama açıklık getirmek için yazılmıştır.

Bu kitap ne değildir?

- Bir standart çevirisi değildir.
- Belge maddelerini ezberleten bir özet kitap değildir.
- Tek bir proje tipine kilitlenmiş dar bir uygulama rehberi değildir.

Bu kitap neyi hedefler?

- DO-178C'nin mantıksal yapısını açıklamak
- Emniyet hedefleri ile yazılım iş ürünleri arasındaki bağı kurmak
- Planlama, geliştirme, doğrulama ve güvence faaliyetlerini ilişkilendirmek
- Araç kalifikasyonunun ve teknoloji eklerinin ne zaman devreye girdiğini göstermek
- Özel konuların neden sertifikasyon açısından önemli olduğunu anlatmak

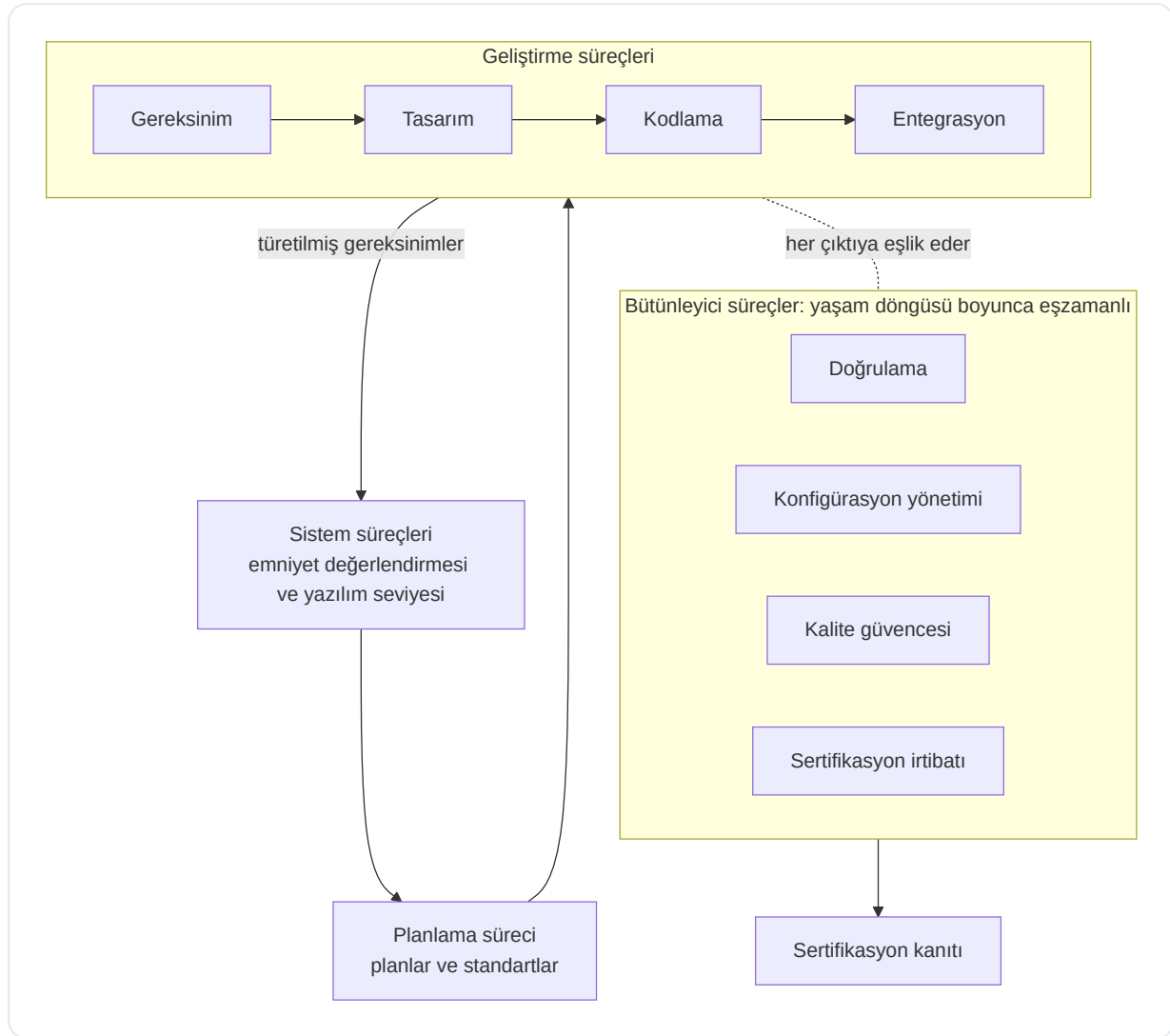
Kapsam ve esas alınan dokümanlar

Anlatım sivil havacılık sertifikasyonu bağlamındadır ve şu dokümanları esas alır:

- **DO-178C / ED-12C** (2011). Yasa değildir; FAA'nın AC 20-115D, EASA'nın AMC 20-115D dokümanı ile (ikisi de 2017) kabul edilebilir uyum yöntemi olarak tanınır.
- Araç kalifikasyonu belgesi **DO-330** ile **DO-331**, **DO-332** ve **DO-333** teknoloji ekleri (hepsi 2011). DO-330 bir teknoloji eki değil, kendi başına duran bir belgedir.
- Sistem tarafında **ARP4754B** ve **ARP4761A** (ikisi de 2023).

Belge ailesinin tamamı, EUROCAE karşılıkları ve kullanım yerleriyle Bölüm 4'teki tabloda verilir. Otorite dokümanları zamanla yenilenir; bir projede hangi sürümün geçerli olduğu sertifikasyon otoritesiyle varılan mutabakata bağlıdır. Kitap yaşayan bir çalışmadır: sitede her sayfanın altındaki son güncelleme tarihi, o sayfanın en son ne zaman elden geçirildiğini gösterir.

Kavramsal akış



Diyagramın üst yarısı işin ana hattını, alt yarısı bu hatta baştan sona eşlik eden faaliyetleri gösterir. Doğrulama, konfigürasyon yönetimi, kalite güvencesi ve sertifikasyon irtibatı geliştirmenin ardından gelen adımlar değildir; DO-178C bunları bütünleyici süreçler (integral processes) olarak tanımlar ve planlamadan teslimata kadar geliştirmeye eşzamanlı yürürler. Ana hat da düz bir üretim hattı değildir, geri beslemelidir: doğrulama bulguları gereksinim dilini düzeltebilir; tasarım eksikleri test kapsamını etkileyebilir; konfigürasyon yönetimi ise tüm çıktılar arasındaki tutarlılığı korur. Sertifikasyon kanıtı sonda derlenen bir paket değil, bu süreçlerin yol boyunca ürettiği verinin toplamıdır.

Bu kitaba atıf ve bağlantı

Kitaptan yararlanıyorsanız kaynağı aşağıdaki biçimde gösterebilirsiniz; erişim tarihini kendiniz yazın:

Karaman, M. S. ve katkıda bulunanlar. DO-178C ile Emniyet-Kritik Aviyonik Yazılım. Aviyonik Yazılım. <https://aviyonikyazilim.com/kitap>. Erişim tarihi: GG.AA.YYYY

İçerik [CC BY-SA 4.0](#) lisanslıdır: kitaba bağlantı verebilir, kaynağı belirterek alıntı yapabilir ve aynı lisansla paylaşabilirsiniz.

Bir web sayfasından ya da dokümandan bağlantı vermek için şu satırı kullanabilirsiniz: `<a href="https://aviyonikyazilim.com/kitap">DO-178C ile Emniyet-Kritik Aviyonik Yazılım</a>`

Yeni blog yazılarını RSS ile takip etmek için akış adresi

<https://aviyonikyazilim.com/blog/rss.xml> şeklindedir. Düzeltme ve katkı önerileri için [GitHub deposunu](#) ya da GitHub kullanmayanlar için e-posta yolunu da gösteren [ana sayfadaki katkı bölümünü](#) kullanabilirsiniz.

KISIM I

Giriş

1. Giriş ve Genel Bakış

Bu kitap, aviyonik yazılımın yalnızca koddan ibaret olmadığını; gereksinim, tasarım, doğrulama ve sertifikasyon kanıtlarından oluşan bütüncül bir çalışma olduğunu anlatır. Bu giriş bölümü, emniyet-kritik yazılımın ne olduğunu, DO-178C'nin sertifikasyondaki yerini ve kitapta kullanılan dilin ve yaklaşımın kısa bir haritasını verir.

Aviyonik ortamda yazılım geliştirmek, sıradan bir ürün yazılımından farklı olarak, emniyet hedefleriyle, donanım sınırlarıyla ve sertifikasyon (certification) beklentileriyle iç içe çalışmayı gerektirir. Dolayısıyla bir proje yalnızca işlev üreten bir uygulama değil; gereksinimleri (requirement) açık, doğrulaması (verification) kanıtlı, baştan sona izlenebilir ve denetlenebilir bir süreç de üretmek zorundadır.

Emniyet-kritik yazılım nedir?

Bir yazılımın "emniyet-kritik" (safety-critical) sayılması, kodun karmaşıklığıyla değil, **hatalı davranışının sonuçlarıyla** ilgilidir. Yazılımın beklenmedik davranışı; uçağın, mürettebatın (flight crew), yolcuların veya yerdeki insanların emniyetini etkileyebiliyorsa, o yazılım emniyet-kritiktir. Uçuş kontrol yazılımı bunun en bilinen örneğidir; ancak motor kontrolü, frenleme, gösterge sistemleri, uyarı üretimi ve seyrüsefer gibi pek çok işlev de aynı sınıfa girer.

Burada kritik olan iki gözlem vardır:

- **Kritiklik dereceli bir kavramdır.** Her aviyonik yazılım aynı ölçüde kritik değildir. Bir kabin eğlence uygulamasının hatası ile bir uçuş kontrol kanununun hatası aynı sonucu doğurmaz. Bu derecelendirme, A'dan E'ye uzanan yazılım seviyeleri (software level) ile resmîleştirilir.
- **Kritiklik yazılımın kendisinden değil, sistemden gelir.** Aynı kod parçası, bir sistemde zararsızken başka bir sistemde tehlikeli olabilir. Bu yüzden yazılım seviyesi, sistem emniyet değerlendirme sürecinin (system safety assessment process) çıktısı olarak belirlenir; yazılım ekibinin kendi başına verdiği bir karar değildir.

Seviyelerin nasıl atandığı ve yazılım ekibinin bu sürece ne geri bildirdiği [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#) bölümünün konusudur.

Temel terimler

Türkçede gündelik dilde birbirinin yerine kullanılan birkaç sözcük, bu alanda farklı şeyleri anlatır. Kitap boyunca şu ayrımlar geçerlidir:

Terim	Bu kitaptaki anlamı
emniyet (safety)	İnsanların ve hava aracının, sistemin kendi yanlış ya da eksik davranışından doğacak kabul edilemez zarardan korunmuş olması
güvenlik (security)	Sistemin kasıtlı ve yetkisiz müdahaleye karşı korunması
hata (error)	Gereksinimde, tasarımda ya da kodda yapılan insan yanlış
kusur (fault)	Hatanın yazılımda bıraktığı iz: yanlış bir ifade, eksik bir koşul, dar seçilmiş bir veri tipi
arıza (failure)	Sistemin ya da bir bileşenin kendisinden beklenen işlevi yerine getirememesi ya da yanlış yerine getirmesi
arıza durumu (failure condition)	Bir ya da birkaç arızanın uçak ve içindekiler üzerindeki etkisiyle tanımlanan durum; şiddetine göre sınıflandırılır

Hata, kusur ve arıza bir zincirin halkalarıdır: mühendis hata yapar, hata üründe kusur olarak kalır, kusur ise ancak onu tetikleyen girdi ve durum bir araya geldiğinde arızaya dönüşür. Her kusur arızaya yol açmaz; ama hangisinin açmayacağı önceden bilinemez. Gündelik konuşmada üçüne de "hata" denir; bu kitap da ayrımın önem taşımadığı yerlerde "yazılım hatası" der, ayrım gerektiğinde üç terimi ayrı kullanır. Arıza durumlarının şiddet sınıfları Bölüm 3'te anlatılır. Kitabın konusu emniyettir; güvenlik, emniyeti etkilediği yerlerde (örneğin uçağa yazılım yüklenmesinde) gündeme gelir.

Emniyet odağı neden önemlidir?

Yazılım, mekanik veya elektronik bileşenlerden farklı bir hata karakterine sahiptir. Bir yapısal parça yorulur, aşınır ve istatistiksel olarak öngörülebilir biçimde bozulur. Yazılım ise **aşınmaz**; kusurları üretim sırasında değil, geliştirme sırasında içine yerleşir ve belirli bir girdi/durum bileşimi oluşana kadar sessizce bekler. Küçük bir örnek bunu somutlaştırır. Gereksinim, bir sensörden 500 ms'den uzun süredir veri gelmiyorsa verinin bayat sayılmasını istiyor olsun:

```
#include <stdbool.h>
#include <stdint.h>

#define TICK_SURESI_MS      (10U)
#define VERI_ZAMAN_ASIMI_MS (500U)

/* Sensörden en son son_veri_tick anında veri geldi; veri bayat mı? */
bool veri_bayat_mi(uint32_t simdi_tick, uint32_t son_veri_tick)
{
    /* Kusur: süre 16 bite daraltılıyor; 65 535 ms'yi aşınca başa sarar. */
    uint16_t gecen_ms = (uint16_t)((simdi_tick - son_veri_tick) * TICK_SURESI_MS);

    return (gecen_ms > VERI_ZAMAN_ASIMI_MS);
}
```

Kod derlenir, uyarı vermez ve sensörün bir saniye ya da bir dakika sustuğu testlerden geçer. Sensör yaklaşık 65,5 saniyeden uzun susarsa hesaplanan süre başa sarar ve işlev, yarım saniye kadar, çoktan bayatlamış veriye "güncel" der; bu pencere her 65,5 saniyede bir yeniden açılır. Zincir eksiksizdir: veri tipini dar seçmek hatadır, o satır kusurdur, bayat verinin kullanılması arızadır. Bu kusuru bulacak olan şans değil yöntemdir: gözden geçirmede veri tipinin aralığını gereksinimle karşılaştırmak, gürbüzlük (robustness) testinde uzun kesintiyi denemek. Örnekten üç sonuç çıkar:

- Yazılım hataları **sistemattiktir**: aynı koşullar oluştuğunda hata her seferinde tekrar eder; yedekli iki kanalda aynı yazılım koşuyorsa, ikisi de aynı anda yanılabilir. Yukarıdaki işlev iki kanalda da aynı anda aynı yanlış cevabı verir; kanalları karşılaştıran bir izleyici uyumsuzluk görmez.
- Yazılıma donanımdaki gibi anlamlı bir **arıza olasılığı** atanamaz. "Bu kodun saatte 10^{-9} olasılıkla hata yapması" ifadesi ölçülebilir bir büyüklük değildir. Olasılık hedefleri sistem düzeyindeki arıza durumları için konur; yazılıma bunun yerine bir yazılım seviyesi atanır ve seviye yükseldikçe gösterilmesi gereken kanıt artar. Bu yaklaşıma geliştirme güvencesi (development assurance) denir.
- Bu yüzden güvence **yalnızca son ürünü test ederek** kurulamaz; ürünü ortaya çıkaran sürecin disipliniyle ve her adımın nesnel kanıtıyla kurulur. Gereksinim tabanlı test bu kanıtın vazgeçilmez parçasıdır; ama sonlu sayıda test, yukarıdaki gibi dar bir pencereyi ancak onu arayan bir yöntemle yakalar. DO-178C bu düşünce üzerine kuruludur.

Yazılımın sistemlerdeki payı her yıl artmaktadır; daha fazla işlev donanımdan yazılıma taşınmakta, yazılım boyutları büyümektedir. Aynı dönemde takvim ve bütçe baskısı da artar. Emniyet odağı, tam da bu baskı altında ilk feda edilen şey olma eğilimindedir — bu kitabın ısrarla süreç ve kanıt vurgusu yapmasının nedeni budur.

DO-178C nedir ve sertifikasyonda nerede durur?

Bir uçak tip sertifikası (type certificate) almadan, üzerindeki ekipman da onaylanmadan hizmete giremez. Bu onayı veren kamu kurumuna sertifikasyon otoritesi (certification authority), kısaca otorite denir: ABD'de Federal Havacılık İdaresi (FAA), Avrupa'da Avrupa Birliği Havacılık Emniyeti Ajansı (EASA), Türkiye'de Sivil Havacılık Genel Müdürlüğü (SHGM). Otorite, başvuru sahibinden ürünün uçuşa elverişlilik (airworthiness) kurallarına uyduğunu göstermesini ister. Bu kurallar sistemin amaçlanan işlevi emniyetle yerine getirmesini şart koşar; yazılımın nasıl geliştirileceğini söylemez.

Aradaki boşluğu DO-178C doldurur. Belge, ABD'de RTCA'nın, Avrupa'da EUROCAE'nin ortak çalışmasıyla hazırlanmış ve 2011'de yayımlanmıştır; Avrupa'da ED-12C adını taşır ve içerikçe aynı belgedir. DO-178C bir yasa değildir. FAA'nın AC 20-115D, EASA'nın AMC 20-115D sayılı

dokümanları onu, hava aracı yazılımı için kabul edilebilir uyum yöntemi (acceptable means of compliance) olarak tanır. Başvuru sahibi başka bir yol önerebilir; ancak o zaman eşdeğer güvenceyi otoriteye kendisi göstermek zorundadır.

DO-178C'yi okurken akılda tutulacak en önemli özellik, hedef tabanlı olmasıdır. Belge bir yaşam döngüsü modeli, yöntem ya da araç dayatmaz; yazılım seviyesine göre karşılanması gereken hedefleri (objective), bu hedeflere götüren faaliyetleri ve hedeflerin karşılandığını gösterecek kanıt tanımlar. Faaliyetler de rehberin parçasıdır; başvuru sahibi başka faaliyetler benimseyebilir, ama bunu otoritenin onayıyla yapar. Seviye A'da 71 hedef vardır; seviye düştükçe sayı azalır ve Seviye E'de hiçbir hedef uygulanmaz. Kanıtın resmî adı yazılım yaşam döngüsü verisidir (software life cycle data): planlar, gereksinimler, tasarım, kod, doğrulama sonuçları ve kayıtlar. Kitapta buna kısaca iş ürünü de denir. Belgenin tarihçesi ve belge ailesi [4. DO-178C ve Destekleyici Dokümanlara Genel Bakış](#) bölümünde, otoriteyle ilişkinin nasıl yürütüldüğü [12. Sertifikasyon İrtibatı](#) bölümünde anlatılır.

Bu kitap ne anlatır?

Kitap, DO-178C ekseninde aşağıdaki sorulara yanıt verir:

- Yazılım neden donanımdan farklı bir güvence yaklaşımı ister?
- Gereksinimler nasıl yazılmalı ve izlenmeli?
- Tasarım, kod ve test birbirine nasıl bağlanmalı?
- Hangi iş ürünleri sertifikasyon kanıtı sayılır?
- Araçlar, modeller ve özel teknikler hangi durumlarda ek değerlendirme ister?

Bu soruların her biri tek başına teknik gibi görünse de aslında süreç tasarımı, organizasyon, iletişim ve kalite kültürüyle doğrudan ilişkilidir.

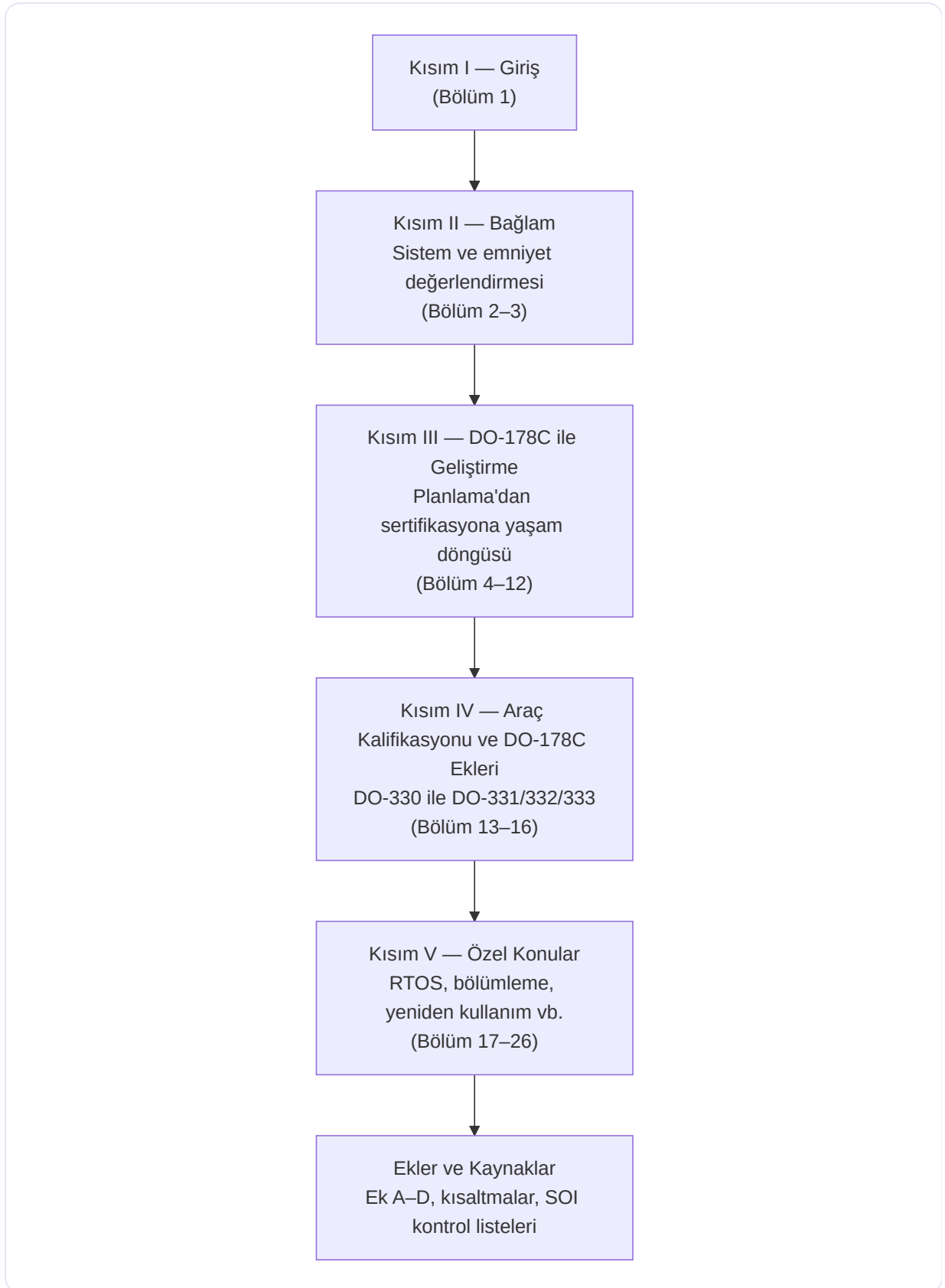
Önemli uyarılar

Kitabı okurken şu sınırlar akılda tutulmalıdır:

- **Bu kitap bir standart metni değildir.** DO-178C ve ilgili dokümanların yerine geçmez; onların mantığını, amacını ve pratikte nasıl uygulandığını kendi cümleleriyle anlatır. Projede resmî dayanak her zaman standardın kendisi ve otoriteyle yapılan anlaşmalardır.
- **Her proje farklıdır.** Burada verilen öneriler yaygın ve denenmiş yaklaşımlardır; ancak sertifikasyon otoritesiyle mutabakata varılmış proje planları her zaman önceliklidir.
- **"Uyum" tek başına amaç değildir.** Hedef, gerçekten emniyetli yazılım üretmektir. Süreç kutucuklarını doldurup emniyeti kaçırmak mümkündür; kitap bu tuzaklara özel olarak dikkat çeker.

Kitabın yapısı

Kitap beş kısımdan oluşur ve kısımlar birbirinin üzerine inşa edilir:



- **Kısım II**, yazılımın içine yerleştiği çerçeveyi kurar: [2. Sistem Bağlamında Yazılım](#) yazılımın tek başına değil bir sistemin parçası olarak geliştirildiğini, [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#) ise yazılım seviyesinin nereden geldiğini anlatır.

- **Kısım III, 4. DO-178C ve Destekleyici Dokümanlara Genel Bakış** ile başlar ve DO-178C'nin süreçlerini birer bölümde işler: planlama, gereksinim, tasarım, kodlama ve entegrasyon (integration), doğrulama, konfigürasyon yönetimi (configuration management), kalite güvencesi (quality assurance) ve sertifikasyon irtibatı (certification liaison). Bölümlerin sırası anlatım kolaylığı içindir; son dört süreç projede geliştirmeyi izleyen adımlar değil, ona baştan sona eşlik eden süreçlerdir.
- **Kısım IV, 13. DO-330 ve Yazılım Aracı Kalifikasyonu** ile araç kalifikasyonunu (tool qualification), ardından DO-178C'nin üç teknoloji ekini (supplement) tanıtır: model tabanlı geliştirme (model-based development), nesne yönelimli teknoloji (object-oriented technology) ve biçimsel yöntemler (formal methods).
- **Kısım V, 17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar** ile başlar ve sahada sık karşılaşılan özel konuları toplar: sahada yüklenebilir yazılım, kullanıcı tarafından değiştirilebilir yazılım, gerçek zamanlı işletim sistemleri, yazılım bölümlenmesi, konfigürasyon verisi, havacılık verileri, yeniden kullanım, tersine mühendislik ve dış kaynak kullanımı.
- **Ekler ve Kaynaklar**, bölümlerde anlatılanı uygulamaya taşıyan başvuru sayfalarıdır: örnek geçiş kriterleri, gerçek zamanlı işletim sistemlerindeki endişe alanları, işletim sistemi seçimi ve servis geçmişi için soru listeleri, **Kısaltmalar** ve otoritenin katılım aşaması (Stage of Involvement, SOI) denetimlerine hazırlık için dört kontrol listesi.

İlk kez okuyan için en verimli sıra kısımların sırasıdır: önce bağlam, sonra süreçler, en son özel konular. Belirli bir konuya ihtiyacı olan okuyucu, Kısım II'deki bağlamı edindikten sonra ilgili bölüme doğrudan da gidebilir; örneğin yapısal kapsam analizi (structural coverage analysis) için **9. Yazılım Doğrulama** bölümüne, araç kalifikasyonu için Bölüm 13'e. Bölümlerin tam listesi ve duruma göre okuma önerileri **Kitap Hakkında** sayfasındadır.

Kitap boyunca dönen üç kavram

Emniyet-kritik yazılımda bir hata yalnızca işlevsel bir sorun değildir; uçuş fazını, bakım prosedürünü veya insan kararını etkileyebilir. Bu yüzden kitabın dili özellikle üç kavrama tekrar tekrar döner:

- **izlenebilirlik** (traceability): bir iş ürününün kökenini ve etkisini gösterebilme,
- **doğrulanabilirlik** (verifiability): beklenen davranışın kanıtlanabilir olması,
- **denetlenebilirlik** (auditability): sürecin dış gözlemci tarafından takip edilebilmesi.

Bu üç nitelik, sertifikasyon otoritesinin sorduğu tek bir sorunun farklı yüzleridir: *"Bu yazılımın amaçlanan işlevini emniyetle yerine getirdiğini bana nasıl gösterirsiniz?"* Kitap boyunca her süreç, her iş ürünü ve her analiz bu soruya verilen yanıtın bir parçası olarak okunmalıdır.

Bu bölümden akılda kalması gerekenler

- Emniyet-kritiklik koddan değil, hatanın sistem düzeyindeki sonucundan gelir ve derecelidir; yazılım seviyesini sistem emniyet değerlendirme süreci belirler.
- Hata insan yanlısıdır, kusur onun yazılımdaki izidir, arıza ise beklenen işlevin yerine gelmemesidir; emniyet ile güvenlik de ayrı kavramlardır.
- Yazılım hataları sistemattiktir ve yazılıma arıza olasılığı atanamaz; güvence, yalnızca son ürünü test etmekle değil, süreci disipline edip her adımın kanıtını üretmekle sağlanır.
- DO-178C yasa değil, otoritelerin tanıdığı kabul edilebilir uyum yöntemidir; yöntem dayatmaz, yazılım seviyesine göre karşılanacak hedefleri tanımlar.
- Bu kitap standardın yerine geçmez; DO-178C'nin mantığını ve pratiğini açıklar.
- Sonraki bölümleri okurken her zaman "hangi kanıt isteniyor?" sorusu sorulmalıdır.

KISIM II

Emniyet-Kritik Yazılım Geliřtirmenin Baęlamı

2. Sistem Bağlamında Yazılım

Yazılımın davranışı, ona atanan sistem fonksiyonları ve dış arayüzleri olmadan anlaşılamaz. Bu bölüm, yazılımın hangi sistem parçasını gerçekleştirdiğini, hangi sınırlar ve arayüzler içinde çalıştığını ve sistem süreçleriyle hangi bilgiyi iki yönde alıp verdiğini açıklar.

Emniyet-kritik geliştirmede temel soru şudur: yazılım hangi fonksiyonu yerine getirir, hangi fonksiyonları devralmaz ve hangi durumlarda sistemi güvenli durumda (safe state) tutar? Bu bakış, sonraki gereksinim ve tasarım faaliyetlerinin temelidir.

Sistem bağlamı neden önemlidir?

Bir yazılım işlevi, sistem bağlamından koparıldığında eksik anlaşılır. Örneğin bir hız hesaplama algoritması, sensör girişinin kalitesi, filtreleme gecikmesi, veri güncelliği ve insan-makine arayüzü ile birlikte değerlendirilmelidir. Aksi hâlde "doğru çalışan" bir kod parçası, yanlış sistem davranışına dönüşebilir.

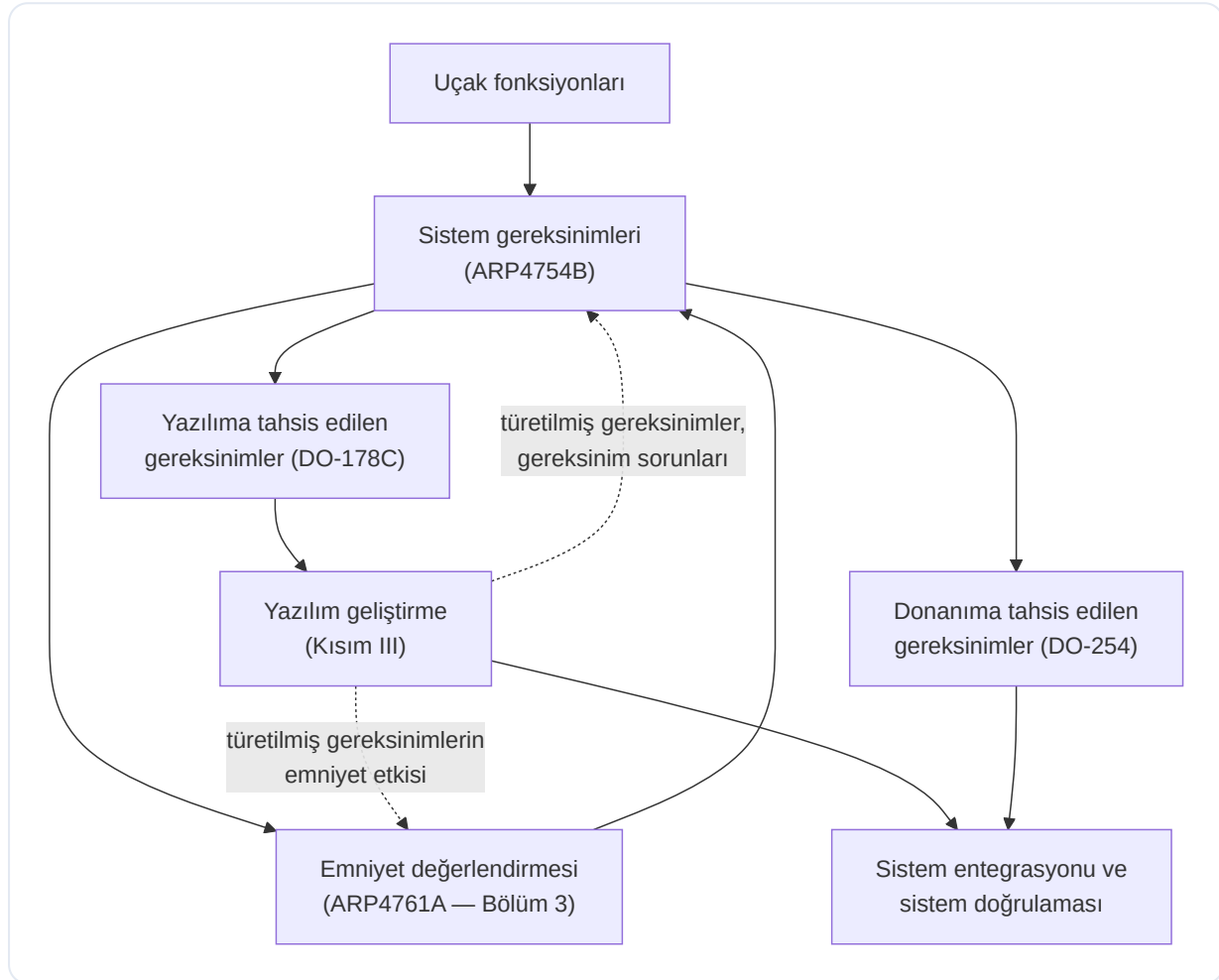
Tipik bir örnek, geçersiz veriyle ne yapılacağıdır. Hava hızı kaynağı geçersiz veri bildirdiğinde yazılım ekibi son geçerli değeri tutmayı makul bulabilir; sistem tarafı ise göstergenin değeri ekrandan kaldırmasını bekliyor olabilir. İki davranış da kendi içinde kusursuz kodlanıp test edilebilir, ama yalnızca biri sistemin emniyet analizindeki varsayımla uyudur. Hangisinin doğru olduğu kodun içinden görünmez; cevap sistem gereksinimlerinde, arayüz tanımında ve emniyet değerlendirmesindedir.

Sistem geliştirmeye genel bakış

Sivil havacılıkta sistem geliştirmenin çerçevesini, sivil uçak ve sistemlerin geliştirilmesine ilişkin kılavuz olan SAE ARP4754B (Avrupa'daki karşılığı EUROCAE ED-79B; Aralık 2023) çizer. Önceki sürüm ARP4754A (2010) hâlâ sık karşınıza çıkar: FAA'nın AC 20-174 genelgesi (2011) bu sürümü kabul edilebilir yöntem olarak tanımıştır ve hizmetteki pek çok program ona göre yürütülmüştür. Bir programda hangi sürümün geçerli olduğu, sertifikasyon otoritesiyle (certification authority) varılan mutabakatla belirlenir. Bu çerçevede uçak, birbirine bağlı üç seviyede ele alınır:

- 1. Uçak seviyesi:** uçağın yerine getireceği fonksiyonlar tanımlanır (kaldırma kontrolü, itki, seyrüsefer, haberleşme...).
- 2. Sistem seviyesi:** uçak fonksiyonları sistemlere paylaştırılır (uçuş kontrol sistemi, motor kontrol sistemi, iniş takımı sistemi...).
- 3. Öğe (item) seviyesi:** her sistemin fonksiyonları donanım ve yazılım öğelerine tahsis edilir (allocation). DO-178C, bu tahsisin **yazılım tarafını** ele alır; donanım tarafındaki karşılığı, havacılık elektronik donanımı için tasarım güvencesi kılavuzu olan DO-254 / ED-80'dir (2000).

Gereksinimlerle birlikte geliştirmenin ne kadar sıkı yürütüleceği de yukarıdan aşağı iner. Sistem tarafında bunun adı geliştirme güvence seviyesidir (development assurance level, DAL): önce fonksiyona fonksiyon geliştirme güvence seviyesi (function development assurance level, FDAL), sonra mimari dikkate alınarak fonksiyonu gerçekleştiren öğelere öge geliştirme güvence seviyesi (item development assurance level, IDAL) atanır. Yazılım ögesine atanan IDAL, DO-178C'de yazılım seviyesi (software level) adını alır. Atamanın nasıl yapıldığı, emniyet değerlendirmesinin yöntemlerini anlatan SAE ARP4761A (ED-135; Aralık 2023) ile birlikte [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#) bölümünün konusudur.



Diyagramdaki kesikli oklar akışın iki yönlü olduğunu gösterir: yazılım geliştirme sırasında keşfedilen sorunlar, eksik veya çelişkili sistem gereksinimleri şeklinde yukarıya geri beslenir; yazılımın kendi ürettiği gereksinimler de sistem ve emniyet süreçlerine iletilir. Sağlıklı bir projede sistem ekibi ile yazılım ekibi arasında sürekli ve kayıt altına alınan bir geri besleme kanalı vardır.

Sistem gereksinimleri

Yazılım gereksinimlerinin kalitesi, kaynağı olan sistem gereksinimlerinin kalitesini aşamaz.

Yazılıma tahsis edilen iyi bir sistem gereksinimi kümesi şu nitelikleri taşır:

- **Atomik ve tekil:** her gereksinim tek bir beklentiye ifade eder; "ve/veya" yığınlarıyla birden çok beklenti tek cümleye sıkıştırılmaz.
- **Doğrulanabilir:** gereksinim, test veya analizle gösterilebilecek biçimde ölçülü yazılır ("hızlı olmalı" değil, "50 ms içinde yanıt vermeli").
- **Belirsizlikten arınmış:** "uygun", "yeterli", "mümkünse" gibi yoruma açık ifadeler içermez.
- **Tutarlı ve tam:** gereksinimler birbiriyle çelişmez; normal koşulların yanında arıza, sınır ve başlatma/kapanma koşullarını da kapsar.
- **İzlenebilir:** her gereksinim, kaynağı olan uçak/sistem fonksiyonuna ve emniyet gereksinimlerine bağlanabilir.

Emniyet değerlendirmesinden gelen, emniyetle ilgili gereksinimler (örneğin bir izleme fonksiyonu, bir sınır koruması, bir güvenli durum geçişi) sistem gereksinimleri içinde **açıkça işaretlenmelidir**; çünkü bunların doğrulanması, sertifikasyon kanıtının en dikkatle incelenen kısmıdır.

Geçerleme ve doğrulama ayrımı

Sistem seviyesinde iki ayrı soru sorulur ve bunlar sık sık karıştırılır:

- **Geçerleme (validation):** *Doğru gereksinimleri mi yazdık?* Gereksinimlerin, gerçekten istenen uçak/sistem davranışını eksiksiz ve doğru ifade ettiğinin gösterilmesidir. Ağırlıklı olarak sistem seviyesinin sorumluluğudur.
- **Doğrulama (verification):** *Ürünü gereksinimlere uygun mu yaptık?* Her seviyedeki çıktının, kendi girdisindeki gereksinimleri karşıladığının gösterilmesidir.

Bu ayrım yazılım ekibi için pratik bir sonuç doğurur: DO-178C süreci ağırlıklı olarak **doğrulama** üzerine kuruludur; yazılıma gelen gereksinimlerin *doğru gereksinimler* olduğu büyük ölçüde sistem seviyesinde geçerlenmiş kabul edilir. Yazılım ekibi yine de anlamsız, çelişkili veya uygulanamaz bir gereksinimle karşılaştığında bunu sisteme geri bildirmekle yükümlüdür — "gelen gereksinim yanlış" savunması, sertifikasyonda kimseyi kurtarmaz.

Tahsis ve yazılımın sınırları

Tahsis, bir sistem gereksiniminin hangi öge tarafından karşılanacağına karar verilmesidir. Yazılım, sistemin her parçasını yapmaz: bazı işlevler donanım, bazıları mürettebat (flight crew) ya da bakım prosedürleri, bazıları da başka sistemler tarafından sağlanır. Bir uçuş kontrol bilgisayarında yazılım sensör verisini yorumlayabilir, komut üretimini yönetebilir, arızaları sınıflandırabilir ve yedek moda geçişi başlatabilir; ama sensörün fiziksel davranışını değiştiremez, kablolamayı düzeltemez ya da çevresel koşulları ortadan kaldıramaz.

Sınırın açık çizilmesi iki tipik sorunu önler:

- **Sahipsiz gereksinim.** Bir sensör arızasını kim algılayacak: donanımın kendi yerleşik testi mi, yazılımın makullük denetimi mi, ikisi birden mi? Tahsis yazılı değilse iki ekip de işi diğerinin yaptığını varsayar ve arıza algılama hiçbir ögenin gereksiniminde yer almaz.
- **Yazılıma sessizce yüklenen varsayım.** "Bu giriş donanımda filtreleniyor", "o değer zaten sınırlanmış geliyor" gibi kabuller bir gereksinime ya da arayüz tanımına bağlanmamışsa, donanım değiştiği gün kimse fark etmeden geçersizleşir.

Sınır her zaman tek bir kutuyla da çakışmaz. Tümleşik modüler aviyonik (integrated modular avionics, IMA) platformlarında birden çok sistem fonksiyonunun yazılımı aynı işlemciyi ve işletim sistemini paylaşır. Bu durumda yazılımın bağlamı hem gerçekleştirdiği fonksiyon hem de üzerinde koştuğu platformdur; farklı yazılım seviyesindeki uygulamaların birbirini etkileyemediği ise yazılım bölümlenmesiyle (software partitioning) gösterilir ([21. Yazılım Bölümlenmesi](#)).

Arayüzler

Arayüzler, sistemin en hassas noktalarıdır. Çünkü veri biçimi, zamanlama, hata işaretleri ve sınır değerler burada tanımlanır. Net olmayan bir arayüz, testin tekrar edilebilirliğini azaltır ve bakım sırasında beklenmeyen yan etkilere yol açar.

İyi bir arayüz tanımı her sinyal için şunları söyler: verinin anlamı ve kaynağı, veri tipi ve birimi, geçerli aralık, zamanlama beklentisi (ne sıklıkla gelir, ne zaman bayat sayılır), geçerliliğin nasıl bildirildiği ve veri geçersiz ya da eksik olduğunda alıcının ne yapacağı. Bu bilgiler sistem seviyesinde arayüz kontrol dokümanında (interface control document, ICD) toplanır ve yazılım gereksinimlerinin doğrudan girdisi olur.

Örnek: tek bir giriş sinyalinin arayüz tanımı

Bir gösterge biriminin, hava veri bilgisayarından ARINC 429 veri yolu üzerinden hesaplanmış hava hızı (computed airspeed) aldığını düşünelim. Sinyalin arayüz tanımı aşağıdaki gibi olabilir (değerler temsildir; gerçek bir projede kaynak ekipmanın arayüz kontrol dokümanından alınır):

Öznitelik	Tanım
Sinyal ve kaynağı	Hesaplanmış hava hızı; hava veri bilgisayarı, ARINC 429 alıcı kanalı 1
Etiket (label) ve kodlama	Etiket 206 (sekizli); ikili (binary, BNR) kodlama
Birim ve çözünürlük	Knot; 0,0625 knot
Geçerli aralık	0–450 knot; aralık dışındaki değer geçersiz sayılır
Yenileme periyodu	100 ms'de bir kelime (word)
Bayatlama süresi	300 ms boyunca yeni kelime alınmazsa veri bayat sayılır

Öznitelik	Tanım
Geçerlilik göstergesi	İşaret/durum matrisi (sign/status matrix, SSM): yalnızca "normal çalışma" değeri geçerlidir; "arıza uyarısı", "hesaplanmış veri yok" ve "işlevsel test" değerleri geçersiz veri anlamına gelir
Geçersiz ya da bayat veride davranış	Değer kullanılmaz, son geçerli değer tutulmaz; hava hızı "geçersiz" olarak işaretlenir ve göstergeden kaldırılır

Tablonun ilk satırları çoğu projede eksiksiz yazılır; eksik kalanlar genellikle son üç satırdır. Bayatlama süresi tanımlı değilse yazılım, kesilmiş bir veri yolunun son değerini süresiz kullanabilir. Geçerlilik göstergesinin hangi değerlerinin "kullanma" anlamına geldiği yazılmamışsa, kaynak ekipmanın test modunda yayımladığı değer gerçek ölçüm sanılabilir. Son satır ise yazılımın tek başına karar veremeyeceği sistem davranışdır; "Sistem bağlamı neden önemlidir?" kısmındaki anlaşmazlık tam burada çözülür. Denetimde de aynı soru sorulur: *bu sinyal kesilirse yazılım ne yapar ve bu nerede yazıyor?*

ARINC 429 kelimesinin alanları için sitedeki [ARINC 429 yazısına](#) bakılabilir. Dış arayüzdeki bu tanımın yazılımın içinde bileşenler arası arayüzlere nasıl taşındığı [7. Yazılım Tasarımı](#) bölümünde ele alınır.

Sistem ile yazılım arasındaki bilgi akışı

Yazılım, sistem gereksinimlerinin bir alt kümesini gerçekleştiren ve sistemin emniyet mimarisinin kendisine biçtiği rolü oynayan bir **öğedir**. Bu ilişki, iki yönde akan ve her biri bir yaşam döngüsü verisinde iz bırakan bilgilerle kurulur.

Sistemden yazılıma gelenler, yazılım geliştirmenin girdisidir:

Bilgi	Kim üretir?	Yazılım tarafında nerede görünür?
Yazılıma tahsis edilen sistem gereksinimleri (işlev, başarımlar, arayüz)	Sistem mühendisliği	Yüksek seviyeli gereksinimlerin (high-level requirements) kaynağı; iz verisi
Emniyetle ilgili gereksinimler ve tasarım kısıtları (izleme, bağımsızlık, bölümlenme)	Sistem emniyet değerlendirme süreci	Emniyetle ilgili olarak işaretlenmiş gereksinimler; yazılım mimarisi
Yazılım seviyesi ve dayandığı arıza durumları (failure condition)	Sistem emniyet değerlendirme süreci	Yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC)
Donanım tanımı ve donanım/yazılım arayüzü (hardware/software interface)	Donanım ve sistem mühendisliği	Gereksinimler ve tasarım tanımı
Hangi gereksinimin sistem düzeyinde doğrulanacağı ve yazılımdan beklenen kanıt	Sistem mühendisliği	Yazılım doğrulama planı (Software Verification Plan, SVP)

Bilgi	Kim üretir?	Yazılım tarafında nerede görünür?
Yazılımın ilettiği türetilmiş gereksinimler ve gereksinim sorunları üzerine yapılan değerlendirmenin sonucu	Sistem mühendisliği ve sistem emniyet değerlendirme süreci	Yazılım doğrulama sonuçları

Yazılımdan sisteme dönenler, sistem ve emniyet süreçlerinin kendi sonuçlarını güncel tutmasını sağlar:

Bilgi	Kim üretir?	Yazılım tarafında nerede görünür?
Türetilmiş gereksinimler ve gerekçeleri	Gereksinim ve tasarım süreçleri	Yazılım gereksinim verisi ve tasarım tanımı
Mimarinin emniyetle ilgili yönleri: bölümler sınırları, izleme ve hata algılama mekanizmaları	Tasarım süreci	Tasarım tanımı
Sistem gereksinimlerinde bulunan eksik, çelişki ve belirsizlikler	Gereksinim, tasarım ve doğrulama süreçleri	Problem raporları (problem report)
Kullanım kısıtları ve açık problem raporlarının işlevsel etkisi	Yazılım ekibi, sertifikasyon irtibatı	Yazılım başarı özeti (Software Accomplishment Summary, SAS)
Yazılımın konfigürasyon tanımlaması; başarımlar, zamanlama ve doğruluk özellikleri	Yazılım ekibi	Yazılım konfigürasyon indeksi (Software Configuration Index, SCI); zamanlama ve bellek payları yazılım başarı özetinde
Sistem doğrulamasında kullanılabilecek yazılım doğrulama kanıtı	Doğrulama süreci	Yazılım doğrulama sonuçları

Türetilmiş gereksinim (derived requirement), yazılım geliştirme süreçlerinde — gereksinim ya da tasarım sırasında — ortaya çıkan ve üst seviye gereksinime doğrudan izlenemeyen ya da orada belirtilenin ötesinde davranış tanımlayan gereksinimdir; ayrıntısı [6. Yazılım Gereksinimleri](#) bölümündedir. Emniyet değerlendirmesinden yazılıma tahsis edilen gereksinim bu anlamda türetilmiş değildir; kaynağı bellidir. Türetilmiş gereksinimin yukarı bildirilmesinin nedeni, sistem analizi yapılırken henüz var olmaması ve emniyet etkisinin değerlendirilmemiş olmasıdır.

Tabloların her satırı aynı zamanda bir denetim sorusudur. Planlama denetiminde yazılım seviyesinin hangi arıza durumuna dayandığı sorulur ([SW SOI-1](#)); geliştirme denetiminde türetilmiş gereksinimlerin sistem süreçlerine iletilişinin kaydı aranır ([SW SOI-2](#)). "Sistem ekibine söylemiştik" bir kayıt değildir.

Donanım/yazılım arayüzü

Yazılımın en yakın komşusu, üzerinde koştugu donanımdır. Donanım/yazılım arayüzü tanımı tipik olarak bellek haritasını, çevre birimi yazmaçlarının adreslerini ve bit anlamlarını, kesme kaynaklarını ve önceliklerini, saat ve zamanlayıcıları, güç verildiğinde ve yeniden başlatmada donanımın hangi durumda olduğunu ve donanımın kendi sağladığı arıza algılama mekanizmalarını içerir.

Bu arayüz iki nedenle özel dikkat ister. Birincisi, türetilmiş gereksinimlerin başlıca kaynağıdır: bekçi köpeği zamanlayıcısının (watchdog timer) hangi aralıkla yenileneceği ya da bir çevre biriminin hangi sırayla başlatılacağı hiçbir sistem gereksiniminde yazmaz, seçilen donanımdan doğar. İkincisi, donanım geliştirme boyunca değişir; yazmaç düzeni ya da kesme ataması değiştiğinde yazılımın bundan haberdar olması, arayüz tanımının sürümlenmiş ve değişiklik kontrolü altında olmasına bağlıdır. Yazılımın bu arayüzle uyumu, sonunda hedef donanım üzerinde koşulan donanım/yazılım entegrasyon testleriyle gösterilir ([9. Yazılım Doğrulama](#)).

Sistem doğrulamasıyla ilişki

Yazılım doğrulaması bittiğinde iş bitmiş olmaz; yazılım donanımla ve diğer öğelerle birleştirilir ve sistem gereksinimlerinin karşılandığı sistem entegrasyonu (system integration) ve sistem doğrulaması sırasında gösterilir. İki düzey birbirinin kanıtını kullanabilir, ama bu kendiliğinden olmaz:

- **Yazılımdan sisteme:** bir sistem gereksinimi yazılım testleriyle zaten gösterilmişse sistem doğrulaması bu kanıta dayanabilir. Bunun için testin hangi ortamda ve hangi yazılım sürümüyle koşulduğu bilinmelidir.
- **Sistemden yazılıma:** bazı yazılım gereksinimleri ancak gerçek sensörler ve eyleyicilerle anlamlı biçimde sınanabilir (uçtan uca gecikme gibi). Bunların sistem düzeyi teste bırakılması bir boşluk değil, plandır — yeter ki hangi gereksinimin nerede doğrulanacağı önceden yazılmış ve otoriteyle paylaşılmış olsun. DO-178C bu krediye bir koşul bağlar: kredi alınan sistem faaliyeti ilgili DO-178C hedeflerini karşılamalı, tamamlandığının kanıtı ve çıktıları da yazılım yaşam döngüsü verisi içinde gösterilmelidir.

Sık görülen hata, bu paylaşımın proje sonunda yapılmasıdır: yazılım ekibi bir gereksinimi "sistem testinde bakılır" diye kapatır, sistem ekibi ise aynı gereksinimin yazılım testinde gösterildiğini varsayar.

Sistem mühendisleri için iyi uygulamalar

Yazılım kalitesini en çok etkileyen kararların bir kısmı, yazılım ekibi işe başlamadan önce sistem seviyesinde verilir. Deneyimin öne çıkardığı uygulamalar:

- **Yazılıma tahsis edilen gereksinimleri erken kararlaştırın**, ama değişecekleri gerçeğini kabul edip değişiklik yönetimini baştan kurun.
- **Arayüzleri tek bir yerde ve sürümlenmiş biçimde tanımlayın** (arayüz kontrol dokümanı); e-posta ve toplantı notlarıyla arayüz yönetmeye çalışmayın.
- **Türetilmiş gereksinimler için bir geri bildirim kanalı kurun.** Yazılım ekibinin gereksinim ve tasarım sırasında ürettiği türetilmiş gereksinimlerin emniyet etkisi sistem seviyesinde değerlendirilmelidir; kimin değerlendireceği ve sonucun nereye kaydedileceği baştan belli olmalıdır.
- **Belirsizliği prosedüre değil, tanıma dönüştürün.** "Bunu entegrasyonda çözeriz" cümlesi çoğu zaman "bunu en pahalı aşamada keşfedeceğiz" anlamına gelir.
- **Yazılım ekibini sistem gözden geçirmelerine dahil edin;** uygulanabilirlik sorunları en ucuz bu aşamada yakalanır.

Sonraki bölüm, bu ilişkinin emniyet tarafını — tehlikelerin nasıl belirlendiğini ve yazılım seviyesinin nasıl atandığını — ele alır.

Bu bölümden akılda kalması gerekenler

- Yazılım sistemin bir ögesi olarak anlaşılmalıdır; gereksinimleri, seviyesi ve emniyet rolü sistem süreçlerinden gelir.
- Geçerleme "doğru gereksinim", doğrulama "gereksinime uygun ürün" sorusudur; DO-178C ağırlıklı olarak doğrulamayı düzenler.
- Tahsis yazılı değilse gereksinim sahipsiz kalır; yazılımın neyi yapmadığı da neyi yaptığı kadar açık olmalıdır.
- Arayüz tanımı sürümlenmiş tek bir kaynaktan yönetilir ve her sinyal için bayatlama süresini, geçerlilik göstergesini ve geçersiz veride beklenen davranışı da söyler.
- Bilgi iki yönde akar: sistemden gereksinimler, yazılım seviyesi ve donanım tanımı gelir; yazılımdan türetilmiş gereksinimler, mimari bilgisi, problem raporları ve kullanım kısıtları döner. Bu alışveriş kayıtlı olmalıdır.
- Hangi gereksinimin yazılım düzeyinde, hangisinin sistem düzeyinde doğrulanacağı planlamada kararlaştırılır; sona bırakılan paylaşım doğrulanmamış gereksinim üretir. Sistem düzeyi faaliyetten kredi alınacaksa o faaliyet ilgili DO-178C hedeflerini karşılamalıdır.

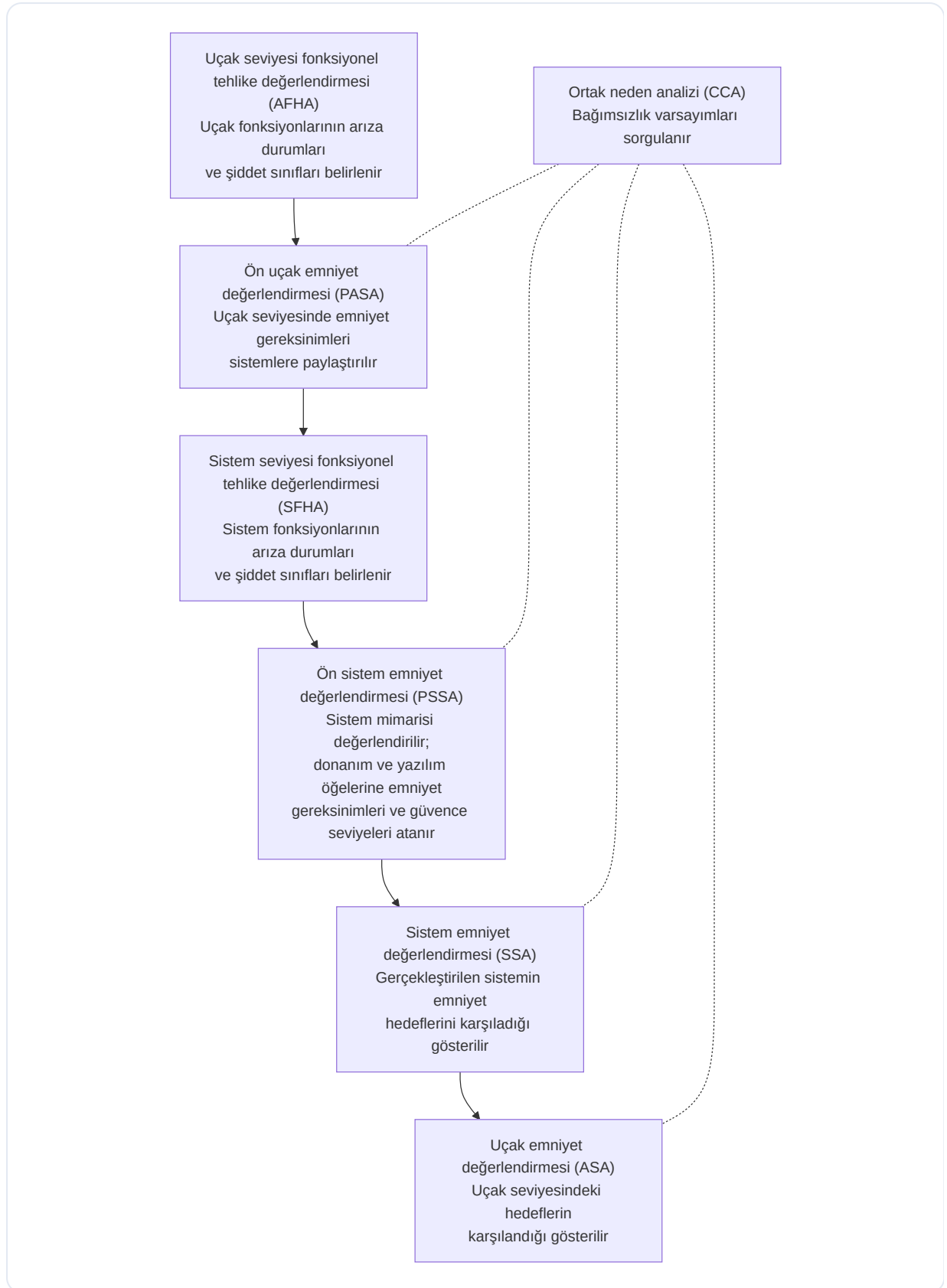
3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım

Sistem emniyet değerlendirme, bir yazılımın neden belirli bir yazılım seviyesinde geliştirilmesi gerektiğini açıklar. Bu bölüm emniyet değerlendirme adımlarını, arıza durumunun şiddetinden yazılım seviyesine giden yolu, seviyeyi etkileyen mimari önlemleri ve emniyet gereksinimlerinin yazılım gereksinimlerine ve doğrulama kanıtına nasıl dönüştüğünü anlatır.

Bu bağlantı kurulduğunda her testin ve her gözden geçirmenin arkasındaki gerekçe görünür olur: yazılım ekibi yalnızca "ne isteniyor" sorusunu değil, "neden bu sıklıkta isteniyor" sorusunu da yanıtlayabilir.

Emniyet değerlendirme sürecine genel bakış

Sivil havacılıkta emniyet değerlendirmesinin yöntemlerini SAE ARP4761A (Avrupa'daki karşılığı EUROCAE ED-135; Aralık 2023) tanımlar. Bu kılavuz, sistem geliştirme sürecini anlatan ARP4754B / ED-79B ile birlikte kullanılır (bkz. [2. Sistem Bağlamında Yazılım](#)). Emniyet değerlendirme uçak geliştirmeye paralel yürür ve sürekli geri besleme üretir. Süreç kabaca şu adımlardan oluşur:



- **Fonksiyonel tehlike değerlendirme (functional hazard assessment, FHA):** "Bu fonksiyon kaybolursa ya da yanlış çalışırsa ne olur?" sorusunu sorar ve her arıza durumuna (failure condition) bir şiddet sınıfı (severity classification) atar. Uçak ve sistem seviyelerinde ayrı ayrı

yapılır (diyagramdaki AFHA ve SFHA). Fonksiyona bakar, çözüme değil; bu yüzden tasarım daha ortada yokken başlayabilir. Aynı fonksiyonun kaybı ile fark edilmeden yanlış çalışması ayrı arıza durumlarıdır ve çoğu zaman farklı sınıflanır: mürettebatın (flight crew) fark edemediği yanıtıcı veri, genellikle verinin tümüyle kaybından daha ağır değerlendirilir.

- **Ön değerlendirmeler:** ön uçak emniyet değerlendirmesi (preliminary aircraft safety assessment, PASA) ve ön sistem emniyet değerlendirmesi (preliminary system safety assessment, PSSA), tasarım henüz tamamlanmadan, önerilen mimarinin emniyet hedeflerini karşılayıp karşılayamayacağını sorgular; emniyet gereksinimlerini ve güvence seviyelerini aşağıya, ögelere dağıtır.
- **Ortak neden analizi (common cause analysis, CCA):** yedeklilik (redundancy) ve bağımsızlık varsayımlarının geçerliliğini sorgular. Bölgesel emniyet analizi (zonal safety analysis, ZSA), belirli risk analizi (particular risk analysis, PRA) ve ortak mod analizi (common mode analysis, CMA) bu başlık altındadır. Aynı yazılımın iki yedek kanalda koşması, tek bir hata kaynağının iki kanalı birden düşürebilmesi demektir — yazılımı doğrudan ilgilendiren tipik bir ortak neden örneğidir.
- **Nihai değerlendirmeler:** sistem emniyet değerlendirmesi (system safety assessment, SSA) ve uçak emniyet değerlendirmesi (aircraft safety assessment, ASA), tasarım ve doğrulama tamamlandığında hedeflerin gerçekten karşılandığını kanıtlarla gösterir.

Bu adımlar bir kez koşulan bir zincir değil, geliştirme boyunca dönen bir çevrimdir: tasarım değiştikçe analizler güncellenir, analiz bulguları tasarımı değiştirir.

İki adlandırma notu. Birincisi, "SSA" kısaltması yalnızca nihai adımın adıdır; bu kitapta sürecin bütünü **sistem emniyet değerlendirme süreci (system safety assessment process)** diye anılır. İkincisi, kılavuzun 1996 tarihli ilk sürümünde (ARP4761) uçak seviyesindeki PASA ve ASA ayrı adımlar olarak tanımlı değildi ve FHA tek başlık altında anılıyordu; o sürüme göre yürüyen programlarda bu adlar görünmeyebilir. Güncel sürüm ayrıca, bir arızanın tümleşik sistemler boyunca yayılan etkilerini izleyen zincirleme etki analizini (cascading effects analysis, CEA) ve model tabanlı emniyet analizini (model-based safety analysis, MBSA) yöntemler arasına almıştır.

Başlıca analiz yöntemleri

Yazılım mühendisinin emniyet raporlarında en sık karşılaşacağı iki yöntem şunlardır:

- **Hata ağacı analizi (fault tree analysis, FTA):** yukarıdan aşağıya çalışır. İstenmeyen bir tepe olaydan (örneğin bir arıza durumundan) başlar ve ona yol açabilecek arıza birleşimlerini mantık kapılarıyla açar. Nicel hesap donanım arızalarının oranlarıyla yapılır; yazılım hataları ağaçta bir olasılık değeriyle değil, ilgili öğeye atanan seviyeyle karşılanır.
- **Arıza türleri ve etkileri analizi (failure modes and effects analysis, FMEA):** aşağıdan yukarıya çalışır. Her bileşenin arıza türlerinin üst düzeydeki etkisini sorar; sonuçlar arıza türleri ve etkileri özetinde (failure modes and effects summary, FMES) toplanır ve hata ağacına girdi olur.

Yazılım ekibinin bu belgeleri üretmesi beklenmez; ama kendi ögesinin ağacın neresinde durduğunu, hangi izleyicinin hangi olayı tutmak üzere varsayıldığını okuyabilmesi gerekir.

Emniyet değerlendirme ne üretir?

Her adımın yazılım ekibine ulaşan bir çıktısı vardır:

Adım	Başlıca çıktı	Yazılım ekibi için anlamı
FHA	Arıza durumları, etkileri ve şiddet sınıfları	Fonksiyonun güvence seviyesinin kaynağı; "kayıp" ile "yanlış çalışma"nın ayrı sınıflandığı yer
PASA / PSSA	Her arıza durumu için emniyet hedefi; ögelere tahsis edilen emniyet gereksinimleri, bağımsızlık gereksinimleri ve güvence seviyeleri	Yazılım seviyesi; izleme, güvenli durum ve algılama süresi gibi gereksinimler
CCA	Bağımsızlık varsayımlarının geçerli olduğunun gösterimi, ortak neden bulguları	Ortak kod, ortak girdi, ortak derleyici ya da işletim sistemi gibi paylaşımların sorgulanması
SSA / ASA	Gerçekleştirilen tasarımın emniyet hedeflerini karşıladığının kanıtı	Yazılımın atanan seviyeye uygun geliştirildiğini gösteren kanıt bu değerlendirmeye girdi olur

Bu çıktılar yazılım ekibine doğrudan "şu kodu yaz" demez. Bunun yerine hangi davranışların gösterilmesi gerektiğini ve hangi koşullarda hatanın kabul edilemeyeceğini tanımlar.

Tabloda görünmeyen ama en az onun kadar önemli bir çıktı da **analiz varsayımlarıdır**: "izleyici uyuşmazlığı bir saniye içinde algılar", "iki kanal ortak veri kullanmaz", "mürettebat uyarıyı görünce yedek göstergeye geçer" gibi. Yazılımı ilgilendiren her varsayım bir gereksinime dönüşmeli ve doğrulanmalıdır; gereksinime dönüşmemiş bir varsayım, kimsenin sahiplenmediği bir emniyet iddiasıdır.

Geliştirme güvencesi ve seviyeler

Yazılım hataları sistematik olduğundan (bkz. [1. Giriş ve Genel Bakış](#)), yazılıma donanımdaki gibi bir arıza olasılığı atanamaz. Bunun yerine **geliştirme güvencesi (development assurance)** yaklaşımı kullanılır: hatanın sonucu ne kadar ağırsa, geliştirme ve doğrulama süreci o kadar sıkı disipline edilir.

Sistem tarafında bu sıklığın ölçüsü geliştirme güvence seviyesidir (development assurance level, DAL) ve iki düzeyde atanır:

- **Fonksiyon geliştirme güvence seviyesi (function development assurance level, FDAL):** fonksiyonun FHA'daki en ağır arıza durumunun sınıfından gelir; fonksiyonun gereksinimlerinin hangi titizlikle geliştirileceğini belirler.
- **Öge geliştirme güvence seviyesi (item development assurance level, IDAL):** fonksiyonu gerçekleştiren donanım ve yazılım ögelerine, mimari dikkate alınarak atanır.

Bir yazılım ögesine atanan IDAL, DO-178C'de **yazılım seviyesi (software level)** adıyla karşılık bulur. Bu kitapta "güvence seviyesi" sözü sistem tarafındaki FDAL ve IDAL için, "yazılım seviyesi" ise DO-178C'nin Seviye A'dan Seviye E'ye uzanan beş basamağı için kullanılır.

Arıza durumunun şiddet sınıfı ile yazılım seviyesi arasındaki yerleşik eşleme şudur:

Arıza durumu şiddeti	Kısa tanım	Olasılık hedefi (uçuş saati başına ortalama)	Yazılım seviyesi
Katastrofik (catastrophic)	Çok sayıda can kaybı; çoğunlukla uçağın kaybıyla birlikte	Son derece olasılık dışı (extremely improbable); 10^{-9} mertebesinden küçük	A
Tehlikeli (hazardous)	Emniyet marjlarında büyük azalma; mürettebatın görevini eksiksiz yapmasına güvenilemeyecek ölçüde iş yükü; mürettebat dışındaki az sayıda kişide ciddi ya da ölümcül yaralanma	Son derece seyrek (extremely remote); 10^{-7} mertebesinden küçük	B
Majör (major)	Emniyet marjlarında belirgin azalma, mürettebat iş yükünde önemli artış; yolcularda ya da kabin ekibinde olası yaralanma	Seyrek (remote); 10^{-5} mertebesinden küçük	C
Minör (minor)	Emniyet marjlarında hafif azalma, rutin prosedürlerle yönetilebilir	Olası (probable); 10^{-3} mertebesinden küçük	D
Emniyet etkisi yok (no safety effect)	Emniyeti etkilemez; uçağın operasyonel yeteneğini azaltmaz, mürettebat iş yükünü artırmaz	Olasılık şartı yok	E

Tabloyu okurken üç noktaya dikkat etmek gerekir:

- **Olasılık sütunu yazılım için değildir.** Bu hedefler sistem düzeyindeki arıza durumlarına aittir ve nicel analizde rastgele donanım arızalarıyla gösterilir. Yazılıma nicel bir arıza olasılığı atanmaz; yazılımın arıza durumuna katkısı son sütundaki seviyeyle yönetilir. İki sütun, aynı şiddet sınıfının donanım ve yazılım tarafındaki karşılıklarıdır.
- **Tanımlar ve sayılar büyük nakliye uçakları içindir.** DO-178C sınıf tanımlarını bu uçak sınıfı için, örnek olarak verir; tam tanım için otoritenin kural ve rehberlerine yönlendirir. Olasılık hedefleri CS-25 / FAR Part 25 kapsamındaki uçaklar için AMC 25.1309 ve AC 25.1309'da verilen değerlerdir. Başka uçak sınıflarında sertifikasyon otoritelerinin rehberleri farklı hedefler tanımlar; hangi rehberin geçerli olduğu programın sertifikasyon temelinde belirlenir.

- **Seviyeyi en ağır katkı belirler.** Bir yazılım bileşeninin hatalı davranışı birden fazla arıza durumuna katkı verebiliyorsa, seviye bunların en ağırına göre atanır.

Seviye pratikte neyi değiştirir?

Yazılım seviyesi, DO-178C'de karşılanması gereken hedeflerin (objectives) sayısını, bu hedeflerden kaçının bağımsızlıkla (independence) karşılanacağını ve yapısal kapsam analizinin (structural coverage analysis) hangi ölçüte kadar ineceğini belirler:

Yazılım seviyesi	Hedef sayısı	Bağımsızlıkla karşılanan hedef	Yapısal kapsam ölçütü
Seviye A	71	30	Değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC), karar kapsama (decision coverage) ve satır kapsama (statement coverage)
Seviye B	69	18	Karar kapsama ve satır kapsama
Seviye C	62	5	Satır kapsama
Seviye D	26	2	Yapısal kapsam hedefi yok
Seviye E	—	—	Otorite Seviye E belirlemesini teyit ettikten sonra DO-178C rehberliği uygulanmaz

Tablo iki şeyi açıkça gösterir. Seviye A ile Seviye B arasındaki fark hedef sayısından çok bağımsızlıkta ve kapsam ölçütündedir; maliyeti asıl artıran da bunlardır. Seviye C ve Seviye D'de bağımsızlıkla işaretli hedefler ise doğrulama (verification) hedefleri değil, kalite güvencesi (quality assurance) hedefleridir. Seviye ayrıca hangi yaşam döngüsü verisinin hangi kontrol kategorisinde yönetileceğini de değiştirir. Hedeflerin yapısı [4. DO-178C ve Destekleyici Dokümanlara Genel Bakış](#) bölümünde, bağımsızlık ve kapsam ölçütleri [9. Yazılım Doğrulama](#) bölümünde ele alınır.

Seviyenin kendisi yazılım ekibinin seçimi değildir; sistem emniyet değerlendirme sürecinin çıktısıdır. Fonksiyonun seviyesi FHA'daki sınıflandırmadan gelir; ögelere tahsis, ön değerlendirmeler sırasında mimariye bakılarak yapılır. Nihai SSA'da seviye atanmaz; gerçekleştirilen sistemin hedefleri karşıladığı gösterilir. Yazılım seviyesi ve gerekçesi yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) yazılır ve otoriteyle mutabakat gerektirir (bkz. [5. Yazılım Planlama](#)). Planlama denetiminde ilk sorulanlardan biri de budur: seviye hangi arıza durumuna dayanıyor ve bu dayanak gösterilebiliyor mu ([SW SOI-1](#))?

Atanan seviye bir alt sınırdır. İleride eklenecek işlevlerin ya da değişecek tahsisin daha ağır bir arıza durumuna yol açması bekleniyorsa yazılımı baştan daha yüksek seviyede geliştirmek, kanıtı sonradan yükseltmekten kolaydır.

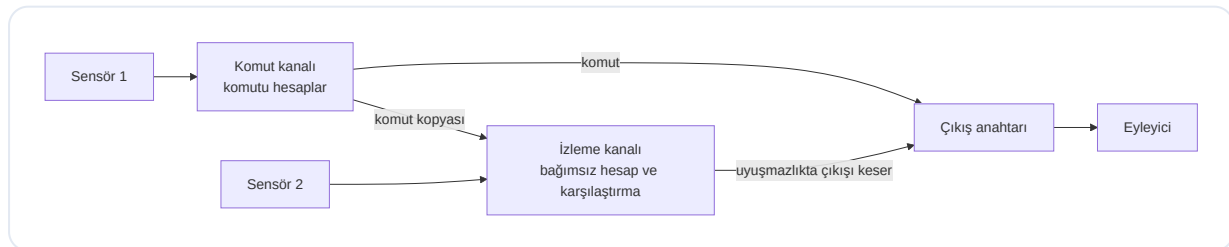
Seviyeyi etkileyen mimari önlemler

Bir fonksiyonun seviyesi ile onu gerçekleştiren her yazılım bileşeninin seviyesi aynı olmak zorunda değildir. Mimari, bir yazılım hatasının arıza durumuna katkısını sınırlıyor ya da hatayı algılayıp etkisini hafifletiyorsa, sistem süreci ilgili bileşene daha düşük bir seviye atayabilir. DO-178C, sistemle ilişkiyi anlatırken bu amaçla kullanılan üç mimari yaklaşımı anar; bunları tercih edilen ya da zorunlu çözümler olarak sunmaz. Çıkış noktası şudur: bileşenlerden herhangi birinin hatası arıza durumunu tek başına doğurabiliyorsa hepsi fonksiyonun en ağır arıza durumunun seviyesini alır. Daha düşük bir seviye ancak arıza durumu için birden çok bileşenin birlikte hatalı davranması gerekiyorsa ve bu bileşenlerin hem işlev hem tasarım (ortak tasarım öğeleri, dil, araç) açısından yeterince bağımsız olduğu gösterilebiliyorsa gündeme gelir.

Bölümleme (partitioning). Yazılım bileşenlerini birbirinden yalıtır. Her bileşene ayrı donanım vermek de bir bölümlemedir; asıl zor olan, aynı donanımı paylaşan bileşenlerin yalıtımıdır: bir bileşen diğerinin kodunu, verisini ve giriş/çıkışını bozamaz, işlemciyi de yalnızca kendisine ayrılan sürede kullanır. Yalıtım gösterilebiliyorsa her bileşen kendi katkı verdiği arıza durumuna göre seviye alır; gösterilemiyorsa hepsi aralarındaki en yüksek seviyeyle geliştirilir. Yalıtımı sağlayan yazılım, yalıtıdığı bileşenlerin en yüksek seviyesinde ya da daha yüksekinde olmalıdır; yalıtımı sağlayan donanım ise sistem emniyet değerlendirme sürecinde ayrıca değerlendirilir. Ayrıntılar [21. Yazılım Bölümlemesi](#) bölümündedir.

Çok sürümlü benzemez yazılım (multiple-version dissimilar software). Aynı fonksiyon, ortak hata kaynaklarının bir kısmından kaçınmak amacıyla birbirinden bağımsız iki ya da daha fazla sürüm olarak geliştirilir: ayrı ekipler, farklı diller, farklı derleyiciler, farklı işlemciler. Sınırı da bellidir: benzemezliğin sağladığı koruma çoğunlukla ölçülemez ve benzemezlik devreye girmeden önce tamamlanan adımlardaki bir hata — örneğin sürümlerin dayandığı ortak gereksinimdeki — her sürüme aynen geçer. Bu yüzden benzemezlik kendiliğinden kredi getirmez: DO-178C benzemez sürümleri çoğunlukla, seviyenin doğrulama hedefleri karşılandıktan sonra eklenen bir koruma katmanı olarak görür. Doğrulamanın hafifletilmesi ancak sistem emniyet değerlendirme süreci bunun doğurabileceği fonksiyon kaybını kabul edilebilir bulursa söz konusudur; ne kadar bağımsızlık sağlandığı ortak neden analizinde gösterilir ve alınacak kredi otoriteyle kararlaştırılır.

Emniyet izleme (safety monitoring). Fonksiyonun çıktısını ya da davranışını gözleyen ayrı bir izleyici, arızayı algılar ve sistemi güvenli duruma götürür. En yaygın biçimi komut/izleme (command/monitor, COM/MON) mimarisidir:



İzlemeden kredi alınabilmesi üç soruya bağlıdır:

- **İzleyicinin seviyesi nedir?** İzleyici, izlediği fonksiyonun en ağır arıza durumuna göre seviye alır. İzlenen yazılıma ise ilgili sistem fonksiyonunun kaybına karşılık gelen seviye atanabilir. İzleme yükü ortadan kaldırmaz, izleyiciye taşır; izleyicinin kendisi ucuza gelmez.
- **Neyi, ne kadar sürede algılar?** Algılanmayan arıza türleri ve algılama gecikmesi, emniyet analizindeki varsayımlarla uyumlu olmalıdır. İzleyicinin fark edilmeden arızalanması da ayrıca ele alınır; aksi hâlde koruma, gerektiği gün orada olmayabilir.
- **İzleyici izlenenden bağımsız mı?** Aynı sensör, aynı kütüphane, aynı güç kaynağı ya da aynı zaman tabanı iki kanalı birlikte yanıtlanabilir. Diyagramdaki her ortak öge bir ortak neden analizi sorusudur.

ARP4754B, bağımsız üyelerden oluşan mimarilerde ögelere fonksiyonun seviyesinden daha düşük seviye atanmasına belirli kurallarla izin verir. Kuralların ayrıntısı sistem sürecinin konusudur; yazılım ekibi için önemli olan, bu indirimin otomatik olmadığını ve hangi varsayıma dayandığını bilmektir.

⚠ SEVİYE PAZARLIĞI EMNİYET ANALİZİYLE YAPILIR, BÜTÇEYLE DEĞİL

"Seviye B pahalı, C yapalım" yaklaşımı ancak mimari bir önlem arıza etkisini gerçekten sınırılıyorsa savunulabilir. Bir ögenin seviyesinin düşürülebilmesi, başka bir ögenin (örneğin bağımsız bir izleyicinin) o arıza durumunu tuttuğunun gösterilmesine dayanır; bu bağımsızlığın kendisi de CCA ile doğrulanır. Sonradan iki kanala ortak bir sürücü ya da ortak bir veri yolu eklemek, kimse fark etmeden bu gerekçeyi ortadan kaldırabilir.

Arıza durumundan gereksinime

Emniyet değerlendirmesinde belirlenen bir arıza durumu, çoğu zaman birden fazla önlemle karşılanır ve bu önlemler farklı yerlere tahsis edilir: bir kısmı donanıma (ikinci bir sensör, ayrı bir güç kaynağı), bir kısmı yazılıma (izleme, geçerlilik denetimi, güvenli duruma geçiş), bir kısmı da operasyonel prosedüre (mürettebatın yedek göstergeye geçmesi).

Burada önemli olan, tahsisin açık olmasıdır. Hangi önlemin yazılımda, hangisinin donanımda, hangisinin prosedürde olduğu yazılı değilse iki tipik hata ortaya çıkar: ya her taraf önlemi diğerinin aldığını varsayar ve önlem sahipsiz kalır, ya da yazılım ekibi kendisinden istenmeyen bir korumayı kendiliğinden ekler ve emniyet analizinin bilmediği bir davranış doğar.

Yazılıma tahsis edilen emniyet gereksinimleri sıradan işlevsel gereksinimlerden daha fazlasını söylemek zorundadır:

- açık sınır koşulları (hangi değerden sonra veri geçersiz sayılır),
- hata durumları için net davranış (ne bildirilir, hangi çıktı kesilir),
- zamanlama (arıza ne kadar sürede algılanır, tepki ne kadar sürede verilir),
- yeniden başlatma ve kısıtlı çalışma modu (degraded mode) tanımı.

Yazılım emniyet sürecine nasıl bağlanır?

Yazılım, bazı durumlarda arıza durumunun etkisini doğrudan azaltan tek bileşen olabilir. Özellikle birden fazla sensörün tutarlılığını değerlendiren, hatalı veriyi sınırlandıran veya güvenli duruma geçişi yöneten işlevler yazılım tarafından taşınır. Bağlantının sağlıklı işlemesi için iki yönlü akış kurulmalıdır:

- **Emniyetten yazılıma:** emniyet gereksinimleri, yazılım gereksinimleri içinde ayrı ve izlenebilir (traceable) biçimde işaretlenir; kaybolmaları ya da sıradan bir işlevsel gereksinim gibi ele alınmaları en yaygın süreç hatasıdır. İşaret, gereksinim değiştiğinde emniyet ekibinin de değişikliği görmesini sağlar.
- **Yazılımdan emniyete:** yazılım geliştirme sırasında, gereksinim ya da tasarım aşamasında ortaya çıkan türetilmiş gereksinimler (derived requirements) gerekçeleriyle birlikte sistem süreçlerine ve emniyet ekibine geri bildirilir. Bunlar üst seviye gereksinime doğrudan izlenemeyen ya da orada belirtilenin ötesinde davranış tanımlayan gereksinimlerdir; sistem analizi yapılırken henüz yoktular ve emniyet etkileri değerlendirilmemişti (bkz. [6. Yazılım Gereksinimleri](#)).

Ayrıca yazılım ekibi, mimari kararlarının (bölümlendirme, izleme, yedeklilik, farklı kanallar) emniyet analizindeki varsayımlarla tutarlı kaldığını her değişiklikte yeniden sorgulamalıdır. Tasarımda bölümlendirme ya da benzeri bir mimari önlem bazı bileşenlerin seviye atamasını değiştirebilecek biçimde kullanılıyorsa DO-178C, buna ilişkin bilginin de türetilmiş gereksinim olarak tanımlanıp sistem süreçlerine iletilmesini bekler.

Doğrulama üzerindeki etkisi

Emniyet değerlendirmesi doğrulamayı iki yoldan etkiler.

Birincisi, doğrulamanın yoğunluğunu **yazılım seviyesi** belirler. DO-178C'nin doğrulama hedefleri, bileşenin seviyesine göre o bileşenin bütün gereksinimlerine ve bütün koduna uygulanır; gözden geçirme (review) ve gereksinim tabanlı test (requirements-based testing) yalnızca "kritik" görülen işlevler için değil, o seviyedeki yazılımın tamamı için beklenir. Aynı yazılımın içinde bir işlevi daha az titizlikle doğrulamak, ancak o işlev bölümlenmeyle ayrılmış ve ayrı bir seviye almışsa mümkündür.

İkincisi, emniyet gereksinimleri test ve analiz içeriğinde bazı konuları öne çıkarır:

- **Gürlüklük testleri (robustness tests):** geçersiz değerler, bozuk ya da tazelenmeyen veri, aşılacak çerçeve süresi, izin verilmeyen durum geçişi. İzleme ve koruma işlevlerinin asıl işi bu koşullarda görünür.
- **Arıza enjeksiyonu (fault injection):** izleyicinin algıladığı varsayılan arıza gerçekten üretilir ve tepkinin gereksinimdeki gibi olduğu gösterilir.

- **Zamanlama:** algılama gecikmesi ile tepki süresinin toplamı, emniyet analizinde varsayılan süreyi aşmamalıdır. Bu, en kötü durum yürütme süresi (worst-case execution time, WCET) analiziyle birlikte değerlendirilir.
- **Yanlış alarm (false alarm):** fazla dar seçilmiş bir eşik sağlam bir kanalı gereksiz yere devreden çıkarır. Kullanılabilirliği düşüren ve mürettebatın uyarılara güvenini aşındıran bu davranış da bir emniyet konusudur.

Doğrulama sonuçları emniyet sürecine geri döner: analizde varsayılan algılama süresi ya da arıza kapsamı (fault coverage), test ve analiz kanıtıyla desteklenmedikçe nihai değerlendirmede kullanılamaz.

Uçtan uca örnek: yanıltıcı hava hızı göstergesi

Aşağıdaki örnek, bir arıza durumunun yazılım gereksinimine ve kanıta nasıl dönüştüğünü gösterir. Sınıflandırma ve sayılar örnek amaçlıdır; gerçek bir programda bunlar uçağın kendi emniyet değerlendirmesinden gelir.

Halka	Örnekteki içerik
Arıza durumu (FHA)	Hava hızının, mürettebat fark etmeden yanlış gösterilmesi; göstergenin kaybı ayrı bir arıza durumudur ve ayrıca sınıflanır
Sınıflandırma	Bu örnekte Tehlikeli varsayılmıştır; fonksiyonun seviyesi FDAL B
Mimari (PSSA)	İki bağımsız hava verisi kaynağı, yazılımda karşılaştırma izleyicisi, uyumsuzlukta mürettebata uyarı
Yazılıma tahsis edilen emniyet gereksinimi	Kaynaklar arasındaki uyumsuzluk en geç 1 saniye içinde algılanıp mürettebata bildirilmelidir
Yazılım seviyesi	İzleyiciyi de içeren gösterge yazılımı için Seviye B

Yazılım ekibi bu tahsisi doğrulanabilir yüksek seviyeli gereksinimlere (high-level requirements) çevirir; aşağıda bunlar YSG örneğiyle numaralanmıştır:

- **YSG-101:** Yazılım, iki hava verisi kaynağından alınan hava hızı değerleri arasındaki mutlak fark 10 knot'ı kesintisiz 500 ms boyunca aştığında "hava hızı uyumsuzluğu" durumunu ilan etmelidir.
- **YSG-102:** Yazılım, uyumsuzluk durumu ilan edildikten sonra en geç 100 ms içinde uyumsuzluk uyarısını gösterge birimine göndermelidir.
- **YSG-103:** Kaynaklardan birinin verisi geçersiz olarak işaretlenmişse ya da 200 ms'den uzun süredir tazelenmemişse yazılım, karşılaştırmının yapılamadığını ayrı bir durum olarak bildirmelidir.

Üç gereksinim de emniyet gereksinimi olarak işaretlenir ve sistem gereksinimine izlenir. Süreler birlikte okunmalıdır: 500 ms'lik teyit süresi ile 100 ms'lik bildirim süresi, tahsis edilen 1 saniyelik bütçenin içinde kalır ve gösterge biriminin kendi gecikmesine pay bırakır. YSG-103 ise analizin çoğu zaman yazmadığı ama gerektirdiği bir şeyi söyler: izleyici çalışmıyorsa bu sessizce geçilmez.

Geliştirme sırasında bir de türetilmiş gereksinim doğar:

- **YSG-104 (türetilmiş):** Yazılım, güç verildikten sonraki ilk 2 saniye boyunca karşılaştırma yapmamalıdır. *Gerekçe:* sensör verisi oturmadan yapılan karşılaştırma yanlış alarm üretir.

Bu gereksinim hiçbir sistem gereksinimine izlenmez ve emniyet açısından masum değildir: izleyici bu pencerede kördür. Emniyet ekibine bildirildiğinde ekip, pencerenin yalnızca yerde ve kalkıştan önce oluşup oluşmadığını değerlendirir; uçuşta yeniden başlatma mümkünse gereksinimin değişmesi gerekebilir. Geri bildirim yapılmasaydı analiz, gerçekte var olmayan kesintisiz bir izlemeye dayanıyor olurdu.

Doğrulama kanıtı da aynı zinciri izler:

- gereksinimlerin, tasarımın ve kodun gözden geçirme kayıtları;
- normal aralık testleri (normal range tests): farkın eşiğin hemen altında ve hemen üstünde, sürenin 500 ms'nin hemen altında ve hemen üstünde olduğu durumlar ile açılıştaki 2 saniyelik pencerenin bitiş sınırı;
- gürbüzlük testleri: geçersiz işaretli veri, tazelenmeyen veri, aralık dışı değer ve uçuşta yeniden başlatma gibi anormal başlatma koşulları;
- Seviye B'nin gerektirdiği yapısal kapsam: satır kapsama ve karar kapsama ile veri bağlaşımı ve kontrol bağlaşımı (data coupling, control coupling) analizi;
- arıza durumundan sistem gereksinimine, oradan yazılım gereksinimine, koda ve test durumlarına uzanan izlenebilirlik kaydı.

Bu bölümden akılda kalması gerekenler

- Emniyet değerlendirme adımları (FHA → PASA/PSSA → SSA/ASA ve bunlara eşlik eden CCA), yazılıma tahsis edilen emniyet gereksinimlerinin ve yazılım seviyesinin kaynağıdır.
- Fonksiyonun seviyesi (FDAL) FHA'daki şiddet sınıfından gelir; ögenin seviyesi (IDAL) mimari dikkate alınarak atanır. Yazılım ögesine atanan IDAL, DO-178C'deki yazılım seviyesidir.
- Olasılık hedefleri sistem düzeyindeki arıza durumları içindir; yazılıma olasılık atanmaz, yazılımın katkısı seviyeyle yönetilir.
- Seviye; hedef sayısını, bağımsızlık beklentisini ve yapısal kapsam ölçütünü belirler ve bileşenin tamamına uygulanır.
- Bölümleme, çok sürümlü benzemez yazılım ve emniyet izleme seviyeyi etkileyebilir; ancak bunun dayandığı bağımsızlık ortak neden analiziyle gösterilmelidir.

- Yazılımın emniyet katkısı açıkça tahsis edilmeli; emniyet gereksinimleri işaretlenmeli, türetilmiş gereksinimler gerekçeleriyle sistem süreçlerine ve emniyet ekibine geri bildirilmelidir.
- Emniyet analizindeki her varsayım (algılama süresi, arıza kapsamı, bağımsızlık) bir gereksinime ve doğrulama kanıtına bağlanmalıdır.

KISIM III

DO-178C ile Emniyet- Kritik Yazılım Geliştirme

4. DO-178C ve Destekleyici Dokümanlara Genel Bakış

DO-178C, aviyonik yazılım için nasıl geliştirme yapılacağını adım adım emreden bir tarif kitabı değildir; bunun yerine, emniyet-kritik yazılımın doğrulanabilir, izlenebilir ve sertifikasyona uygun biçimde geliştirilmesi için bir çerçeve sunar. Bu bölüm standardın tarihçesini, süreç ve hedef yapısını, yazılım seviyelerini, çevresindeki belge ailesini ve düzenleyici konumunu özetleyerek sonraki bölümlerin zeminini kurar.

Belgeyi okurken amaç, "ne yapılmalı?" sorusundan çok "hangi çıktılar gösterilmeli?" sorusuna odaklanmaktır. DO-178C'nin değeri, proje ekibine yalnızca teknik yönerge vermesinde değil, aynı zamanda sertifikasyon otoritesinin (certification authority) beklediği kanıt dilini ortaklaştırmasında yatar. Belge; yazılım yaşam döngüsü süreçlerini, her sürecin hedeflerini (objective), üretilmesi beklenen veriyi, gözden geçirme beklentilerini ve her seviyede hangi hedeflerin karşılanması gerektiğini tarif eder. Faaliyetlerin sırasını ise dayatmaz: hangi yaşam döngüsü modelinin izleneceği ve bir süreçten ötekine hangi koşulla geçileceği, projenin kendi planlarında tanımladığı geçiş kriterlerine (transition criteria) bırakılır. Bu kriterlerin nasıl yazılabileceğini [Ek A: Örnek Geçiş Kriterleri](#) gösterir.

DO-178'in kısa tarihçesi

1970'lerin sonuna gelindiğinde yazılım, uçak sistemlerinde artık yardımcı bir unsur olmaktan çıkıp uçuş fonksiyonlarının doğrudan parçası olmaya başlamıştı. Donanım için yerleşmiş güvenilirlik yöntemleri (arıza oranı hesapları, yedekleme mimarileri) yazılıma birebir uygulanamıyordu; çünkü yazılım "eskimez", hataları tasarım kaynaklıdır ve istatistiksel arıza oranı kavramı ona iyi oturmaz. Endüstri ile otoritelerin ortak bir dile ihtiyacı vardı. Bu ihtiyaçla RTCA (ABD tarafında) ve EUROCAE (Avrupa tarafında) ortak komiteler kurdu; ortaya çıkan belgeler bu yüzden çift numara taşır (örneğin DO-178C / ED-12C).

Gelişimi dört ana durak üzerinden özetlemek mümkündür:

Sürüm	Yıl	Ana katkısı
DO-178	1982	İlk ortak çerçeve; yazılımın kritikliğine göre kaba bir sınıflandırma
DO-178A	1985	Kademeli yazılım seviyeleri ve daha belirgin süreç/faaliyet tanımları
DO-178B	1992	Hedef tabanlı (objective-based) yaklaşım; A–E seviyeleri; hedef tabloları
DO-178C	2011	Netleştirilmiş metin; araç kalifikasyonu belgesi (DO-330) ve üç teknoloji eki (DO-331/332/333) ile modüler yapı

DO-178 (1982), konunun ilk kez ortak bir belgeye bağlanması açısından önemliydi; ancak içerik büyük ölçüde nitelikseldi ve "iyi mühendislik pratiği" düzeyinde kalıyordu. Yazılım, uçuş emniyetine etkisine göre yalnızca birkaç kaba kategoriye ayrılıyor, hangi kanıtın yeterli sayılacağı büyük ölçüde projeye ve otoriteye bırakılıyordu.

DO-178A (1985), ilk deneyimlerin ışığında yazılımı kademeli seviyelere ayırma fikrini belirginleştirdi ve geliştirme ile doğrulama faaliyetlerini daha somut tanımladı. Yine de belge, faaliyetleri belirli geliştirme yaklaşımlarına bağlı biçimde anlatıyordu; bu da farklı yöntem kullanan projelerde yorum farklarına yol açıyordu.

DO-178B (1992), bugün bildiğimiz yapının kurulduğu asıl kırılma noktasıdır. Belge, "şu faaliyeti şöyle yap" demek yerine "şu hedefin sağlandığını göster" diyen hedef tabanlı yaklaşıma geçti. Yazılım seviyeleri A'dan E'ye netleşti; her seviye için hangi hedeflerin geçerli olduğu ve hangilerinin bağımsız kişilerce doğrulanması gerektiği tablolara bağlandı. Yapısal kapsam (structural coverage) beklentileri de bu sürümle kademelendi: en kritik seviyede değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC) dahil olmak üzere seviyeye göre artan kapsam ölçütleri tanımlandı. DO-178B yaklaşık yirmi yıl boyunca fiilen tüm sivil aviyonik yazılım projelerinin ortak referansı oldu.

Bu uzun kullanım süresi, güncelleme ihtiyacını da biriktirdi. **DO-178C (2011)** çalışmasının başlıca gerekçeleri şunlardı:

- DO-178B metnindeki bazı ifadeler farklı yorumlanabiliyordu; yıllar içinde biriken sık sorulan sorular ve açıklama yazıları (DO-248 serisi) bu belirsizliklerin giderilmesi gerektiğini gösteriyordu.
- Model tabanlı geliştirme (model-based development), nesne yönelimli teknoloji (object-oriented technology) ve biçimsel yöntemler (formal methods) gibi teknikler yaygınlaşmıştı; 1992 tarihli metin bunların nasıl ele alınacağını söylemiyordu.
- Araç kalifikasyonu (tool qualification), DO-178B içindeki kısa bölüme sığmayacak kadar büyümüştü; hem yazılım hem donanım projelerinde kullanılabilecek bağımsız bir belgeye ihtiyaç vardı.
- Hedefler, faaliyetler ve üretilen veri arasındaki eşleme yer yer tutarsızdı; bazı beklentiler metinde geçtiği hâlde hedef tablolarında görünmüyordu. Terimlerin ve tabloların hizalanması gerekiyordu.

2005–2011 arasında yürütülen ortak komite çalışması (RTCA SC-205 / EUROCAE WG-71), bilinçli bir tercihle DO-178B'nin çekirdeğini korudu: temel süreçler, seviyeler ve hedef mantığı değişmedi. Yenilik; metnin netleştirilmesi, teknolojiye özgü konuların üç ayrı teknoloji ekine (supplement: DO-331, DO-332, DO-333) ve araç kalifikasyonunun bağımsız bir belgeye (DO-330) taşınmasıydı. Bu modüler yapı sayesinde ana standart sık sık değişmek zorunda kalmadan, yeni teknikler kendi ekleri üzerinden ele alınabilir hâle geldi.

DO-178B'den DO-178C'ye ne değişti?

Çekirdek aynı kaldığı için fark ilk bakışta küçük görünür; DO-178B bilen bir okurun gündelik işine dokunan değişiklikler ise şunlardır:

- **Araç kalifikasyonu yeniden kuruldu.** "Geliştirme aracı / doğrulama aracı" ikiliğinin yerini, aracın sürece etkisine bakan üç ölçüt ve beş kademeli araç kalifikasyon seviyesi (tool qualification level, TQL) aldı; ayrıntı DO-330'a taşındı (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)).
- **Parametre verisi ögesi (parameter data item, PDI) tanımlandı.** Çalıştırılabilir nesne kodunu değiştirmeden yazılımın davranışını etkileyen veri dosyaları, adı konmuş bir yaşam döngüsü verisi oldu ve doğrulanması açıkça beklenir hâle geldi (bkz. [22. Konfigürasyon Verisi](#)).
- **İz verisi (trace data) ayrı bir veri ögesi oldu.** Gereksinim, tasarım, kod ve doğrulama arasındaki izlenebilirlik (traceability) bağları, başka belgelerin içine dağılmış bir özellik olmaktan çıkıp kendi başına üretilen ve denetlenen bir veri sayıldı.
- **Örtük beklentiler açık hedefe dönüştü.** DO-178B'nin metninde geçen ama hedef tablolarında yer almayan iki beklenti tablolara taşındı. Biri Seviye A'ya özgüdür: derleyicinin ürettiği ve kaynak koda doğrudan izlenemeyen nesne kodunun belirlenip doğrulanması artık ayrı bir hedeftir. Öteki kalite güvencesindedir: planların ve standartların geliştirilip tutarlılık açısından gözden geçirildiğine dair güvence. Bu iki hedefin, parametre verisi için eklenen iki doğrulama hedefinin ve tablolardaki yeniden düzenlemenin sonucunda Seviye A'nın hedef sayısı 66'dan 71'e çıktı.
- **Hedef tabloları faaliyetlere bağlandı.** Tablolara, her hedefi destekleyen faaliyetlerin atfı eklendi; "bu hedef için hangi iş öngörülüyor?" sorusu artık tablonun kendisinden yanıtlanabilir.
- **Dar ama etkili netleştirmeler yapıldı.** MC/DC tanımı genişledi: DO-178B'nin yorumu olan benzersiz neden MC/DC'nin (unique-cause MC/DC) yanında maskeleyen MC/DC (masking MC/DC) ve kısa devre değerlendirmesi (short-circuit evaluation) de kabul edilir. Türetilmiş gereksinimlerin de yalnızca emniyet değerlendirme sürecine değil, onu da içeren sistem süreçlerine iletilmesi istenir.

Pratik sonucu şudur: DO-178B ile deneyimi olan bir ekip DO-178C'ye geçerken süreçlerini baştan kurmaz; ama araç listesini yeni ölçütlerle yeniden sınıflandırır, iz verisini ve parametre verisini planlarına yazar ve kullandığı teknikler bir ekin kapsamına giriyorsa o ekin hedeflerini de plan setine dahil eder.

DO-178C neyi kapsar?

DO-178C'nin konusu; uçakta, motorda, pervanede ve bölgeye göre yardımcı güç ünitesinde kullanılan sistem ve ekipmanların yazılımıdır. Standart bu yazılımın yaşam döngüsünü üç süreç grubuyla ele alır:

- **Planlama süreci (planning process):** planların ve standartların hazırlanması.
- **Geliştirme süreçleri (development processes):** gereksinim, tasarım, kodlama ve entegrasyon (integration).
- **Bütünleyici süreçler (integral processes):** doğrulama (verification), konfigürasyon yönetimi (configuration management), kalite güvencesi (quality assurance) ve sertifikasyon irtibatı (certification liaison).

Bu süreçlerin üzerine bir de hedefler yerleştirilir. Yani standart yalnızca görevleri sıralamaz; her görevin sonunda neyin gösterilmiş olması gerektiğini de söyler.

Metnin düzeni de aynı gruplamayı izler. Giriş, sistemle ilişki ve yaşam döngüsü kavramlarından sonra planlama sürecine, dört geliştirme sürecine birlikte ve bütünleyici süreçlerin her birine ayrı birer kısım ayrılır; ardından sertifikasyon sürecine bilgi amaçlı kısa bir bakış, yaşam döngüsü verisinin içeriği ve "ek hususlar" (önceden geliştirilmiş yazılım, araç kalifikasyonu, alternatif yöntemler) gelir. Hedeflerin seviyelere göre dökümü, belgenin sonundaki Ek A (Annex A) tablolarındadır. Standartı ilk kez eline alan biri için pratik okuma sırası tersinedir: önce Ek A'da ilgili tabloyu bulmak, sonra tablonun gösterdiği metin kısmına gitmek. Tablolar yine de tek başına bir kontrol listesi değildir; standart, uyumun her yönünü yansıtmadıklarını ve metnin bütünüyle birlikte okunmaları gerektiğini kendisi belirtir.

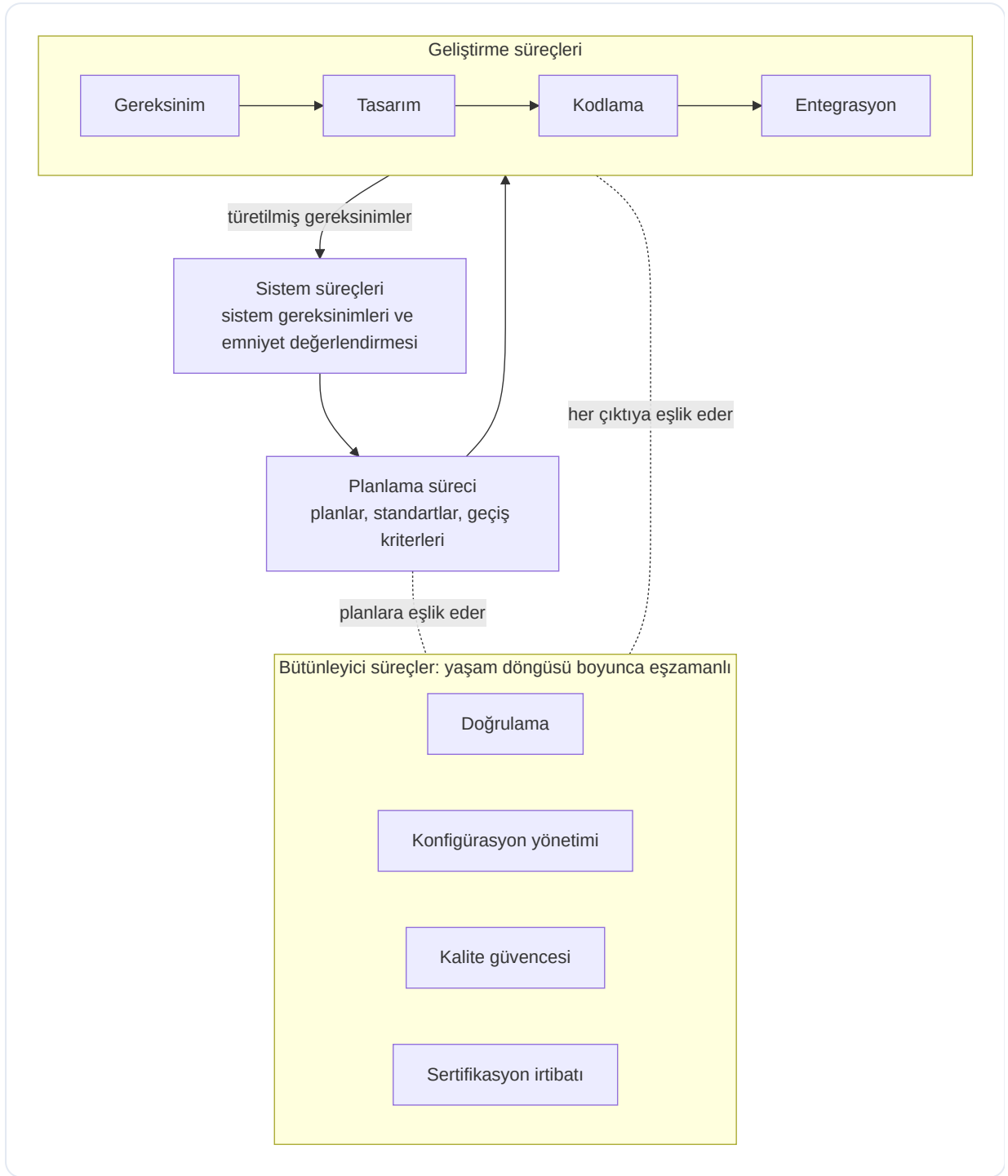
Neyin kapsam **dışında** kaldığını bilmek de aynı ölçüde önemlidir:

- Sistem gereksinimlerinin tanımlanması ve geçerlenmesi (validation), emniyet değerlendirmesi ve yazılım seviyesinin atanması sistem süreçlerinin işidir (ARP4754B ve ARP4761A). DO-178C bu süreçlerle bilgi alışverişini tarif eder, onların yerini tutmaz.
- Elektronik donanımın geliştirme güvencesi ayrı bir belgenin, DO-254'ün konusudur.
- Standart belirli bir programlama dili, yöntem, araç ya da yaşam döngüsü modeli önermez; seçim projeye aittir ve planlarda gerekçelendirilir.
- Başvuru sahibinin organizasyon yapısı, tedarikçileriyle ticari ilişkisi ve personel yeterlilik ölçütleri de standardın konusu değildir; otoritenin projeye ne ölçüde katılacağını da standart belirlemez.

Yazılımın bu sistem çerçevesindeki yeri [2. Sistem Bağlamında Yazılım](#) bölümünde anlatılmıştı.

Süreç bakışı

Üç süreç grubunun birbirine göre konumu aşağıdaki gibidir. Düz oklar bilgi akışını, kesikli çizgiler ise bir sırayı değil, eşlik ilişkisini gösterir.



Bu görünümde dikkat edilmesi gereken iki nokta vardır.

Birincisi, bütünleyici süreçler geliştirmenin **ardından gelen adımlar değildir**; planlama dahil yaşam döngüsünün tamamına eşlik eder. Planlar da gözden geçirilir, konfigürasyon kontrolüne alınır ve kalite güvencesinin denetiminden geçer; otoriteyle irtibat ilk planla başlar. Doğrulamayı "kod bittikten sonra yapılan test" olarak çizen şemalar bu yüzden yanıltıcıdır: gereksinim yazıldığında gereksinim gözden geçirmesi, tasarım çıktığında tasarım gözden geçirmesi yapılır.

İkincisi, geliştirme süreçleri art arda çizilse de akış tek yönlü değildir. Gereksinimlerdeki bir değişiklik tasarımı, kodlamayı ve doğrulamayı etkiler; doğrulama bulguları ise planları ve hatta gereksinim dilini geri besleyebilir. Geliştirme sırasında ortaya çıkan türetilmiş gereksinimler (derived requirement) sistem süreçlerine geri bildirilir; çünkü emniyet değerlendirmesinin hesaba katmadığı bir davranış tanımlıyor olabilirler. Standart bu döngülerin hangi sırayla işleyeceğini belirlemez: şelale, artımlı ya da yinelemeli bir model seçilebilir. Seçilen model ve geçiş kriterleri planlara yazılır; proje de kendi yazdığı bu kurallara göre denetlenir.

Hedef, faaliyet ve yaşam döngüsü verisi

DO-178C'yi okumanın anahtarı, üç kavram arasındaki ilişkidir:

- **Hedef:** bir sürecin sonunda gösterilmiş olması gereken şeydir. Uyum, hedefler üzerinden değerlendirilir.
- **Faaliyet (activity):** hedefe ulaşmak için standardın tarif ettiği iştir. Farklı bir yol izlenecekse bu, planlarda tanımlanır ve otoritenin onayına bağlıdır.
- **Yazılım yaşam döngüsü verisi (software life cycle data):** faaliyetin çıktısı ve hedefin sağlandığının kanıtıdır. Gündelik dilde "iş ürünü" de denir.

Bir örnek ilişkiyi somutlaştırır. Hedef, düşük seviyeli gereksinimlerin (low-level requirement) yüksek seviyeli gereksinimlerle (high-level requirement) uyumlu olmasıdır. Faaliyet, tasarım verisinin gözden geçirilmesi ve analizidir. Veri ise doğrulama sonuçlarıdır: hangi gereksinim kümesinin hangi sürümünün neye karşı gözden geçirildiğini ve bulguların nasıl kapandığını gösteren kayıt. Denetçi "gözden geçirme yaptınız mı?" diye değil, "bu kayıt hedefin sağlandığını gösteriyor mu?" diye bakar; işaretlenmiş ama neye bakıldığını söylemeyen bir kontrol listesi faaliyetin yapıldığını gösterir, hedefin karşılandığını göstermez.

Hedefler Ek A'daki on tabloda toplanmıştır. Her satır bir hedeftir ve dört soruyu yanıtlar: hedef hangi seviyelerde geçerlidir, hangilerinde bağımsızlıkla (independence) karşılanmalıdır, kanıtı hangi veridir ve bu veri hangi kontrol kategorisinde (control category) yönetilir. Kontrol kategorileri, verinin ne kadar sıkı bir konfigürasyon yönetimine tabi olacağını belirler ve [10.](#)

[Yazılım Konfigürasyon Yönetimi](#) bölümünde açıklanır.

Tablo	Konusu	Kitapta
A-1	Yazılım planlama süreci	Bölüm 5
A-2	Yazılım geliştirme süreçleri	Bölüm 6, 7 ve 8
A-3	Gereksinim sürecinin çıktılarının doğrulanması	Bölüm 6 ve 9
A-4	Tasarım sürecinin çıktılarının doğrulanması	Bölüm 7 ve 9
A-5	Kodlama ve entegrasyon süreçlerinin çıktılarının doğrulanması	Bölüm 8 ve 9

Tablo	Konusu	Kitapta
A-6	Entegrasyon sürecinin çıktıların test edilmesi	Bölüm 9
A-7	Doğrulama sürecinin sonuçlarının doğrulanması (test kapsamı ve yapısal kapsam dahil)	Bölüm 9 ve 17
A-8	Yazılım konfigürasyon yönetimi süreci	Bölüm 10
A-9	Yazılım kalite güvencesi süreci	Bölüm 11
A-10	Sertifikasyon irtibatı süreci	Bölüm 12

Tablonun dağılımı standardın ağırlık merkezini de gösterir: on tablonun beşi doğrulamaya ayrılmıştır. Geliştirme süreçleri tek tabloya sığarken, o süreçlerin çıktıların doğru olduğunu göstermek dört tablo, doğrulamanın kendisinin yeterli olduğunu göstermek ise ayrı bir tablo tutar.

Yazılım seviyesi

DO-178C, yazılımı uçuş emniyetine etkisine göre beş yazılım seviyesine (software level) ayırır: Seviye A'dan Seviye E'ye. Seviye, yazılımın katkıda bulunabileceği en ağır arıza durumunun (failure condition) şiddetine göre sistem emniyet değerlendirme sürecinde belirlenir; yazılım ekibinin seçimi değildir. Eşlemenin nasıl kurulduğu [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#) bölümünde anlatılmıştı. Uçuş emniyetine etki arttıkça beklentiler sıkılaşır: daha fazla hedef, daha çok bağımsızlık ve daha derin doğrulama gerekir.

Yazılım seviyesi	İlişkili arıza durumu şiddeti	Hedef sayısı	Bağımsızlıkla karşılanan hedef
Seviye A	Katastrofik	71	30
Seviye B	Tehlikeli	69	18
Seviye C	Majör	62	5
Seviye D	Minör	26	2
Seviye E	Emniyet etkisi yok	Otorite teyidinden sonra hedef uygulanmaz	—

Tablodaki sayılar birkaç şeyi birden söyler:

- **A ile B arasındaki fark sayıda küçük, işte büyüktür.** Hedef sayısı yalnızca iki azalır: MC/DC ile kaynak koda doğrudan izlenemeyen nesne kodunun doğrulanması Seviye A'ya özgüdür. Asıl fark bağımsızlıktadır; bağımsızlıkla karşılanması gereken hedef sayısı 30'dan 18'e iner.
- **Yapısal kapsam seviyeyle kademelenir.** Satır kapsama (statement coverage) A, B ve C'de; karar kapsama (decision coverage) A ve B'de; MC/DC yalnızca A'da aranır. Seviye D'de yapısal kapsam hedefi yoktur. Ayrıntı [9. Yazılım Doğrulama](#) bölümündedir.

- **C ve D'de bağımsızlık, kalite güvencesinin bağımsızlığıdır.** Bu seviyelerde bağımsızlıkla işaretli hedefler kalite güvencesi hedefleridir; doğrulama hedeflerinde bağımsızlık A ve B'de aranır.
- **Seviye E muafiyet değil, gösterilmesi gereken bir sonuçtur.** DO-178C hedefleri uygulanmaz; ama yazılımın gerçekten emniyet etkisi taşımadığı emniyet değerlendirmesiyle gösterilmiş ve otoritece teyit edilmiş olmalıdır.

Seviye bu yüzden yalnızca bir etiket değildir; hangi verinin gerekli olduğunu, hangi doğrulama derinliğinin beklendiğini, hangi hedeflerde bağımsızlık aranacağını ve denetimde hangi kanıtın özellikle sorgulanacağını belirleyen çerçevedir.

Sahada aynı kavram için sıkça "DAL" dendiği duyulur. Geliştirme güvence seviyesi (development assurance level, DAL) sistem tarafının terimidir: fonksiyona fonksiyon geliştirme güvence seviyesi (FDAL), ögeye öge geliştirme güvence seviyesi (IDAL) atanır. Yazılım ögesine atanan IDAL, DO-178C'de yazılım seviyesi olarak karşılık bulur. Bu kitapta DO-178C bağlamında hep "yazılım seviyesi" denir.

Uyumdan çok kanıt

DO-178C'nin en önemli zihinsel modeli şudur: amaç yalnızca "yazılımı geliştirmek" değil, belirli hedeflerin sağlandığını gösterecek kanıtı üretmektir. Bu kanıt, yazılım yaşam döngüsü verisinin tamamıdır: planlar, standartlar, gereksinimler, tasarım tanımı, kaynak kod, doğrulama durumları ve sonuçları, gözden geçirme kayıtları, kapsam sonuçları ve problem raporları (problem report).

Bu verinin hepsi otoriteye gönderilmez. Asgari olarak üç belge sunulur: yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC), yazılım konfigürasyon indeksi (Software Configuration Index, SCI) ve yazılım başarı özeti (Software Accomplishment Summary, SAS). Geri kalanı da üretilir ve kontrol altında tutulur; talep edildiğinde ve denetimlerde gösterilir.

Bu yaklaşımın sonucu olarak:

- eksik izlenebilirlik bir kalite kusuru değil, doğrudan bir sertifikasyon problemi hâline gelir,
- test faaliyetleri sadece hata bulmak için değil, hedefleri göstermek için de yürütülür,
- bağımsızlık beklentisi rol ve görev dağılımını etkiler: bağımsızlığın arandığı hedeflerde doğrulayan kişi doğruladığı ürünün yazarı olamaz, ama bunun için ayrı bir organizasyon birimi şart değildir (Bölüm 9),
- her planlama kararı, sonraki iş ürünlerinin üretim biçimini değiştirir.

Belge ailesi

DO-178C çoğu projede tek başına okunmaz; kullanılan tekniğe göre aşağıdaki belgelerden biri ya da birkaçı devreye girer. Model, nesne yönelimli teknik, biçimsel yöntem ya da kalifiye araç kullanmayan bir projede ise ana belge yeterlidir. Belgelerin tümü RTCA tarafından 2011'de yayımlanmıştır; EUROCAE karşılıkları aynı içeriği taşır.

Belge	EUROCAE karşılığı	Niteliği	Ne zaman devreye girer?	Kitapta
DO-178C	ED-12C	Ana çerçeve: süreçler, hedefler ve yaşam döngüsü verisi	Her uçak yazılımı projesinde	Bölüm 5–12
DO-330	ED-215	Araç kalifikasyonu belgesi; teknoloji eki değildir	Bir araç bir süreci kaldırıyor, azaltıyor ya da otomatikleştiriyor ve çıktısı ayrıca doğrulanmıyorsa	13. DO-330 ve Yazılım Aracı Kalifikasyonu
DO-331	ED-218	Teknoloji eki: model tabanlı geliştirme ve doğrulama (model-based development and verification)	Gereksinim ya da tasarım bir modelle ifade ediliyorsa	14. DO-331 ve Model Tabanlı Geliştirme ve Doğrulama
DO-332	ED-217	Teknoloji eki: nesne yönelimli teknoloji ve ilgili teknikler	Kalıtım, çok biçimlilik, istisna işleme ya da dinamik bellek yönetimi gibi teknikler kullanılıyorsa	15. DO-332 ve Nesne Yönelimli Teknoloji ve İlgili Teknikler
DO-333	ED-216	Teknoloji eki: biçimsel yöntemler	Doğrulama kanıtının bir bölümü biçimsel analizle üretiliyorsa	16. DO-333 ve Biçimsel Yöntemler
DO-248C	ED-94C	Destekleyici bilgi: sık sorulan sorular, tartışma yazıları ve gerekçeler; yeni şart koymaz	Bir hedefin ya da ifadenin nasıl yorumlanacağı tartışıldığında	—
DO-278A	ED-109A	Yer tabanlı haberleşme, seyrüsefer, gözetim ve hava trafik yönetimi (CNS/ATM) yazılımı için DO-178C'nin paralel belgesi	Yazılım uçakta değil, yer sisteminde çalışıyorsa	—

Tablodaki iki ayrım sık karıştırılır. İlki, **teknoloji eki (supplement)** ile bağımsız belge ayrımıdır. DO-331, DO-332 ve DO-333 tek başına okunmaz; ana belgenin hedeflerine ekleme yapar, bazılarını değiştirir ya da tekniğe uyarlar. DO-330 ise bir ek değildir: kendi hedefleri, faaliyetleri ve veri kümesi olan, havacılık dışındaki alanlarda da kullanılabilecek biçimde yazılmış ayrı bir belgedir. İkincisi sözcükle ilgilidir: bu kitapta "teknoloji eki" ayrı yayımlanan bu üç belgeyi, "Ek A" ise DO-178C'nin kendi içindeki hedef tablolarını anlatır.

Tamamlayıcı belgeler, ana standardın ayrıntıya girmedeği alanları doldurur. Örneğin model tabanlı bir akış kullanılıyorsa modelin gereksinim mi tasarım mı sayılacağını ve hangi ek hedeflerin doğacağını DO-331 söyler; bir analiz aracının çıktısına ayrıca doğrulamadan güveniliyorsa o güveni temellendirmek için DO-330 devreye girer.

Ailenin dışında kalan ama her projede karşılaşılan komşu belgeler de vardır: sistem geliştirme için ARP4754B / ED-79B ve emniyet değerlendirmesi için ARP4761A / ED-135 (ikisi de 2023), elektronik donanım için DO-254 / ED-80 (2000).

Standardın hukuki konumu

DO-178C bir yasa ya da yönetmelik değildir; RTCA ve EUROCAE çatısı altında endüstri ile otoritelerin birlikte yazdığı bir uzlaşi belgesidir. Uçuşa elverişlilik (airworthiness) kuralları sistemlerin emniyetli olmasını ister, yazılımın nasıl geliştirileceğini söylemez. Aradaki bağı otoritelerin rehber dokümanları kurar: FAA'nın AC 20-115D ve EASA'nın AMC 20-115D dokümanları (ikisi de 2017), DO-178C'yi DO-330 ve üç teknoloji ekiyle birlikte **kabul edilebilir uyum yöntemi (acceptable means of compliance)** olarak tanır.

Bu konumun pratikte üç sonucu vardır:

- **Tek yol değildir, ama fiilen ortak yoldur.** Başvuru sahibi başka bir yöntem önerebilir; ancak o yöntemin eşdeğer güvence sağladığını göstermek kendisine düşer. Pratikte projeler bu yükü almak yerine DO-178C'yi seçer.
- **Bağlayıcılık projenin kendi beyanından doğar.** Uyum yöntemi olarak DO-178C'nin seçildiği PSAC'ta yazılır ve otoriteyle mutabakata bağlanır; o andan sonra proje standarda ve kendi planlarına göre denetlenir.
- **Yazılım tek başına sertifikalandırılmaz.** Onay uçağa, motora ya da ekipmana verilir; yazılım kanıtı bu onayın bir parçasıdır. "DO-178C sertifikalı yazılım" ifadesi bu yüzden gevşek bir kısaltmadır: kastedilen, belirli bir sistem ve seviye için DO-178C hedeflerine uyumun otoriteye gösterilmiş olmasıdır.

Rehber dokümanlar yalnızca standardı tanımakla kalmaz. AC 20-115D ve AMC 20-115D, önceki sürümlerle onaylanmış yazılımın değiştirilmesi gibi geçiş konularını da ele alır. Belirli konular için ayrı dokümanlar yayımlanmıştır: açık problem raporlarının yönetimi için AC 20-189 / AMC 20-189, çok çekirdekli işlemciler için AC 20-193 / AMC 20-193. FAA'nın AC 00-69 dokümanı ile DO-248C ise yeni şart koymaz; iyi uygulamaları ve gerekçeleri açıklar. Katılım aşaması (Stage of Involvement, SOI) denetimleri de DO-178C metninde tanımlı değildir; otoritenin gözetimini nasıl yürüttüğüne ilişkin bir uygulamadır ve [12. Sertifikasyon İrtibatı](#) bölümünde ele alınır.

Sonraki bölümler

Kısım III'ün geri kalanı, bu bölümde özetlenen süreçleri tek tek açar:

- [5. Yazılım Planlama](#)
- [6. Yazılım Gereksinimleri](#)
- [7. Yazılım Tasarımı](#)
- [8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#)
- [9. Yazılım Doğrulama](#)
- [10. Yazılım Konfigürasyon Yönetimi](#)
- [11. Yazılım Kalite Güvencesi](#)
- [12. Sertifikasyon İrtibatı](#)

Bu sıralama bir süreç akışı değildir; standardın mantığını anlaşılır bir sırada açmak için seçilmiştir. Önce planları, sonra geliştirme süreçlerinin ürettiği veriyi, ardından bu verinin nasıl doğrulandığını ve kontrol altında tutulduğunu ele almak, belgenin neden bu kadar sıkı bir izlenebilirlik beklediğini anlamayı kolaylaştırır. Bütünleyici süreçlerin kitapta sonda yer alması, projede sonda başladıkları anlamına gelmez. Araç kalifikasyonu ve teknoloji ekleri ise Kısım IV'te, 13–16. bölümlerde işlenir.

Bu bölümden akılda kalması gerekenler

- DO-178C bir "nasıl kod yazılır" standardı değildir; süreçleri, hedefleri ve beklenen veriyi tanımlar, faaliyet sırasını ve yaşam döngüsü modelini projenin planlarına bırakır.
- Hedef tabanlı yaklaşım ve seviye mantığı DO-178B ile kuruldu; DO-178C bu çekirdeği korudu, metni netleştirdi, teknolojiye özgü konuları üç teknoloji ekine (DO-331, DO-332, DO-333), araç kalifikasyonunu bağımsız bir belgeye (DO-330) taşıdı.
- Doğrulama, konfigürasyon yönetimi, kalite güvencesi ve sertifikasyon irtibatı bütünleyici süreçlerdir: geliştirmenin ardından gelmez, planlamadan itibaren ona eşlik eder.
- Uyum, Ek A tablolarındaki hedefler üzerinden değerlendirilir; her hedefin bir faaliyeti ve kanıtı olan bir yaşam döngüsü verisi vardır.
- Yazılım seviyesi emniyet değerlendirmesinden gelir ve hedef sayısını (A: 71, B: 69, C: 62, D: 26), bağımsızlığı ve yapısal kapsam derinliğini belirler.
- DO-178C yasa değildir; AC 20-115D ve AMC 20-115D ile kabul edilebilir uyum yöntemi olarak tanınır ve projeyi PSAC'taki beyan üzerinden bağlar.
- Kanıt üretmek, yazılım geliştirme kadar önemli bir iştir; kullanılan teknik bir ekin ya da DO-330'un kapsamına giriyorsa onun hedefleri de plan setine dahil edilir.

5. Yazılım Planlama

Yazılım planlama, projenin nasıl yürütüleceğini ve hangi kanıtların üretileceğini iş başlamadan tanımlar. Bu bölüm beş yazılım planını, üç geliştirme standardını, geçiş kriterlerini, yaşam döngüsü ortamını ve araç kalifikasyonu kararlarını ele alır.

Planlama, DO-178C açısından bir idari formalite değildir. Aksine, projenin nasıl çalışacağını, hangi kuralların uygulanacağını ve hangi kanıtların hangi sırayla biriktirileceğini belirleyen ana sözleşmedir. Bu yüzden iyi bir plan, sonraki tüm faaliyetlerin yanlış anlaşılmasını önleyen ortak bir referans noktasıdır.

Planların amacı

Planlar, "ne yapıyoruz?" sorusundan çok "nasıl ve hangi disiplinle yapıyoruz?" sorusuna cevap verir. Bu ayrım önemlidir; çünkü aynı proje içinde farklı ekipler aynı kelimeyi farklı anlamlarda kullanabilir. Planlar bu belirsizliği azaltır.

Plan seti birlikte okunduğunda şu soruların cevabı bulunabilmelidir:

- Hangi yaşam döngüsü modeli izlenecek, bir süreçten diğerine hangi koşulla geçilecek?
- Hangi yaşam döngüsü verisi (software life cycle data) üretilecek ve bu veri hangi standartlara göre yazılacak?
- Hangi gözden geçirmeler, analizler ve testler yapılacak; bağımsızlık (independence) nerede ve nasıl sağlanacak?
- Araçlar hangi rollerde kullanılacak, geliştirme ve test ortamı nasıl yönetilecek?
- Konfigürasyon öğeleri nasıl korunacak, problemler ve plandan sapmalar nasıl ele alınacak?

Bu soruların net olması, özellikle ekip büyüdükçe önem kazanır. Çünkü planlar sadece sertifikasyon için değil, ekip içi koordinasyon için de gereklidir.

Neden erken yazılmalı?

Planlar geç yazıldığında ekip çoğu kararı fiilen vermiş olur; belge ise sadece gecikmiş bir açıklamaya dönüşür. Bu durum sertifikasyon açısından zayıftır, çünkü kararların nasıl alındığını değil, sonradan nasıl anlatıldığını gösterir.

Erken yazılan planlar ise şu faydaları sağlar:

- ekip beklentilerini hizalar,
- rol ve sorumlulukları belirginleştirir,

- gereksinim ve test yaklaşımını baştan uyumlu hâle getirir,
- otorite ve müşteri denetimlerinde sürprizleri azaltır.

Ana planlar

DO-178C beş plan bekler. Her biri ayrı bir soruya cevap verir ve ayrı bir okur kitlesine seslenir:

Plan	Yanıtladığı soru	Ayrıntısı
Yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC)	Standarda uyum otoriteye nasıl gösterilecek?	12. Sertifikasyon İrtibatı
Yazılım geliştirme planı (Software Development Plan, SDP)	Yazılım hangi yaşam döngüsü, yöntem ve ortamla geliştirilecek?	Bu bölüm ve 6–8. bölümler
Yazılım doğrulama planı (Software Verification Plan, SVP)	Ürünün doğruluğu hangi yöntemlerle ve kim tarafından gösterilecek?	9. Yazılım Doğrulama
Yazılım konfigürasyon yönetimi planı (Software Configuration Management Plan, SCMP)	Veri nasıl tanımlanacak, korunacak ve kontrollü biçimde değiştirilecek?	10. Yazılım Konfigürasyon Yönetimi
Yazılım kalite güvencesi planı (Software Quality Assurance Plan, SQAP)	Süreçlerin planlara uyduğu nasıl ve kim tarafından güvence altına alınacak?	11. Yazılım Kalite Güvencesi

Yazılım sertifikasyon planı (PSAC)

PSAC, sertifikasyon otoritesiyle varılacak mutabakatın yazılı hâlidir ve beş plan içinde otoriteye sunulması beklenen plandır. Ayrıntıya girmez; otoritenin "bu projede neye dikkat etmeliyim?" sorusunu cevaplar. Tipik içeriği şöyledir:

- sistemin ve yazılımın kısa tanıtımı: yazılımın hangi işlevi yerine getirdiği, hangi donanımda çalıştığı, emniyet ve yazılım bölümlenmesi (software partitioning) yaklaşımı,
- sertifikasyon hususları: yazılım seviyesi (software level) ve bunun sistem emniyet değerlendirme sürecindeki dayanağı ([3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#)), uyumun hangi belgelere göre gösterileceği, hangi teknoloji eklerinin (supplement) uygulanacağı,
- izlenecek yaşam döngüsünün ve diğer planların özeti,
- üretilecek yaşam döngüsü verisi; hangisinin otoriteye sunulacağı, hangisinin talep edildiğinde gösterileceği,
- takvim ve otoritenin sürece katılacağı noktalar,
- ek hususlar (additional considerations): araç kalifikasyonu, önceden geliştirilmiş yazılım (previously developed software, PDS), ticari hazır yazılım (commercial off-the-shelf, COTS), devre dışı bırakılmış kod (deactivated code), sahada yüklenebilir yazılım (field-loadable)

software), kullanıcı tarafından değiştirilebilir yazılım (user-modifiable software), parametre verisi ögesi (parameter data item, PDI), alternatif yöntemler,

- tedarikçi kullanılıyorsa gözetim yaklaşımı.

En pahalı hata, ek hususları eksik bildirmektir. PSAC'ta anılmayan bir kod üretici (code generator) ya da yeniden kullanılan bir bileşen son denetimde ortaya çıkarsa, tartışma projenin en az esnek olduğu anda yapılır.

Yazılım geliştirme planı (SDP)

SDP, geliştirme ekibinin çalışma kılavuzudur ve üç şeyi tanımlar: yazılım yaşam döngüsünü (süreçler, aralarındaki ilişki ve geçiş kriterleri), kullanılacak standartları ve geliştirme ortamını (gereksinim ve tasarım yöntemleri, programlama dili, derleyici, bağlayıcı, yükleyici, diğer araçlar ve bunların çalıştığı donanım platformları). Gereksinim, tasarım ve kod kurallarının kendisi SDP'de değil standartlarda yer alır; SDP onlara atıf yapar. Küçük projelerde standartlar SDP ile aynı belgede paketlenabilir; o durumda da kurallar numaralı ve atıf yapılabilir kalmalıdır.

Yazılım doğrulama planı (SVP)

SVP, doğrulama hedeflerinin (objective) nasıl karşılanacağını anlatır: doğrulama organizasyonu ve bağımsızlığın hangi faaliyette kim tarafından sağlanacağı; gözden geçirme, analiz ve test yöntemleri; doğrulama ortamı (test donanımı, emülatör, simülatör, araçlar); doğrulama faaliyetlerine giriş için geçiş kriterleri; değişiklik sonrası yeniden doğrulama yaklaşımı ve derleyiciye ilişkin varsayımlar. Bölümleme kullanılıyorsa bütünlüğünün nasıl doğrulanacağı da burada yer alır.

Zayıf bir SVP "yazılım test edilecektir" düzeyinde kalır. Denetçi ise somut cevap arar: yapısal kapsam (structural coverage) hangi araçla ölçülecek, kapsanmayan kod nasıl çözümlenecek, hangi testler hedef donanımda koşulacak?

Yazılım konfigürasyon yönetimi planı (SCMP)

SCMP; konfigürasyon tanımlama kurallarını (adlandırma, sürümleme), temel çizgileri (baseline), problem raporlama (problem reporting) ve değişiklik kontrolü akışını, konfigürasyon durum muhasebesini (configuration status accounting), arşivleme, geri getirme ve sürüm teslimini, yazılım yükleme kontrolünü ve yaşam döngüsü ortamının kontrolünü tanımlar. Hangi verinin hangi kontrol kategorisinde (control category) yönetileceği ve tedarikçi verisinin nasıl kontrol edileceği de buradadır. Sık sorulan bir soru, problem raporunun hangi andan itibaren zorunlu olduğudur: gözden geçirme bulgusu ile problem raporu arasındaki sınırı SCMP çizer.

Yazılım kalite güvencesi planı (SQAP)

SQAP; kalite güvencesinin yetkisini, sorumluluğunu ve bağımsızlığını, yapacağı faaliyetleri (süreç denetimleri, geçiş kriterlerinin izlenmesi, yazılım uygunluk gözden geçirmesi), bu faaliyetlerin zamanını, uygunsuzlukların nasıl kaydedilip kapatılacağını ve tutulacak kayıtları tanımlar. Yazılım uygunluk gözden geçirmesi (software conformity review), sertifikasyona sunulan sürüm için süreçlerin tamamlandığını, verinin eksiksiz olduğunu ve çalıştırılabilir nesne kodunun kontrol altında ve yeniden üretilebilir olduğunu gösteren kapanış denetimidir. Tipik zayıflık ölçülemeyen taahhüttür: "kalite güvencesi tüm faaliyetleri izler" cümlesi ne planlanabilir ne de denetlenebilir; "her temel çizgi öncesinde geçiş kriterleri kontrol listesiyle denetlenir" cümlesi ikisini de sağlar.

Planlar arasındaki ilişki

Bu planlar birbirinden bağımsız belgeler gibi görünse de gerçekte aynı sistemin farklı bakış açılarıdır. SVP'nin vaat ettiği bağımsızlık organizasyon yapısıyla, SDP'deki geçiş kriterleri SQAP'taki denetim noktalarıyla, SVP'deki test hatası akışı SCMP'deki problem raporlamayla örtüşmelidir. Bir plandaki değişiklik bu yüzden diğer planları da etkileyebilir.

Beş plan beş ayrı belge olmak zorunda da değildir. Küçük projelerde bazıları tek belgede birleştirilir, büyük kuruluşlarda ise ortak kurum süreçlerine atıf yapan ince proje planları yazılır. Belirleyici olan paketleme değil, beklenen içeriğin bulunabilir ve tutarlı olmasıdır. Tek çekince şudur: bazı otoriteler PSAC'ı ve yazılım başarı özetini ayrı belgeler olarak görmek isteyebilir.

Beş plan A'dan D'ye bütün yazılım seviyelerinde beklenir; seviyeyle değişen, yedi planlama hedefinden kaçının arandığıdır. Seviye A, B ve C'de yedisi de geçerlidir. Seviye D'de yalnızca ikisi aranır: yaşam döngüsü süreçlerinin faaliyetlerinin tanımlanması ve ek hususların ele alınması (Ek A'nın planlama tablosunda 1. ve 4. hedef). Yaşam döngüsünün ve geçiş kriterlerinin belirlenmesi, yaşam döngüsü ortamının seçilip tanımlanması, geliştirme standartlarının tanımlanması, planların standarda uyumu ve planların eşgüdümü Seviye D'de başvuru sahibinin takdirine bırakılır; üç geliştirme standardı da bu seviyede beklenen veri arasında değildir. Kalite güvencesi tarafı bununla uyumludur: Seviye D'de süreçlerin onaylı planlara uyduğuna ve yazılım uygunluk gözden geçirmesinin yapıldığına dair güvence istenir; planların ve standartların gözden geçirilmesi, standartlara uyum ve geçiş kriterlerinin sağlanması için ayrıca güvence istenmez. Bu, düşük seviyede standart ya da geçiş kriteri tanımlamanın yararsız olduğu anlamına gelmez; yalnızca standardın o seviyede bunları şart koşmadığını gösterir.

Yaşam döngüsü ve geçiş kriterleri

DO-178C belirli bir yaşam döngüsü modeli (life cycle model) dayatmaz. Şelale, artımlı ya da yinelemeli bir model seçmek projenin kararıdır; SDP modeli tarif eder, PSAC özetler. Standart süreçleri ve birbirlerini nasıl beslediklerini tanımlar; hangi sırayla ve hangi geri beslemeyle

yürüyeceklerini planlara bırakır. Bütünleyici süreçler (integral processes) — doğrulama, konfigürasyon yönetimi, kalite güvencesi ve sertifikasyon irtibatı — bu sıranın bir adımı değildir; geliştirmeyle eşzamanlı yürür.

Sırayı belirleyen araç geçiş kriterleridir (transition criteria): bir sürece girmek için sağlanması gereken koşullar. İyi bir kriter, bir kaydın varlığıyla denetlenebilir. "Gereksinimler yeterince olgunlaşınca tasarıma başlanır" cümlesi denetlenemez; "tasarımı yapılacak işlevin yüksek seviyeli gereksinimleri gözden geçirilmiş, bulguları kapatılmış ve temel çizgiye alınmıştır" cümlesi denetlenebilir.

Kriter yazarken iki noktaya dikkat edilir:

- **Kriter, ekibin gerçekte nasıl çalışacağını yansıtmalıdır.** Bir sürecin başlaması için girdilerinin tümünün tamamlanmış olması gerekmez; kriter işlev ya da artım bazında tanımlanabilir. Kâğıt üzerinde katı bir şelale tarif edip fiilen artımlı çalışan ekip, her denetimde planından sapmış görünür.
- **Kriterler yalnızca geliştirme süreçleri için yazılmaz.** Kredi alınacak testlerin hangi koşulda başlayabileceği (kod temel çizgiye alınmış mı, test prosedürleri gözden geçirilmiş mi, test ortamı kayıtlı mı) en az tasarıma geçiş kadar önemlidir.

Kriterlerin sağlandığını izlemek kalite güvencesinin işidir; bu yüzden her kriterin arkasında gösterilebilir bir kayıt bulunmalıdır. Süreçler arası her geçiş için örnek kriterler [Ek A: Örnek Geçiş Kriterleri](#) sayfasındadır.

Üç geliştirme standardı

Planların yanında DO-178C, Seviye A, B ve C'de üç geliştirme standardının tanımlanmasını bekler: gereksinim standardı, tasarım standardı ve kodlama standardı. Bunlar ekibin iş ürünlerini hangi kurallara göre yazacağını belirler ve gözden geçirmelerin nesnel ölçütü hâline gelir.

Bu üç standart, planların "nasıl çalışacağız?" sorusuna verdiği cevabın teknik karşılığıdır. Planlar süreci tarif ederken standartlar, üretilen her bir iş ürününün kendisine bakar: gereksinim nasıl yazılır, tasarım nasıl ifade edilir, kod hangi kurallara uyar. Gözden geçirme yapan kişi elinde bu standartlar olmadan "bence böyle daha iyi olurdu" düzeyinde kalır; standart varsa "bu madde 4.2'ye aykırı" diyebilir. Fark, öznel görüş ile nesnel bulgu arasındaki farktır.

Gereksinim standardı

Gereksinim standardı (requirements standard), yüksek seviyeli gereksinimlerin nasıl yazılacağını tanımlar. Tipik içeriği şunlardır:

- kullanılacak gereksinim yazım biçimi (örn. "Yazılım, ... olduğunda ... yapmalıdır" kalıbı),

- her gereksinimin taşıması gereken nitelikler: tek bir davranışı anlatması, doğrulanabilir olması, belirsiz ifadelerden arınmış olması,
- yasaklı kelimeler listesi ("uygun şekilde", "hızlıca", "vb." gibi test edilemeyen ifadeler),
- gereksinim kimliklendirme ve izlenebilirlik (traceability) kuralları,
- türetilmiş gereksinimlerin (derived requirements) nasıl işaretleneceği ve sistem emniyet değerlendirme süreci dahil sistem süreçlerine nasıl iletileceği,
- tolerans, birim ve zamanlama değerlerinin nasıl belirtileceği.

Sık yapılan hata, standardın "iyi gereksinim yazma" üzerine genel öğütlerle dolu olup projeye özgü karar içermemesidir. "Gereksinimler açık olmalıdır" cümlesi bir gözden geçirmede kullanılamaz; "her zamanlama gereksinimi milisaniye cinsinden alt ve üst sınır içermelidir" cümlesi kullanılabilir. Standardın gereksinim yazımında nasıl kullanıldığı [6. Yazılım Gereksinimleri](#) bölümünde işleniyor.

Tasarım standardı

Tasarım standardı (design standard), yazılım mimarisinin ve düşük seviyeli gereksinimlerin ifade biçimini belirler. Tipik olarak şunları kapsar:

- mimari gösterim yöntemi (blok diyagramları, veri/kontrol akışı gösterimi),
- modülleşme ve arayüz tanımlama kuralları,
- düşük seviyeli gereksinimlerin (low-level requirement) ayrıntı düzeyi: koddan bire bir kopya olmayacak kadar soyut, kodlayıcıya yorum payı bırakmayacak kadar somut,
- kesme (interrupt) kullanımı, görev önceliklendirme, çizelgeleme ve zamanlama tasarımına dair sınırlar,
- özyineleme (recursion), dinamik bellek gibi riskli yapıların yasaklanması ya da hangi koşullarda serbest olduğu.

Burada en sık görülen sorun, düşük seviyeli gereksinim ayrıntı düzeyinin tanımlanmamasıdır. Ayrıntı düzeyi ekipten ekibe geçince, aynı projede kimi modül sözde kod (pseudocode) düzeyinde, kimi modül tek cümlelik özet düzeyinde kalır; bu da doğrulama maliyetini öngörülemez hâle getirir. Ayrıntı düzeyi tartışması [7. Yazılım Tasarımı](#) bölümünde sürüyor.

Kodlama standardı

Kodlama standardı (coding standard), kaynak kodun uyacağı kuralları tanımlar. C ile çalışan aviyonik projelerde genellikle MISRA C gibi yerleşik bir kural kümesi temel alınır ve projeye özgü eklemelerle genişletilir. Kural kümesinin hangi sürümünün esas alındığı standartta açıkça yazılır ve statik analiz aracının o sürümü desteklediği planlama sırasında kontrol edilir. Tipik içerik şöyledir:

- dil alt kümesi: hangi dil özelliklerinin yasak olduğu (örn. `goto`, işaretçi aritmetiğinin sınırlandırılması, örtük tip dönüşümleri),

- adlandırma, dosya düzeni ve yorum kuralları,
- karmaşıklık sınırları (örn. fonksiyon başına çevrimsel karmaşıklık (cyclomatic complexity) üst sınırı),
- derleyici uyarılarının nasıl ele alınacağı,
- sapma (deviation) mekanizması: bir kural ihlalinin hangi gerekçeyle, kimin onayıyla kabul edilebileceği.

Küçük bir örnek, kuralın ne kadar somut olabileceğini gösterir:

```
/* Kural: her switch ifadesinde default dalı bulunmalı ve
   beklenmeyen değer bir hata işleyicisine raporlanmalıdır. */
switch (mod) {
    case MOD_BEKLEME:
        bekleme_isle();
        break;
    case MOD_CALISIYOR:
        calisma_isle();
        break;
    default:
        hata_raporla(HATA_GECERSIZ_MOD);
        break;
}
```

Böyle bir kuralın doğrulamaya da yansıması vardır: savunmacı `default` dalına gereksinim tabanlı testlerle ulaşamayabilir ve yapısal kapsam analizinde gerekçe ister. Bu durumun nasıl ele alınacağı [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#) bölümünde, standardın kodlama sırasında uygulanması ise [8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#) bölümünde anlatılıyor.

Üç standardın ortak tuzağı aynıdır: standart yazılır ama uygulanmaz. Standardın yaşaması için gözden geçirme kontrol listelerinin standarda madde madde atıf yapması ve mümkün olan kuralların statik analiz araçlarıyla otomatik denetlenmesi gerekir.

Standart	Neyi düzenler?	Tipik nesnel ölçüt örneği
Gereksinim standardı	Gereksinimlerin yazımı	Zamanlama değerleri alt/üst sınırla verilir
Tasarım standardı	Mimari ve düşük seviyeli gereksinimler	Dinamik bellek kullanımı yasaktır
Kodlama standardı	Kaynak kod	Fonksiyon başına karmaşıklık sınırı aşılamaz

Yaşam döngüsü ortamının planlanması

Yazılım yaşam döngüsü ortamı (software life cycle environment), yazılımı geliştirmek, doğrulamak, kontrol altında tutmak ve üretmek için kullanılan yöntem, araç ve donanımların bütünüdür. Ortam seçimi bir satın alma kararı gibi görünse de sonraki bütün doğrulama işinin

maliyetini belirlediği için planlama kararıdır.

- **Dil, derleyici ve derleme seçenekleri.** Uçağa yüklenen ve doğrulanması gereken ürün çalıştırılabilir nesne kodu olduğundan, derleyicinin sürümü kadar hangi seçeneklerle ve hangi eniyileme (optimization) düzeyinde kullanılacağı da planlanır. Seviye A'da kaynak koda doğrudan izlenemeyen nesne kodu ek doğrulama gerektirdiği için bu seçim doğrudan iş yüküne dönüşür.
- **Geliştirme araçları.** Gereksinim yönetimi, modelleme ve kod üretimi araçları hem standartlarla (araç hangi gösterimi destekliyor?) hem de araç kalifikasyonu kararıyla birlikte düşünülür.
- **Test ortamı.** Hangi testin hedef donanımda, hangisinin emülatör ya da ana bilgisayar simülasyonunda koşulacağı SVP'de belirlenir. Hedef dışı bir ortamdan kredi alınacaksa, o ortamla hedef arasındaki farkların hata yakalama ve işlevi doğrulama yeteneğine etkisi değerlendirilir; o ortamda yakalanamayacak hataların hangi doğrulama faaliyetiyle yakalanacağı SVP'ye yazılır. Donanım arayüzleri ve zamanlama gibi yalnızca hedefte ortaya çıkabilecek hatalar için seçilmiş testler entegre hedef ortamında koşulacak biçimde planlanır.

Seçilen ortam, sürümleri ve ayarlarıyla birlikte yazılım yaşam döngüsü ortam konfigürasyon indeksine (Software Life Cycle Environment Configuration Index, SECI) kaydedilir. Proje ortasında derleyici sürümünü ya da derleme seçeneklerini değiştirmek, önceki test ve kapsam sonuçlarının geçerliliğini sorgulatar; bu yüzden böyle bir değişikliğin nasıl değerlendirileceği de baştan planlanır. Dil ve derleyici seçimindeki ölçütler 8. bölümde, ortamın kayıt altına alınması [10. Yazılım Konfigürasyon Yönetimi](#) bölümünde ayrıntılı işleniyor.

Araç kalifikasyonu planlaması

Araç kalifikasyonu (tool qualification), bir yazılım aracının çıktısına, o çıktı ayrıca doğrulanmadan güvenilebileceğini göstermek için yapılan çalışmadır. Hangi araçların kalifikasyon gerektirdiği sorusu bir doğrulama sorusu gibi görünse de aslında bir planlama sorusudur: karar planlama aşamasında verilmez ve PSAC'a yazılmazsa, proje sonunda ya beklenmedik bir kalifikasyon maliyeti ya da otoriteyle zor bir tartışma ortaya çıkar.

Planlama aşamasında her araç için sorulacak iki temel soru vardır:

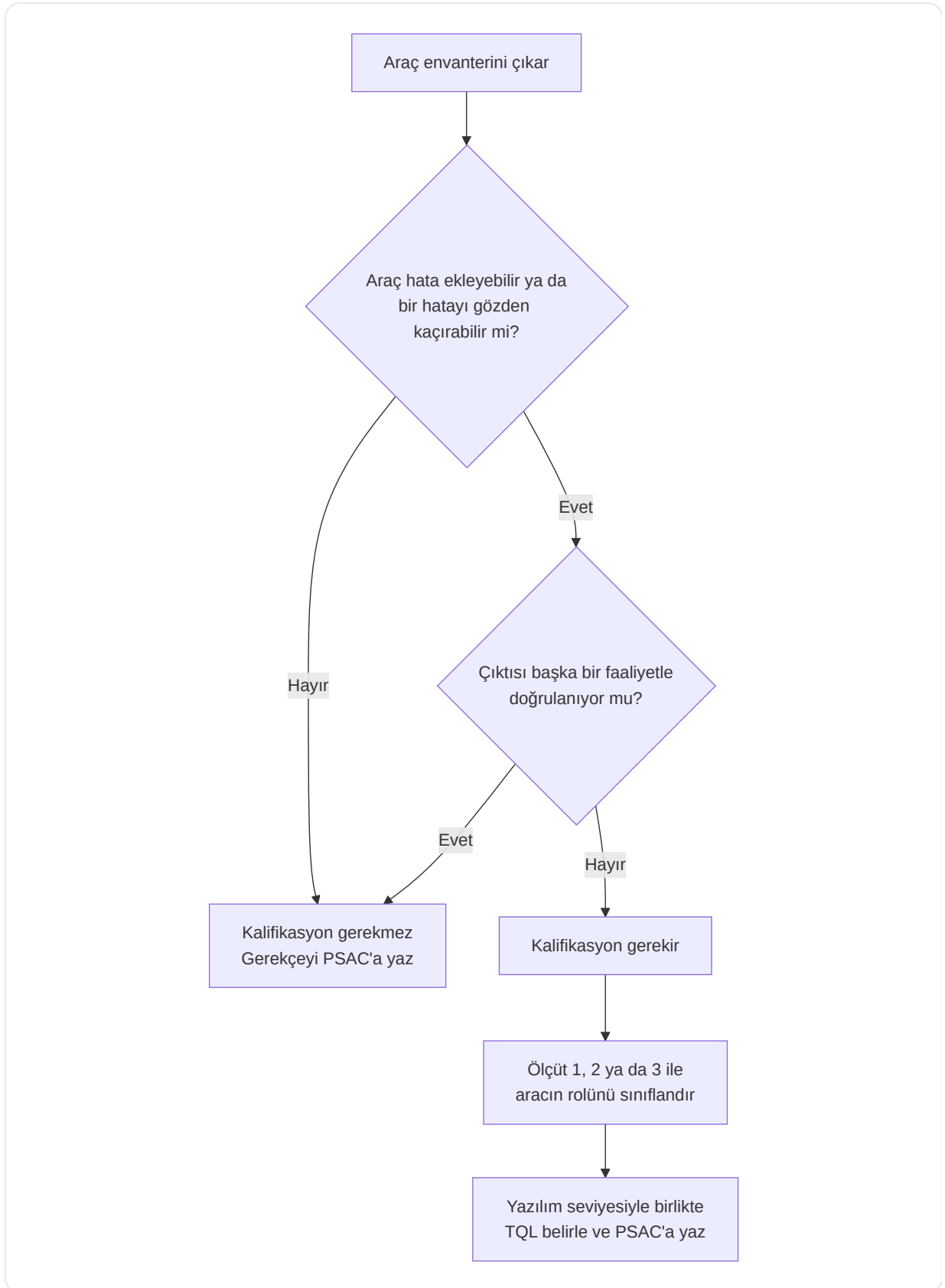
1. **Araç yazılıma hata ekleyebilir ya da var olan bir hatayı gözden kaçırabilir mi?** Bir kod üretici hatalı kod üretirse bu kod doğrudan çalıştırılabilir nesne koduna gider; bir test aracı başarısız bir testi başarılı gösterebilir.
2. **Aracın çıktısı başka bir faaliyetle zaten doğrulanıyor mu?** Üretilen kod, elle yazılmış kodla aynı gözden geçirme ve test sürecinden geçiyorsa ya da test aracının sonuçları bir mühendis tarafından ayrıca kontrol ediliyorsa aracın hatası yakalanır; kalifikasyon gerekmez.

Kalifikasyon yalnızca ilk sorunun cevabı "evet", ikincisinin cevabı "hayır" olduğunda gerekir; bu koşul geliştirme araçları kadar doğrulama araçları için de geçerlidir. Derleyici, kalifikasyon gerekmeyen durumun bilinen örneğidir: ürettiği nesne kodu testlerle doğrulandığı için genellikle kalifiye edilmez.

Kalifikasyon gereken araçlar, projedeki rollerine göre üç ölçütten birine girer:

- **Ölçüt 1:** çıktısı uçuş yazılımının parçası olan, dolayısıyla hata ekleyebilen araç (örn. çıktısı ayrıca gözden geçirilmeyen kod üretici) — en ağır kalifikasyon yükü.
- **Ölçüt 2:** doğrulamayı otomatikleştiren ve çıktısı başka geliştirme ya da doğrulama faaliyetlerini kaldırmanın ya da azaltmanın gerekçesi olarak kullanılan araç (örn. sonucuna dayanılarak bazı testlerden vazgeçilen bir statik analiz aracı).
- **Ölçüt 3:** kendi kullanım amacı içinde bir hatayı gözden kaçırabilen, ama başka bir faaliyetin azaltılmasına gerekçe yapılmayan araç (örn. yapısal kapsam analizi aracı) — en hafif kalifikasyon yükü.

Ölçüt, yazılım seviyesiyle birleşerek araç kalifikasyon seviyesini (Tool Qualification Level, TQL) verir: Ölçüt 1'de Seviye D'den A'ya doğru TQL-4'ten TQL-1'e sıkışır, Ölçüt 2'de Seviye A ve B için TQL-4, C ve D için TQL-5'tir, Ölçüt 3'te her seviyede TQL-5'tir. Ölçütler 1'den 3'e doğru sırayla değerlendirilir. Tablonun tamamı ve kalifikasyon süreci [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#) bölümünde ele alınıyor; planlama açısından önemli olan, kararın kendisinin ve gerekçesinin erken verilmesidir.



PSAC'ta her araç için en azından şunlar yer almalıdır: aracın adı ve sürümü, projedeki rolü, kalifikasyon gerekip gerekmediği, gerekiyorsa öngörülen TQL ve kalifikasyon verisinin nasıl üretileceği. Kalifikasyon gerekmediğine karar verilen araçlar için de gerekçenin yazılması iyi bir

alışkanlıktır; çünkü otoritenin ilk sorularından biri genellikle "bu araca neden güveniyorsunuz?" olur.

Deneyimden gelen bir uyarı: en çok gözden kaçan araçlar, gösterişli geliştirme araçları değil, arka plandaki küçük betiklerdir. Test sonuçlarını toplayan bir betik, izlenebilirlik matrisini üreten bir makro ya da yapılandırma dosyalarını dönüştüren bir araç da aynı sorgulamadan geçmelidir. Planlama aşamasında eksiksiz bir araç envanteri çıkarmak, bu sürprizleri önlemenin en ucuz yoludur.

Planların gözden geçirilmesi, onayı ve değişmesi

Planlar ve standartlar da birer yaşam döngüsü verisidir: gözden geçirilir, konfigürasyon kontrolüne alınır ve kontrollü biçimde değişir.

Gözden geçirme iki soruya bakar. Birincisi uyumdur: yazılım seviyesinin gerektirdiği her hedef için planlarda bir faaliyet ve o faaliyetin üreteceği kanıt tanımlı mı? Bunu göstermenin yaygın yolu, hedefleri plan bölümleriyle eşleyen bir uyum matrisidir. İkincisi tutarlılıktır: bir planın "SVP'de tanımlanmıştır" dediği konu gerçekten SVP'de var mı, iki plan aynı akışı farklı mı anlatıyor? Seviye A, B ve C'de kalite güvencesinden, planların ve standartların geliştirildiğine ve bu iki açıdan gözden geçirildiğine dair güvence beklenir.

Onay tarafında planların hepsi aynı yolu izlemez. PSAC otoriteye erken sunulur ve üzerinde mutabakat aranır; yazılım konfigürasyon indeksi (Software Configuration Index, SCI) ve yazılım başarı özeti (Software Accomplishment Summary, SAS) ile birlikte otoriteye asgari olarak sunulan veriyi oluşturur. Diğer dört plan ve üç standart otoritenin erişimine açık tutulur ve uygulamada ilk katılım aşaması (Stage of Involvement, SOI) denetiminde incelenir. Bu denetimin giriş koşulları ve otoritenin planlarda baktığı noktalar [SW SOI-1](#) kontrol listesindedir.

Değişiklik kaçınılmazdır; önemli olan kontrollü olmasıdır. Plan değişikliği değişiklik kontrolünden geçer, etkilenen diğer planlarla birlikte yeniden gözden geçirilir; PSAC'ın özünü etkiliyorsa (seviye, kapsam, araç ya da yöntem değişikliği) otoriteye bildirilir. Uygulama plandan ayrıldığında iki dürüst yol vardır: önce planı değiştirmek ya da sapmayı kayda geçirmek. Plan ve standartlardan sapmalar problem raporlama ve kalite güvencesi kayıtlarıyla izlenir, proje sonunda da SAS'ta özetlenir. Denetimde en sık duyulan cümle "planınız şunu söylüyor, kaydını görebilir miyim?" olduğundan, plana yalnızca gerçekten yapılacak ve kanıtlanabilecek şeyler yazılmalıdır.

Diğer planlar

Beş plan, sertifikasyon açısından beklenen çekirdeği oluşturur; ama gerçek projelerde plan seti bununla sınırlı kalmaz. Kuruluşlar, kendi iç işleyişlerini yönetmek için ek planlar yazar. Bunlar DO-178C'nin tanımladığı yaşam döngüsü verisi arasında sayılmaz; yine de beş planla çelişmemeli ve gerektiğinde onlardan atıfla erişilebilir olmalıdır.

Sık karşılaşılan ek planlar şunlardır:

- **Proje yönetim planı (project management plan):** takvim, bütçe, kaynak ataması ve risk yönetimi gibi konuları kapsar. DO-178C bu konularla doğrudan ilgilenmez; ancak gerçekçi olmayan bir takvim, en iyi yazılmış planları bile işlevsiz bırakır. Bu yüzden proje yönetim planındaki kilometre taşlarının, planlardaki geçiş kriterleriyle uyumlu olması gerekir.
- **Gereksinim yönetim planı (requirements management plan):** gereksinimlerin hangi araçta tutulacağını, öznitelik şemasını, değişiklik akışını ve izlenebilirlik bağlarının nasıl kurulacağını tanımlar. Gereksinim standardı "nasıl yazılır" sorusuna, gereksinim yönetim planı "nasıl yönetilir" sorusuna cevap verir.
- **Test planı (test plan):** SVP'nin çizdiği çerçeveyi somutlaştırır; test ortamlarını, test donanımını, test kampanyalarının sırasını ve sorumluluklarını ayrıntılandırır. Küçük projelerde bu içerik doğrudan SVP içinde de verilebilir.
- **Entegrasyon planı, laboratuvar/ortam planı** gibi belgeler: hedef donanım üzerinde çalışma, test düzeneklerinin bakımı ve ortam konfigürasyonunun korunması gibi pratik konuları düzenler.

Bu belgelerle beş plan arasındaki ilişkiyi kurarken iki hataya dikkat edilmelidir:

1. **Aynı bilgiyi iki yerde tanımlamak.** Örneğin değişiklik onay akışı hem SCMP'de hem gereksinim yönetim planında farklı anlatılırsa, ekip hangisine uyacağını bilemez. Kural basittir: her konu tek bir belgede tanımlanır, diğer belgeler ona atıf yapar.
2. **Ek planları görünmez tutmak.** Otorite ya da müşteri denetimi sırasında ekip fiilen gereksinim yönetim planındaki akışa göre çalışıyorsa, bu belgenin varlığı ve beş planla ilişkisi PSAC'ta ya da ilgili planda anılmalıdır. Görünmeyen bir plan, denetimde "tanımsız süreç" izlenimi verir.

Özetle ek planlar, beş planın alternatifi değil tamamlayıcısıdır. İyi kurgulanmış bir plan setinde okuyucu, hangi sorunun cevabının hangi belgede olduğunu tahmin edebilir; kötü kurgulanmış bir sette ise aynı cevabın üç farklı sürümü üç farklı belgede yaşar.

Bu bölümden akılda kalması gerekenler

- Planlama, sonradan belge yazma işi değildir; projenin çalışma biçimidir ve erken yapıldığında doğrulama ve sertifikasyon riskini azaltır.
- Beş plan (PSAC, SDP, SVP, SCMP, SQAP) ayrı sorulara cevap verir ama birlikte okunur. PSAC otoriteye sunulur; diğer planlar ve standartlar erişime açık tutulur ve SOI-1 denetiminde incelenir. Planlar her seviyede beklenir; Seviye D'de yedi planlama hedefinden yalnızca ikisi aranır.
- DO-178C yaşam döngüsü modeli dayatmaz; süreçler arası sırayı planlardaki, kayıtlı denetlenen geçiş kriterleri belirler.

- Üç geliştirme standardı, gözden geçirmelerin nesnel ölçütüdür; projeye özgü ve denetlenebilir kurallar içermelidir. Kurallar standartlarda durur, SDP onlara atıf yapar.
- Dil, derleyici, derleme seçenekleri ve test ortamı planlama kararıdır; ortam kayıt altına alınır ve proje ortasındaki değişikliği yeniden doğrulama gerektirebilir.
- Araç kalifikasyonu, araç hata ekleyebiliyor ya da gözden kaçırabiliyor ve çıktısı ayrıca doğrulanmıyorsa gerekir; karar gerekçesiyle PSAC'a yazılır ve küçük betikler de araç envanterine dahildir.
- Planlar gözden geçirilir ve kontrollü değişir; plandan sapma kayda geçer. Ek planlar beş planı tamamlar; her konu tek belgede tanımlanır, diğerleri ona atıf yapar.

6. Yazılım Gereksinimleri

Yazılım gereksinimleri, sistemin yazılımdan beklediğini doğrulanabilir davranış tanımlarına çevirir. Bu bölüm, yüksek seviyeli gereksinimlerin (high-level requirements) sistem gereksinimlerinden nasıl geliştirildiğini, neleri kapsadığını, nasıl yazılıp gözden geçirildiğini ve değişikliklerinin nasıl yönetildiğini anlatır.

Düşük seviyeli gereksinimler (low-level requirements) tasarım sürecinin çıktısıdır; burada yalnızca iki katman arasındaki sınır çizilir, ayrıntısı [7. Yazılım Tasarımı](#) bölümündedir. Güçlü gereksinimler tasarım, kod ve test için sağlam bir başlangıç noktası sağlar; zayıf gereksinimin bedeli ise bu üçünde birden ödenir.

Gereksinim neden bu kadar önemli?

Gereksinim, sistem ihtiyacını yazılım diline çeviren köprüdür. Köprü sağlam değilse, tasarım yanlış yöne kayar, kod istenmeyen davranış üretir ve test ekibi neyi doğruladığını tam olarak bilemez.

DO-178C'de bu bağ gevşek bir benzetme değildir: test, gereksinim tabanlıdır (requirements-based testing). Test durumları (test case) koddan değil gereksinimden üretilir ve yazılım, gereksinimin söylediğine göre "doğru" ya da "yanlış" sayılır. Gereksinimde yazmayan davranış test edilmez; yanlış yazılmış davranış ise kusursuz biçimde kodlanıp "geçti" diye doğrulanır. Gereksinim hatası, sonraki bütün adımların sadakatle taşıdığı bir hatadır.

Bu yüzden gereksinim yazımı yalnızca doğru kelimeleri seçmek değildir; aynı zamanda doğru düzeyi seçmektir. Çok üst düzey bir ifade test edilemez kalır, çok alt düzey bir ifade ise gerekli mimari ayrımı bozar.

Yüksek ve düşük seviyeli gereksinimler

DO-178C yazılım gereksinimlerini iki katmanda ele alır. Katmanları ayıran, cümlelerin uzunluğu ya da madde sayısı değil; hangi sürecin çıktısı oldukları ve kime hitap ettikleridir.

- **Yüksek seviyeli gereksinimler**, yazılım gereksinim sürecinin çıktısıdır. Yazılıma tahsis edilen sistem gereksinimlerinin, emniyetle ilgili gereksinimlerin ve sistem mimarisinin analizinden geliştirilir. Yazılıma dışarıdan bakar: girdileri, çıktıları, gözlenen davranışı ve kısıtları söyler; yazılımın iç yapısına girmez.
- **Düşük seviyeli gereksinimler**, tasarım sürecinin çıktısıdır. Yüksek seviyeli gereksinimlerden ve tasarım kararlarından geliştirilir; ölçütü, kaynak kodun başka hiçbir bilgiye gerek kalmadan doğrudan yazılabileceği ayrıntıda olmasıdır.

İki katman her seviyede aynı ağırlıkta aranmaz. Yüksek seviyeli gereksinimlerin geliştirilmesi Seviye A'dan Seviye D'ye bütün seviyelerde hedeftir; düşük seviyeli gereksinimlerin geliştirilmesi ise Seviye D'de hedef olarak aranmaz.

Aynı işlevin üç katmandaki görünüşü farkı somutlaştırır:

Katman	Örnek	Ne söylüyor?
Sistem gereksinimi	Sistem, hava hızı verisinin kaybını en geç 1 saniye içinde tespit etmeli ve mürettebata bildirmelidir.	Sistemin işlevi; işin ne kadarının yazılıma düştüğü belli değil
Yüksek seviyeli gereksinim	Yazılım, hava hızı verisi kesintisiz 500 ms (± 20 ms) boyunca geçersiz kaldığında, bu sürenin dolmasından sonra en geç 50 ms içinde hava hızı arza bayrağını etkinleştirmelidir.	Yazılımın sınırında gözlenen davranış: girdi, koşul, süre, tepki
Düşük seviyeli gereksinim	Hava hızı izleme işlevi 20 ms'lik görevde her çağrıldığında, girdi geçersizse geçersizlik sayacını bir artırır, geçerliyse sıfırlar; sayaç 25'e ulaştığında arza bayrağını kurar.	Kodlayıcının doğrudan gerçekleştireceği ayrıntı

Sistem gereksinimindeki 1 saniyelik bütçenin toleransla birlikte en çok 570 ms'si yazılıma ayrılmış, kalanı sensör, veri yolu ve gösterge gecikmelerine bırakılmıştır. Yüksek seviyeli gereksinim sayaçtan, görevden ya da işlev adından söz etmez; bunlar tasarımın kararıdır ve yarın değişebilir. Düşük seviyeli gereksinim ise 25 çağrının 500 ms ettiğini bilen bir tasarımın ürünüdür.

Hangi katmanda olduğunuzdan emin değilseniz şu soru işe yarar: *bu cümleyi doğrulamak için yazılımın içini bilmem gerekiyor mu?* Yalnızca girdileri sürüp çıktıları gözleyerek doğrulanabiliyorsa yüksek seviyelidir; bir değişkenin, sayacın ya da görevin adını anmadan yazılamıyorsa tasarıma inilmiştir.

Gereksinimler her zaman düz metin olmak zorunda da değildir; bir model olarak ifade edildiklerinde katman ayrımı aynen geçerlidir, ama ek beklentiler doğar ([14. DO-331 ve Model Tabanlı Geliştirme ve Doğrulama](#)).

Türetilmiş gereksinimler

Türetilmiş gereksinim (derived requirement), yazılım geliştirme süreçlerinin kendi ürettiği ve şu iki koşuldaki en az birini taşıyan gereksinimdir:

- bir üst seviye gereksinime doğrudan izlenemez, ya da
- izlenebilse bile üst seviyede belirtilenin ötesinde bir davranış tanımlar.

Terimin Türkçesi yanıltıcı olabilir: gündelik dilde "yüksek seviyeli gereksinimler sistem gereksinimlerinden türetilir" denir, ama teknik anlamda türetilmiş gereksinim tam tersine, üstünde karşılığı *olmayan* gereksinimdir. Emniyet değerlendirmesinden yazılıma tahsis edilen bir gereksinim de türetilmiş değildir; onun kaynağı bellidir.

İki koşula birer örnek:

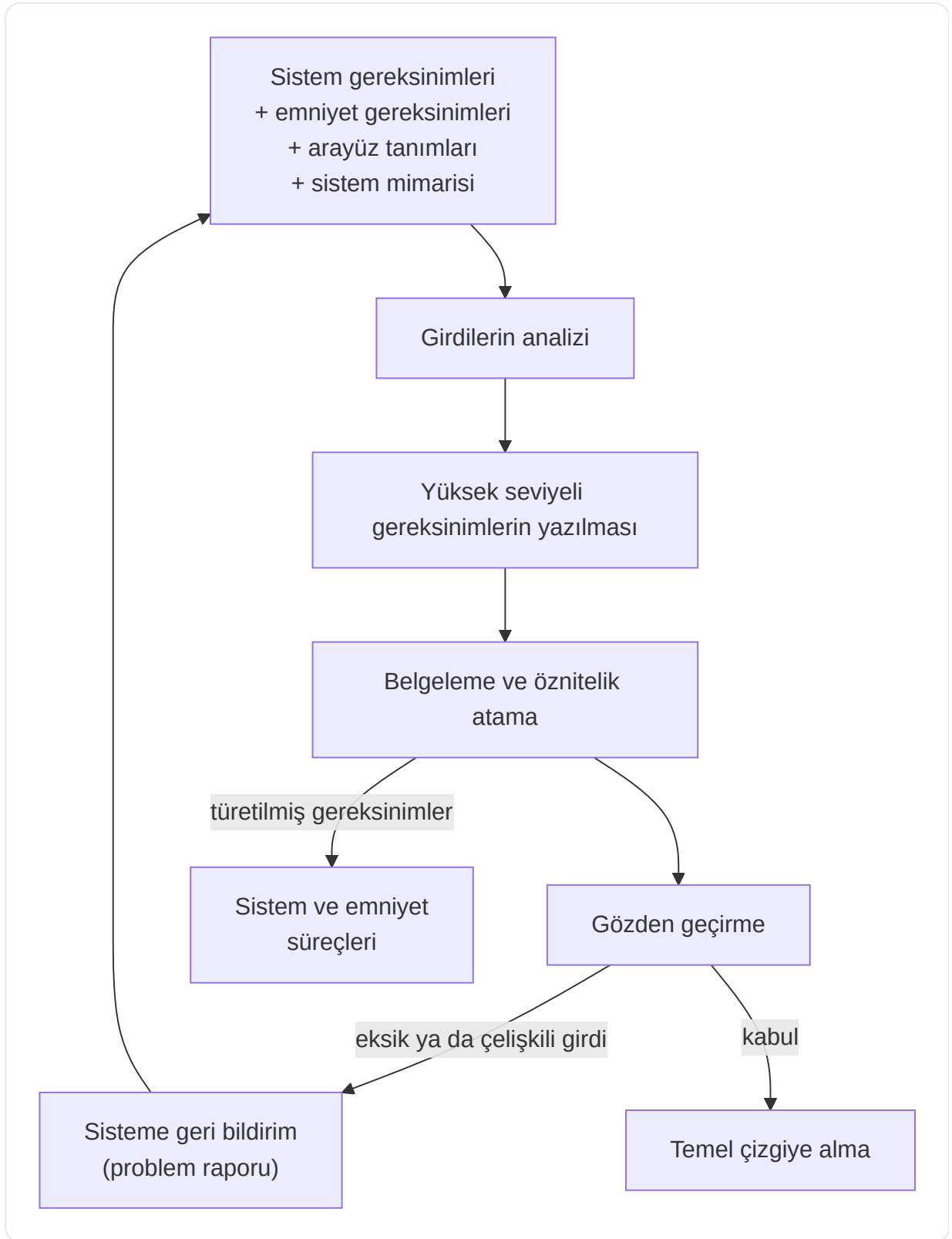
- **İzlenemeyen:** "Yazılım, donanımın bekçi köpeği zamanlayıcısını (watchdog timer) her 10 ms'lik çerçevede bir kez yenilemelidir." Hiçbir sistem gereksinimi bundan söz etmez; ihtiyaç, seçilen işlemci kartından doğmuştur.
- **Ötesinde davranış tanımlayan:** Sistem gereksinimi yalnızca "geçersiz hava hızı verisi kullanılmamalıdır" der; yazılım gereksinimi buna "ardışık iki örnek arasındaki fark 30 knot'tan büyükse veri geçersiz sayılmalıdır" kuralını ekler. İz vardır, ama bu makullük denetimi (reasonableness check) sistemin bilmediği yeni bir davranıştır.

Türetilmiş gereksinim hem yüksek hem düşük seviyede doğabilir; birincisi gereksinim sürecinde, ikincisi tasarım sırasında ortaya çıkar. Her birinin **gerekçesi** yazılır ve gereksinim, sistem süreçlerine — sistem emniyet değerlendirme süreci (system safety assessment process) dahil — iletilir. Nedeni basittir: emniyet değerlendirmesi, yazılımın ne yaptığına dair varsayımlar üzerine kuruludur ve sistemin haberi olmayan bir davranış bu varsayımları bozabilir. Yukarıdaki makullük denetimi, hava hızının gerçekten hızla değiştiği bir anda geçerli veriyi reddedebilir; bunun kabul edilebilir olup olmadığına yazılım ekibi değil, sistem ve emniyet ekibi karar verir. Geri bildirimin emniyet tarafındaki karşılığı [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#) bölümünde anlatılır.

Gözden geçirmede iki uç da soru doğurmalıdır: hiç türetilmiş gereksinimi olmayan bir yazılımda büyük olasılıkla türetilmiş gereksinimler işaretlenmemiştir; gereksinimlerinin çoğu türetilmiş olan bir yazılım ise çoğu kez sistem gereksinimlerinin eksik kaldığına işaret eder.

Gereksinim geliştirme süreci

Gereksinim geliştirme, tek oturuşta biten bir yazım işi değil; girdilerin toplanması, analiz, yazım, belgeleme ve geri bildirimden oluşan yinelemeli bir süreçtir. DO-178C terminolojisiyle bu, yazılım gereksinim sürecinin (software requirements process) çıktısı olan yüksek seviyeli gereksinimlerin üretilmesidir; girdisi ise emniyetle ilgili olanlar dahil yazılıma tahsis edilen sistem gereksinimleri, donanım arayüzü ve sistem mimarisi ile planlama sürecinden gelen yazılım geliştirme planı (Software Development Plan, SDP) ve gereksinim standardıdır.



Girdilerin toplanması ve analizi. İlk adım, yazılıma tahsis edilen sistem gereksinimlerini, emniyet değerlendirmesinden gelen kısıtları ve donanım/yazılım arayüz tanımlarını bir araya getirmektir; bu girdilerin sistem tarafında nasıl oluştuğu [2. Sistem Bağlamında Yazılım](#)

bölümünün konusudur. Analiz sırasında sorulacak sorular bellidir: Bu gereksinim yazılımla mı karşılanacak, donanımla mı? Hata koşulları tanımlanmış mı? Zamanlama ve doğruluk sınırları verilmiş mi? Eksik ya da çelişkili her nokta not edilir.

Yazım ve belgeleme. Gereksinimler tek tek numaralandırılır ve her birine kaynak izi, gereksinim mi tasarım bilgisi mi olduğu, doğrulama yöntemi gibi öznelitlikler eklenir. Türetilmiş gereksinimler ayrıca işaretlenir ve gerekçeleriyle birlikte kaydedilir. Yazım biçimi, öznelitlikler ve işaretleme kuralları kişiden kişiye değişmemelidir; bunları projenin gereksinim standardı (requirements standard) belirler ([5. Yazılım Planlama](#)).

Sisteme geri bildirim. Yazılım ekibi analiz sırasında sistem gereksinimindeki boşluğu bulduğunda bunu kendi başına "düzeltmez"; problem raporu (problem report) ya da benzeri resmî bir kanalla sistem ekibine iletir. Sistem gereksinimi değişir, değişiklik yazılıma yeniden akar. Bu döngü yavaş görünür ama katmanlar arasındaki tutarlılığı koruyan tek yoldur.

Sık düşülen tuzaklar da tanındıktır:

- **Tasarıma erken inmek:** "Yazılım, geçersizlik sayacı 25'e ulaştığında arıza bayrağını kurmalıdır" gibi bir cümle, davranış yerine gerçekleştirimi yazar. Sayaç bir tasarım kararıdır; gereksinim "500 ms boyunca geçersiz veri" gibi gözlemlenebilir bir koşul vermelidir.
- **Tek katmanlı gereksinim:** Yüksek ve düşük seviyeyi tek listede eritmek, kısa vadede zaman kazandırır ama izlenebilirliği (traceability) ve gözden geçirmeyi içinden çıkılmaz hâle getirir. Kodun doğrudan yüksek seviyeli gereksinimlerden yazılabildiği küçük ve basit yazılımlarda tek katman mümkündür; ancak bu, planlarda gerekçelendirilmiş bilinçli bir karar olmalıdır ve o tek katman iki katmanın beklentilerini birden karşılamak zorundadır.
- **Doğrudan koda gitmek:** "Gereksinimi sonra yazarız" yaklaşımı, kodun davranışını gereksinim diye belgelemekle sonuçlanır. Bu durumda gereksinim artık bağımsız bir doğrulama ölçütü değildir; kodun aynasıdır ve koddaki hatayı da birlikte taşır.

Gereksinimden tasarıma ne zaman geçileceğini takvim değil, planlarda tanımlı geçiş kriterleri (transition criteria) belirler; gereksinim süreci için örnekler [Ek A: Örnek Geçiş Kriterleri](#) sayfasındadır.

Yüksek seviyeli gereksinimler neleri kapsar?

Tek tek iyi yazılmış cümleler, eksiksiz bir gereksinim kümesi anlamına gelmez. Gereksinim belgelerinin en sık görülen kusuru yanlış yazılmış madde değil, hiç yazılmamış maddedir: işlevler ayrıntıyla anlatılır, zamanlama, arayüz ve arıza davranışı ise "nasılsa tasarımda netleşir" diye boş bırakılır. Yüksek seviyeli gereksinim kümesinin şu başlıkların her birine yanıt vermesi beklenir:

Kategori	Yanıtladığı soru	Örnek gereksinim
İşlevsel davranış ve çalışma kipleri	Yazılım hangi kipte ne yapar?	Yazılım, bakım verisi indirme isteğini yalnızca YER kipinde kabul etmelidir.

Kategori	Yanıtladığı soru	Örnek gereksinim
Performans	Ne kadar doğru, ne kadar hassas?	Yazılım, gösterilen hava hızını 40–450 knot aralığında ± 1 knot doğrulukla hesaplamalıdır.
Zamanlama	Ne sıklıkta, ne kadar sürede?	Yazılım, hava hızı çıktısını 50 ms (± 5 ms) aralıkla güncellemelidir.
Bellek kısıtları	Hangi bütçenin içinde?	Yazılım, arıza kayıtları için kalıcı bellekte en çok 64 kB yer kullanmalıdır.
Donanım ve yazılım arayüzleri	Hangi veri, hangi biçimde, hangi hızda?	Yazılım, hava hızını çıkış veri kelimesinde 0,25 knot çözünürlükle kodlamalıdır.
Arıza tespiti ve emniyet izleme	Yazılım arızayı nasıl fark eder, ne yapar?	Yazılım, açılışta hesapladığı program belleği sağlama toplamı (checksum) beklenen değerle uyuşmazsa çıkışlarını etkinleştirmemelidir.
Bölümleme	Hangi yazılım parçası hangisinden korunur?	Yazılım, bakım bölümünün (partition) gösterge bölümüne ait belleğe yazmasını engellemelidir.
Anormal girdi ve koşullar	Beklenmeyen durumda ne olur?	Yazılım, 0–512 knot aralığının dışındaki hava hızı değerini geçersiz saymalıdır.

Bir kategori projeye uygulanmıyorsa (örneğin bölümleme yoksa) bunun açıkça söylenmesi, başlığın sessizce atlanmasından iyidir; gözden geçiren kişi "unutulmuş mu, gerekmiyor mu?" sorusuyla baş başa kalmaz. İlk yedi başlık DO-178C'nin yazılım gereksinim verisi için saydığı içerikle örtüşür; sonuncusu orada ayrı bir başlık değildir, gürbüzlük testlerinin dayanağı olduğu için eklenmiştir. Parametre verisi ögesi (parameter data item, PDI) planlanıyorsa yüksek seviyeli gereksinimler ayrıca bu verinin yazılımca nasıl kullanıldığını, yapısını ve veri elemanlarının özniteliklerini de tanımlar ([22. Konfigürasyon Verisi](#)).

Son satır özellikle önemlidir. Gürbüzlük (robustness) testleri, yazılımın geçersiz girdilere ve anormal koşullara verdiği tepkiyi sınar; ancak test mühendisi beklenen sonucu uyduramaz, gereksinimden okur. Aralık dışı değerde, bozuk veride, aşılacak çerçeve süresinde ya da izin verilmeyen durum geçişinde yazılımın ne yapacağı gereksinimde yazmıyorsa, ortada test edilecek bir davranış da yoktur. Normal aralık ve gürbüzlük testlerinin gereksinimden nasıl üretildiği [9. Yazılım Doğrulama](#) bölümünde anlatılır.

İyi gereksinimin özellikleri

Küme eksiksiz olduktan sonra sıra tek tek maddelerin kalitesine gelir. İyi bir gereksinim:

- **tek bir davranışı anlatır:** "ve" ile bağlanmış iki tepki iki gereksinimdir; ayrı test edilir, ayrı izlenir, ayrı değişir;
- **tek biçimde yorumlanır:** yazar, tasarımcı ve test mühendisi aynı cümleden aynı davranışı anlar;

- **doğrulanabilir:** "karşılandı mı?" sorusuna evet ya da hayır denebilir; bunun için sınır, tolerans ve birim yazılıdır;
- **tutarlıdır:** başka bir gereksinimle çelişmez, onu yinelemez ve aynı kavramı aynı adla anar;
- **tasarımdan bağımsızdır:** neyin istendiğini söyler, nasıl yapılacağını değil.

İlk bakışta iyi görünen bir örnek üzerinden gidelim: "Sistem, sensör verisi 500 ms boyunca geçersiz kalırsa güvenli moda geçmelidir." Koşul var, süre var, tepki var; "uygun şekilde" türünden bir kaçamak yok. Yine de test mühendisi bu cümleyle test yazamaz:

- Özne "sistem"dir; bu davranışın yazılımdan mı, donanımdan mı beklendiği belli değildir.
- Hangi sensör? "Geçersiz" neye göre?
- 500 ms'nin toleransı yoktur: 480 ms'de geçiş hata mıdır, 520 ms'de geçmemek hata mıdır?
- Koşul oluştuğundan sonra tepkinin ne kadar sürede gelmesi gerektiği yazmaz.
- "Güvenli mod" tanımlı bir kip midir, yoksa okurun hayal gücüne mi bırakılmıştır?

Eksikler tamamlandığında bölümün başındaki katman tablosunda yer alan yüksek seviyeli gereksinime varılır: özne yazılımdır, veri bellidir, süre toleransı ile verilmiştir, tepki ve tepki süresi yazılıdır. "Geçersiz" sözcüğünün ölçütü (örneğin verinin durum bilgisinin arıza göstermesi ya da verinin beklenen sürede yenilenmemesi) ayrı bir gereksinimde tanımlanır ve her yerde aynı anlamda kullanılır.

Belirsizliğin en tanınmış kaynağı ölçüsüz sözcüklerdir: "uygun şekilde", "yeterli hızda", "gerekli durumlarda", "mümkün olduğunca kısa sürede". Bu sözcükler toplantıda iletişimi kolaylaştırır; gereksinimde ise kararı yazardan alıp kodlayıcıya ve test mühendisine devreder — ikisi de farklı karar verebilir. Yerlerine koşul, zaman, sınır ve tepki içeren açık ifadeler gelmelidir:

Zayıf ifade	Sorun	Yeniden yazım
Yazılım, hava hızını yüksek doğrulukla hesaplamalıdır.	Ölçüt yok	Yazılım, hava hızını 40–450 knot aralığında ± 1 knot doğrulukla hesaplamalıdır.
Yazılım, arıza durumunda mümkün olduğunca kısa sürede uyarı vermelidir.	Hangi arıza, ne kadar süre?	Yazılım, hava hızı arıza bayrağı etkinleştikten sonra en geç 100 ms içinde uyarı çıkışını etkinleştirmelidir.
Yazılım, geçersiz veriyi reddetmeli, arızayı kaydetmeli ve mürettebatı uyarmalıdır.	Üç davranış tek maddede	Üç ayrı gereksinim; her birinin kendi koşulu, süresi, testi ve izi olur.
Yazılım, hiçbir zaman çerçeve süresini aşmamalıdır.	Süre yazmıyor; "hiçbir zaman" testle gösterilemez; aşım olursa ne olacağı belirsiz	İki gereksinim: "Yazılım, her çevrimini 20 ms'lik çerçeve içinde tamamlamalıdır" ve "Yazılım, çerçevesinde tamamlanmayan bir çevrim tespit ettiğinde çerçeve aşımı arızasını kaydetmelidir."

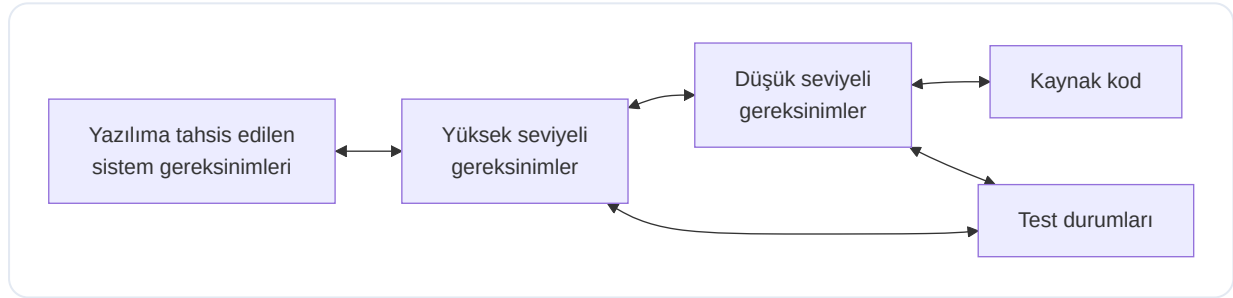
Son satır iki şeyi birden gösterir. Mutlak ifadeler ("her zaman", "asla") yalnızca testle doğrulanamaz; çerçeve süresine uyulduğu zamanlama analiziyle gösterilir ve ölçümle desteklenir, bu yüzden doğrulama yöntemi gereksinim yazılırken düşünülür. Ayrıca kuralın çiğnendiği

durumdaki davranış da ayrı bir gereksinim ister; aksi hâlde anormal koşul yine tanımsız kalır.

Projenin bu konudaki kararları — yasaklı sözcükler, toleransın ve birimin nasıl yazılacağı, cümle kalıbı — tek tek yazarların sağduyusuna bırakılmaz; gereksinim standardına yazılır ve gözden geçirmede o standarda göre denetlenir.

İzlenebilirlik

İzlenebilirlik, iki iş ürününün öğeleri arasındaki kayıtlı ilişkidir: hangi yüksek seviyeli gereksinim hangi sistem gereksinimini karşılıyor, hangi test durumu hangi gereksinimi doğruluyor. DO-178C bu ilişkilerin kaydını — iz verisini (trace data) — başlı başına bir yazılım yaşam döngüsü verisi (software life cycle data) sayar; yani iz kaydı bir yan ürün değil, üretilmesi, gözden geçirilmesi ve kontrol altında tutulması gereken bir çıktıdır. Standart bunun biçimini dayatmaz: iz ayrı bir matriste de, adlandırma kuralıyla ya da verinin içine gömülü atıflarla da tutulabilir.



Zincirin her halkası iki yönde okunur ve iki yön farklı hatayı yakalar:

- **Aşağı yönde** (sistem gereksiniminden yazılım gereksinimine): yazılıma tahsis edilen her sistem gereksinimi en az bir yüksek seviyeli gereksinimle karşılanıyor mu? Karşılığı olmayan sistem gereksinimi, unutulmuş bir işlevdir.
- **Yukarı yönde** (yazılım gereksiniminden sistem gereksinimine): her yüksek seviyeli gereksinimin bir dayanağı var mı? Dayanaksız gereksinim ya türetilmiştir — o zaman öyle işaretlenir, gerekçesi yazılır ve sistem süreçlerine iletilir — ya da kimsenin istemediği bir işlevdir ve kaldırılır.

Yazılım gereksinimi, yazılıma tahsis edilen sistem gereksinimine izlenir. Emniyet hedefi, operasyonel ihtiyaç ya da arayüz kısıtı gibi kaynaklar yazılıma sistem gereksinimleri ve arayüz belgeleri üzerinden ulaşır; bir gereksinim "emniyet için gerekli" diye yazılmış ama hiçbir sistem gereksinimine bağlanamıyorsa, iz uydurulmaz, gereksinim türetilmiş olarak ele alınır.

Zincir gereksinim sürecinde bitmez: tasarım onu düşük seviyeli gereksinimlere ve koda, doğrulama ise test durumlarına, test prosedürlerine ve sonuçlara uzatır. Bu yüzden izlenebilirlik sonradan kurulan bir tablo olmamalıdır. Proje sonunda geriye dönük doldurulan matris, bağlantıların varlığını gösterir ama doğruluğunu göstermez; iz, gereksinim yazılırken kurulur ve gözden geçirmede içeriğiyle birlikte sınanır. "Bağlantı var" demek yetmez; bağlanan iki madde

gerçekten aynı davranıştan mı söz ediyor? Otoritenin denetimde yaptığı da budur: rastgele bir gereksinim seçer ve zinciri iki yönde yürür ([SW SOI-2](#) sayfasındaki iz sürme alıştırmaları aynı yolu izler).

Gereksinimlerin gözden geçirilmesi

Gereksinim hatasını yakalamanın en ucuz anı, gereksinim henüz kâğıt üzerindeyken yapılan gözden geçirmedir. Aynı hata tasarımda yakalanırsa maliyet katlanır, testte yakalanırsa daha da katlanır, sahada ortaya çıkarsa artık maliyet değil emniyet konuşulur. Bu yüzden akran gözden geçirmesi (peer review), gereksinim sürecinin süsü değil çekirdeğidir.

DO-178C, yüksek seviyeli gereksinimlerin doğrulanmasında yedi şeye bakılmasını ister: sistem gereksinimleriyle uyum, doğruluk ve tutarlılık, hedef bilgisayarla (target computer) uyumluluk, doğrulanabilirlik, gereksinim standardına uygunluk, sistem gereksinimlerine izlenebilirlik ve algoritmaların doğruluğu. Bunların bir kısmı tek tek cümlelere bakılarak, bir kısmı ise ancak küme bir bütün olarak ve sistem gereksinimleriyle yan yana okunarak değerlendirilebilir. İyi bir kontrol listesi yedisini de somut sorulara çevirir.

İşleyen bir gözden geçirme pratiğinde şunlar bulunur:

- **Doğru katılımcılar:** Yazarın kendisi, bir başka gereksinim/yazılım mühendisi, bir sistem veya emniyet temsilcisi ve mümkünse bir test mühendisi. Test mühendisi masadaki en değerli kişilerden biridir; "bunu nasıl test ederim?" sorusu belirsizliği herkesten önce yakalar.
- **Hazırlık süresi:** Katılımcıların malzemeyi toplantıdan önce okuması. Toplantıda ilk kez okunan gereksinim, gözden geçirilmiş sayılmaz.
- **Makul kapsam:** Tek oturumda yüzlerce gereksinim "geçirilmez". Yorgun gözden geçirici her şeyi onaylar; kısa ve odaklı oturumlar daha çok hata bulur.
- **Kontrol listesi (checklist):** Değerlendirmenin kişisel zevke değil ölçüte dayanmasını sağlar.

Tipik bir gereksinim gözden geçirme kontrol listesinden satırlar:

Soru	Aradığı hata
Tahsis edilen sistem gereksiniminin istediği davranışı eksiksiz karşılıyor mu?	Eksik ya da yanlış yorumlanmış işlev
Gereksinim tek bir davranışı mı anlatıyor?	Bileşik, bölünmesi gereken madde
Girdi, koşul ve beklenen tepki açık mı?	Belirsizlik, örtük varsayım
Ölçülebilir sınır/tolerans verilmiş mi?	Test edilemezlik
Kaynağına izlenebilir mi; izlenemiyorsa türetilmiş olarak işaretli mi?	Dayanaksız gereksinim, bildirilmemiş türetilmiş gereksinim
Başka bir gereksinimle çelişiyor veya çakışıyor mu?	Tutarsızlık, tekrar

Soru	Aradığı hata
Hata ve sınır koşulları ele alınmış mı?	Yalnızca "mutlu yol" tanımı
Hedef donanımın yetenekleriyle bağdaşır mı (çözünürlük, hız, bellek)?	Gerçekleştirilemeyecek gereksinim
Formül, ölçekleme ve sınır geçişleri doğru mu?	Yanlış algoritma, süreksizlik noktasında hatalı davranış
Gereksinim standardına ve terim sözlüğüne uyuyor mu?	Standart sapması, aynı kavrama iki ad

Neye bakılacağı ve kimin bakacağı yazılım seviyesine (software level) bağlıdır. Bu başlığın girişinde sayılan yedi ölçüt standartta birer doğrulama hedefidir (objective). Yedisi de Seviye A ve B'de geçerlidir; Seviye C'de hedef bilgisayarla uyumluluk aranmaz, Seviye D'de ise yalnızca üçü kalır: sistem gereksinimleriyle uyum, doğruluk ve tutarlılık, izlenebilirlik. Bağımsızlık (independence), yani hedefin yazardan başka biri tarafından karşılanması, Seviye A ve B'de üç hedefte aranır: sistem gereksinimleriyle uyum, doğruluk ve tutarlılık, algoritmaların doğruluğu. Seviye C ve D'de bu hedeflerde bağımsızlık şart değildir. Bağımsızlığın nasıl sağlandığı ve kayıtlarda nasıl gösterildiği [9. Yazılım Doğrulama](#) bölümünde ele alınır.

Gözden geçirmenin ikinci işlevi kanıt üretmektir. Bulunan her bulgu, verilen karar (düzeltildi, reddedildi, ertelendi) ve kapanış durumu kayda geçirilir. Bu kayıtlar, sertifikasyon otoritesine ve kalite güvencesine "gereksinimler gerçekten gözden geçirildi" iddiasının nesnel kanıtıdır; katılım aşaması (Stage of Involvement, SOI) denetimlerinde, özellikle geliştirme verisine bakılan SOI-2'de ilk istenen malzemeler arasındadır. İmzalanmış ama bulgu içermeyen yüzlerce kayıt ise tersine şüphe uyandırır: hiç hata bulamayan bir gözden geçirme süreci, büyük olasılıkla hata aramamıştır.

Son bir pratik not: gözden geçirme, yazarı savunmaya iten bir sınav değildir. Bulgu sayısı yazarın karnesi gibi kullanılmaya başlandığı anda ekip bulgu saklamayı öğrenir ve sürecin değeri sıfırlanır. Amaç kişiyi değil metni sınamaktır.

Gereksinim yönetimi ve prototipleme

Gereksinimler yazılıp onaylandığı anda donmaz; proje boyunca değişir. Değişimin kendisi sorun değildir — kontrolsüz değişim sorundur. Bu yüzden gereksinimler, konfigürasyon yönetiminin (configuration management) kapsamına giren birer konfigürasyon ögesi olarak ele alınır: her gereksinim kümesinin bir sürümü vardır ve belirli bir olgunluk noktasında **temel çizgi (baseline)** olarak dondurulur.

Temel çizgi alındıktan sonra işleyiş değişir:

- Temel çizgi öncesinde yazar, gereksinimi görece serbestçe düzenleyebilir; gözden geçirme bulguları gözden geçirme kaydında izlenip kapatılır.

- Temel çizgi sonrasında her değişiklik bir değişiklik talebi ya da problem raporu ile başlar, etkisi analiz edilir, yetkili bir kurul (çoğu projede değişiklik kontrol kurulu, change control board) tarafından onaylanır ve ancak ondan sonra uygulanır.

Bu mekanizmanın bütün iş ürünleri için nasıl işlediği [10. Yazılım Konfigürasyon Yönetimi](#) bölümünün konusudur.

Değişiklik etki analizi (change impact analysis) bu zincirin en kritik halkasıdır. Bir gereksinim değiştiğinde yalnızca o madde değil, ona izlenebilirlikle bağlı düşük seviyeli gereksinimler, tasarım öğeleri, kod ve testler de gözden geçirilmek zorundadır. İz verisi güncel tutulmuşsa etki analizi bir sorgudur; tutulmamışsa arkeolojik kazıya dönüşür. Gereksinim yönetim araçlarının (DOORS ve benzerleri) asıl değeri burada ortaya çıkar: sürüm geçmişi, öznitelikler ve bağlantılar tek yerde durur.

Pratik bir uyarı: "küçük değişiklik" diye bir kategori icat etmeyin. Deneyim, en pahalı hataların "iki kelime değişti, analize gerek yok" denilen düzeltmelerden çıktığını gösteriyor. Sürecin ağırlığı değişikliğin etkisine göre ölçeklenebilir; ama etkinin büyüklüğüne kişisel sezgi değil, analiz karar vermelidir.

Prototiplemeye gelince: gereksinimler her zaman masa başında olgunlaşmaz. Bir kontrol algoritmasının kazançları, bir ekran düzeninin okunabilirliği ya da bir arayüzün zamanlama davranışı çoğu kez ancak çalışan bir deneme üzerinde görülür. Prototip, "ne istediğimizi görmeden yazamayız" sorununa meşru bir cevaptır ve gereksinim belirsizliğini erkenden azaltır.

Riski de aynı yerde başlar: prototip kodunun sertifikasyon kanıtı olmadan ürüne sızması. Prototip, gereksinim ve tasarım disiplini uygulanmadan, hızla yazılmış koddur; DO-178C süreç kanıtlarına (gözden geçirme, izlenebilirlik, yapısal kapsam) sahip değildir. "Zaten çalışıyor, baştan yazmak israf" cümlesi duyulduğunda proje tehlikeli bir yola girmiştir. Sağlıklı kullanım için:

- Prototipin amacı baştan yazılı olarak sınırlanır: neyi öğrenmek için yapılıyor?
- Prototipten beklenen çıktı **kod değil bilgidir**: öğrenilenler gereksinimlere yazılır, prototip kenara konur.
- Prototip kodu ürüne taşınacaksa bu bir istisna değil bilinçli bir karardır ve kod, ürün koduyla aynı geliştirme ve doğrulama sürecinden eksiksiz geçirilir: gereksinimleri ve tasarımı yazılır, izlenebilirliği kurulur, kod standarda göre gözden geçirilir ve gereksinim tabanlı testlerle doğrulanır. Var olan koddan geriye doğru gereksinim ve tasarım üretmenin kendine özgü riskleri vardır; [25. Tersine Mühendislik](#) bölümü bunları ele alır.

Doğru kullanılan prototip gereksinimi olgunlaştırır; yanlış kullanılan prototip, gereksinim sürecini atlamanın kılıfı olur.

Gereksinim mühendisinin rolü

Gereksinim yazmak çoğu projede "boşta kalan mühendise verilen" bir iş gibi görülür; oysa emniyet-kritik yazılımda bu, projenin kaderini belirleyen görevlerden biridir. Gereksinim mühendisi (requirements engineer), sistem ekibi ile yazılım ekibi arasındaki tercümandır: bir yanda uçuş mekaniği, sensör davranışı ve emniyet hedefleriyle konuşan sistem mühendisleri, diğer yanda veri yapıları ve zamanlama kısıtlarıyla düşünen yazılım geliştiriciler vardır. İkisinin de dilini konuşamayan bir gereksinim, iki tarafın da yanlış anlayacağı bir metne dönüşür.

Deneyim, iyi bir gereksinim mühendisinde şu becerilerin bir arada bulunduğunu gösteriyor:

- **Alan bilgisi (domain knowledge):** Yazılan gereksinimin arkasındaki fiziksel ve operasyonel gerçeği anlamak. ARINC 429 etiketinin ne taşıdığını, bir pitot-statik sensörün nasıl bozulduğunu bilmeyen kişi, o sensörle ilgili hata koşulunu doğru yazamaz.
- **Yazılı iletişim:** Gereksinim bir kez yazılır, yüzlerce kez okunur. Kısa, tek yorumlu ve dilbilgisi açısından tutarlı cümle kurabilmek başlı başına bir beceridir.
- **Soyutlama:** "Ne" ile "nasıl"ı ayırabilmek; davranışı tasarım kararlarına bulaştırmadan ifade edebilmek. Bu, deneyimli geliştiricilerin bile zorlandığı bir alışkanlıktır, çünkü geliştirici refleksi hemen çözüme atlamaktır.
- **Sorgulama:** Sistem gereksinimindeki boşluğu, çelişkiyi ve söylenmemiş varsayımı fark edip "burada ne olmalı?" diye sormaktan çekinmemek. En pahalı hatalar, kimsenin sormaya cesaret edemediği sorulardan doğar.
- **Sabır ve düzen:** Gereksinim çalışması, kod yazmak kadar "görünür" ilerlemez. Yüzlerce maddeyi tutarlı biçimde numaralandırmak, öznelilikleriyle birlikte yönetmek disiplin ister.

Ekip içindeki konumuna gelince: gereksinim mühendisi ne sistem ekibinin sekreteri ne de yazılım ekibinin sözcüsüdür. Sistem gereksinimlerini olduğu gibi kopyalayan kişi değer üretmez; yazılım ekibinin tasarım tercihlerini gereksinim diye yazan kişi ise katmanları bozar. Sağlıklı projelerde gereksinim mühendisi:

- sistem ekibiyle birlikte girdileri analiz eder ve belirsizlikleri geri bildirir,
- yazılım mimarı ve geliştiricilerle uygulanabilirliği tartışır,
- test ekibiyle erken temas kurar; çünkü test edilemeyen gereksinimi ilk fark eden genellikle test mühendisidir.

Küçük projelerde bu rolü ayrı bir kişi üstlenmeyebilir; geliştirici ve gereksinim yazarı aynı kişi olabilir. Bu durumda kritik olan, kişinin hangi şapkayla yazdığını bilmesi ve gözden geçirmenin başka bir çift göz tarafından yapılmasıdır. Standart yazar dışı gözden geçirmeyi yalnızca belirli seviye ve hedeflerde şart koşar; yine de her seviyede iyi uygulamadır, çünkü yazarın kendi varsayımını kendi metninde görmesi zordur.

Bu bölümden akılda kalması gerekenler

- Yüksek seviyeli gereksinim yazılımın dışarıdan gözlenen davranışını, düşük seviyeli gereksinim kodun doğrudan yazılabileceği ayrıntıyı tanımlar; ilki gereksinim sürecinin, ikincisi tasarım sürecinin çıktısıdır.
- Türetilmiş gereksinim, üst seviyeye izlenemeyen ya da üst seviyede belirtilenin ötesinde davranış tanımlayan gereksinimdir; gerekçesi yazılır ve sistem emniyet değerlendirme süreci dahil sistem süreçlerine iletilir.
- Gereksinim kümesi yalnızca işlevleri değil performansı, zamanlamayı, arayüzleri, arıza tespitini ve anormal koşullardaki davranışı da kapsar; gürbüzlük testlerinin dayanağı bu son gruptur.
- İyi gereksinim tekil, tek yorumlu, toleransıya ölçülebilir ve tasarımdan bağımsızdır; eksik ve çelişkili sistem girdileri yazılımda "düzeltilmez", resmî kanalla sisteme geri bildirilir.
- İzlenebilirlik iki yönlüdür: aşağı yön unutulmuş işlevi, yukarı yön dayanaksız gereksinimi yakalar; iz verisi güncel tutulursa değişiklik etki analizi bir sorguya iner.
- Akran gözden geçirmesi kontrol listesiyle yapılır, kayıtları sertifikasyon kanıtıdır; yedi doğrulama hedefinden üçünde Seviye A ve B'de bağımsızlık şarttır, yazar dışı gözden geçirme ise her seviyede iyi uygulamadır.
- Temel çizgi sonrası her değişiklik etki analizi ve onaydan geçer; prototipin çıktısı kod değil bilgidir ve ürüne sızması yönetilmesi gereken bir risktir.

7. Yazılım Tasarımı

Tasarım, gereksinimlerin uygulanabilir bir yazılım yapısına dönüştüğü katmandır. Bu bölüm, tasarımın iki çıktısını (yazılım mimarisi ve düşük seviyeli gereksinimler), arayüzlerin, durumların ve hata davranışının nasıl tasarlandığını ve DO-178C'nin tasarımdan beklediği doğrulamayı anlatır.

İyi tasarım, kodun yalnızca çalışmasını değil, okunmasını, doğrulanmasını ve bakımının yapılmasını da kolaylaştırır. Kritik nokta, tasarım kararlarının gereksinimlerle açık biçimde bağlanmasıdır.

Tasarımın rolü

Tasarım, bir gereksinimin "nasıl" tarafını görünür kılar. Yazılım ekibi burada işlevleri alt bileşenlere ayırır, veri akışını tanımlar, durum geçişlerini belirler ve hata işleme kurallarını netleştirir.

Bu aşama, yalnızca mimari çizim üretmek için değil, doğrulama için uygun bir yapı kurmak için vardır. İyi tasarım, test ekibinin hangi davranışı hangi koşul altında gözlemlemesi gerektiğini anlamasını sağlar.

DO-178C gözüyle tasarım iki iş ürününden oluşur: **yazılım mimarisi** (software architecture; bileşenler ve aralarındaki ilişkiler) ve **düşük seviyeli gereksinimler** (low-level requirements; kodun doğrudan gerçekleştireceği davranış tanımları). Düşük seviyeli gereksinimlerin yüksek seviyeli gereksinimlere (high-level requirements) izlenebilir, mimarinin ise onlarla uyumlu olması beklenir. Yüksek seviyeli gereksinimlerin nasıl yazıldığı [6. Yazılım Gereksinimleri](#) bölümünde anlatılmıştır.

Tasarım tek yönlü bir aktarım da değildir. Bir gereksinimin eksik, çelişkili ya da gerçekleştirilemez olduğunu çoğu zaman ilk fark eden kişi tasarımcıdır. Böyle bir durumda boşluk tasarımda sessizce doldurulmaz; gereksinim sürecine geri bildirilir ve gereksinimin kendisi düzeltilir.

Tasarım tanımı ve düşük seviyeli gereksinimler

Tasarım sürecinin ürettiği yazılım yaşam döngüsü verisi (software life cycle data) **tasarım tanımıdır** (design description). Adı tek bir belgeyi çağırırsa da içerik pratikte bir mimari dokümanına, gereksinim yönetim aracındaki düşük seviyeli gereksinimlere ve bir veri sözlüğüne (data dictionary) dağılır; önemli olan paketleme değil, içeriğin eksiksiz olmasıdır. Tasarım tanımında bulunması beklenenler şöyle özetlenebilir:

- **Mimari:** bileşenler, sorumlulukları ve arayüzleri; veri akışı ve kontrol akışı.

- **Düşük seviyeli gereksinimler:** algoritmalar ve veri yapıları dahil, yüksek seviyeli gereksinimlerin nasıl karşılandığının ayrıntılı tanımı; tasarım sırasında doğan türetilmiş gereksinimler (derived requirements).
- **Girdi ve çıktı tanımları:** iç ve dış arayüzlerdeki verilerin sözlüğü.
- **Kaynak kullanımı:** işlemci zamanı ve bellek gibi kaynakların sınırları, bırakılan pay ve bunların nasıl ölçüleceği.
- **Çizelgeleme (scheduling) ve iletişim:** görev yapısı, kesmeler, görevler ya da işlemciler arası iletişim mekanizmaları.
- **Özel mekanizmalar:** varsa yazılım bölümlenmesi (software partitioning) yöntemi, devre dışı bırakılmış kodun (deactivated code) etkinleşmesini önleyen düzen, yazılım yükleme ya da kullanıcı tarafından değiştirilebilir yazılım (user-modifiable software) için seçilen tasarım yöntemleri.
- **Bileşenlerin kaynağı:** hangi bileşenin yeni, hangisinin önceden geliştirilmiş olduğu.
- **Tasarım gerekçeleri:** emniyetle ilgili sistem gereksinimlerine dayanan tasarım kararlarının nedeni.

Bu içeriğin en çok tartışılan kısmı düşük seviyeli gereksinimlerin ayrıntı düzeyidir. Ölçüt şudur: kodlayıcı, düşük seviyeli gereksinimi ve mimariyi okuyarak kodu başka bilgiye ihtiyaç duymadan, kendi başına davranış kararı vermeden yazabilmelidir. Bunun iki sınırı vardır. Fazla soyut kalan gereksinim ("uyarı mantığını uygula") kararı koda bırakır; kodda gereksinimde yazmayan davranış belirir ve gereksinim tabanlı test (requirements-based testing) onu sınamaz. Fazla ayrıntılı gereksinim ise kodun düzyazıya çevrilmiş kopyasına dönüşür; ondan türetilen test kodun doğruluğunu değil kendisiyle tutarlılığını gösterir, her kod değişikliği de bir gereksinim değişikliğine dönüşür. Ayrıntı düzeyi bu yüzden kişisel tercihe bırakılmaz; tasarım standardında tanımlanır (bkz. [5. Yazılım Planlama](#)).

Aşağıdaki örnek, bölüm boyunca kullanılacak aşırı hız uyarısı işlevinde bir yüksek seviyeli gereksinimin düşük seviyeli gereksinimlere nasıl indirildiğini gösterir. Kimliklerdeki HLR ve LLR, yüksek ve düşük seviyeli gereksinimin yerleşik kısaltmalarıdır; kimlikler ve sayılar temsilîdir:

Kimlik	Gereksinim	İz
HLR-42	Havahızı aşılmaması gereken hızı (VNE) geçtiğinde yazılım, en geç 200 ms içinde aşırı hız uyarısını etkinleştirmelidir.	Sistem gereksinimi
LLR-42-1	Uyarı mantığı her 50 ms'lik çevrimde bir kez çalışır ve o çevrimde doğrulanmış havahızı örneğini kullanır.	HLR-42
LLR-42-2	Havahızı geçerliyse ve <code>VNE_LIMIT</code> değerinden büyükse uyarı çıkışı ETKİN yapılır.	HLR-42
LLR-42-3	Uyarı ETKİN iken havahızı <code>VNE_LIMIT</code> değerinin 3 knot altına inmeden uyarı kaldırılmaz.	HLR-42; türetilmiş

İlk iki satır üst gereksinimin "ne zaman" ve "hangi koşulda" kısımlarını koda yazılabilir hâle getirir. Üçüncü satır farklıdır: yüksek seviyeli gereksinim uyarının ne zaman kalkacağını söylemez; histerezis (hysteresis), hız sınır çevresinde dalgalanırken uyarının açılıp kapanmasını önlemek

için tasarımda alınmış bir karardır. Üst gereksinime bağlansa da orada belirtilenin ötesinde davranış tanımladığı için türetilmiş gereksinimdir; gerekçesiyle kaydedilir ve emniyet değerlendirmesi dahil sistem süreçlerine bildirilir. Filtre katsayıları, zaman aşımı değerleri ve teyit süreleri tasarımda en sık karşılaşılan türetilmiş gereksinim kaynaklarıdır.

Yüksek seviyeli gereksinimler kodun doğrudan yazılabileceği ayrıntıdaysa tek gereksinim katmanıyla çalışmak mümkündür; o durumda bu gereksinimler düşük seviyeli gereksinim işlevini de görür ve her iki katman için beklenen doğrulama onlara uygulanır. DO-178C bu yolu açık bırakır, ama tek gereksinim katmanı üreten bir süreç için başvuru sahibinden gerekçe istenebileceğini de not eder. Otoriteler katmanların birleştirilmesine temkinli yaklaşır: aradaki inceltme adımı görünmez olduğunda izlenebilirlik analizinin boşlukları yakalaması zorlaşır. Böyle bir yaklaşım yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) açıkça yazılmalı ve otoriteyle erkenden konuşulmalıdır.

Tasarım yaklaşımları

Aviyonik yazılımda iki ana tasarım geleneği vardır: fonksiyonel ayrıştırmaya (functional decomposition) dayalı **yapısal tasarım** (structured design) ve **nesne yönelimli tasarım** (object-oriented design). İkisi de aynı soruyu yanıtlar — sistem hangi parçalardan oluşacak ve bu parçalar nasıl konuşacak — ama parçalama eksenleri farklıdır.

Yapısal tasarımda sistem, "ne yapıyor" sorusuna göre bölünür: girdi topla, doğruyla, hesapla, çıktı üret. Her modül bir işlevi temsil eder; veriler modüller arasında açıkça taşınır. Nesne yönelimli tasarımda ise sistem, "neyi yönetiyor" sorusuna göre bölünür: her birim, bir veri kümesini ve onun üzerinde geçerli işlemleri birlikte kapsüller.

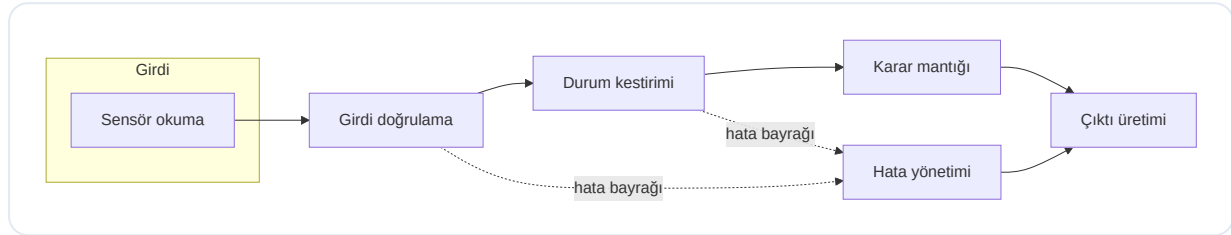
Ölçüt	Yapısal tasarım	Nesne yönelimli tasarım
Parçalama eksen	İşlevler	Veri ve sorumluluklar
Veri görünürlüğü	Açık veri akışı, izlemesi kolay	Kapsülleme, iç durum gizli
Çalışma zamanı davranışı	Statik, öngörülebilir çağrı yapısı	Dinamik bağlama kullanılırsa analiz zorlaşır
Doğrulama etkisi	Kontrol akışı doğrudan görünür	Sanal çağrılar ve kalıtım ek analiz ister
Aviyonikte yaygınlık	Çok yaygın (özellikle C ile)	Sınırlı; genellikle kısıtlanmış alt kümelerle

Emniyet-kritik projelerde C dilinin baskın olmasının bir nedeni de budur: yapısal tasarım, çağrı grafiğinin ve veri akışının derleme zamanında tamamen belirli olmasını kolaylaştırır. Nesne yönelimli teknikler yasak değildir; ancak kalıtım (inheritance), çok biçimlilik (polymorphism) ve dinamik bellek gibi özellikler, doğrulama yükünü artırdığı için genellikle bilinçli olarak kısıtlanır. DO-178C'nin nesne yönelimli teknolojiye ayrılmış eki (DO-332), bu ek analiz konularını ayrıca ele alır; ayrıntısı [15. DO-332 ve Nesne Yönelimli Teknoloji ve İlgili Teknikler](#) bölümündedir.

Hangi yaklaşım seçilirse seçilsin, tasarım en az iki görünümle anlatılmalıdır:

- **Veri akışı (data flow) görünümü:** hangi verinin nereden üretilip nerede tüketildiğini gösterir. Sensör girdisinden aktüatör çıktısına giden zinciri takip etmeyi sağlar.
- **Kontrol akışı (control flow) görünümü:** hangi bileşenin hangi sırayla ve hangi koşulda çalıştığını gösterir. Zamanlama, kesmeler ve görev öncelikleri burada görünür olur.

Aşağıdaki diyagram, tipik bir "algıla, karar ver, çıktı üret" zincirinin veri akışı görünümüdür. Kontrol akışı görünümünün örnekleri (çevrim sırası ve durum diyagramı) bir sonraki kısımdadır.



Yaklaşım seçiminde pratikte şu ölçütler belirleyicidir:

- **Yazılım seviyesi (software level):** Seviye A ve B projelerinde analiz edilebilirlik öne çıkar; dinamik yapılar sınırlanır.
- **Ekip ve araç ekosistemi:** Derleyici, statik analiz ve kapsama araçlarının dili ve yapıları ne kadar desteklediği.
- **Mevcut kod ve yeniden kullanım:** Önceki projelerden gelen bileşenlerin tasarım tarzı, yeni tasarımı fiilen yönlendirir.
- **Sertifikasyon geçmişi:** Otoriteyle daha önce kabul görmüş bir yaklaşım, yeni ve savunulması zor bir yaklaşıma çoğu zaman tercih edilir.

Mimari: bileşenler, arayüzler ve durumlar

Mimari kararlarının çoğu üç soruya iner: sorumluluk hangi bileşende, bileşenler birbirine hangi arayüzden bağlı ve yazılım hangi durumda ne yapıyor. Bu kararlar kodun düzenini belirlediği kadar test stratejisini de belirler; neyin tek başına sınanabileceği, hangi arayüzün entegrasyon testinde işletileceği burada kesinleşir.

Bileşenler ve katmanlar. Gömülü aviyonik yazılımda yaygın düzen katmanlı mimaridir (layered architecture): donanıma dokunan sürücüler altta, zamanlama ve haberleşme gibi ortak hizmetler ortada, uygulama mantığı üsttedir. Uygulama katmanı da kendi içinde işleve göre ayrılır; veri toplama, girdi doğrulama, karar mantığı, çıktı üretimi ve hata yönetimi ayrı bileşenlerdir. Bu düzeni değerli kılan çizimi değil, kurallarıdır: bağımlılık yalnızca üstten alta doğrudur, katman atlanmaz ve donanım yazmaçlarına (register) yalnızca sürücü katmanı erişir. Kural tutulduğunda donanım değişikliği sürücü katmanında kalır, bir bileşendeki hata komşusuna ancak tanımlı arayüzden yayılabilir ve değişiklik etki analizinin (change impact analysis) kapsamı küçülür.

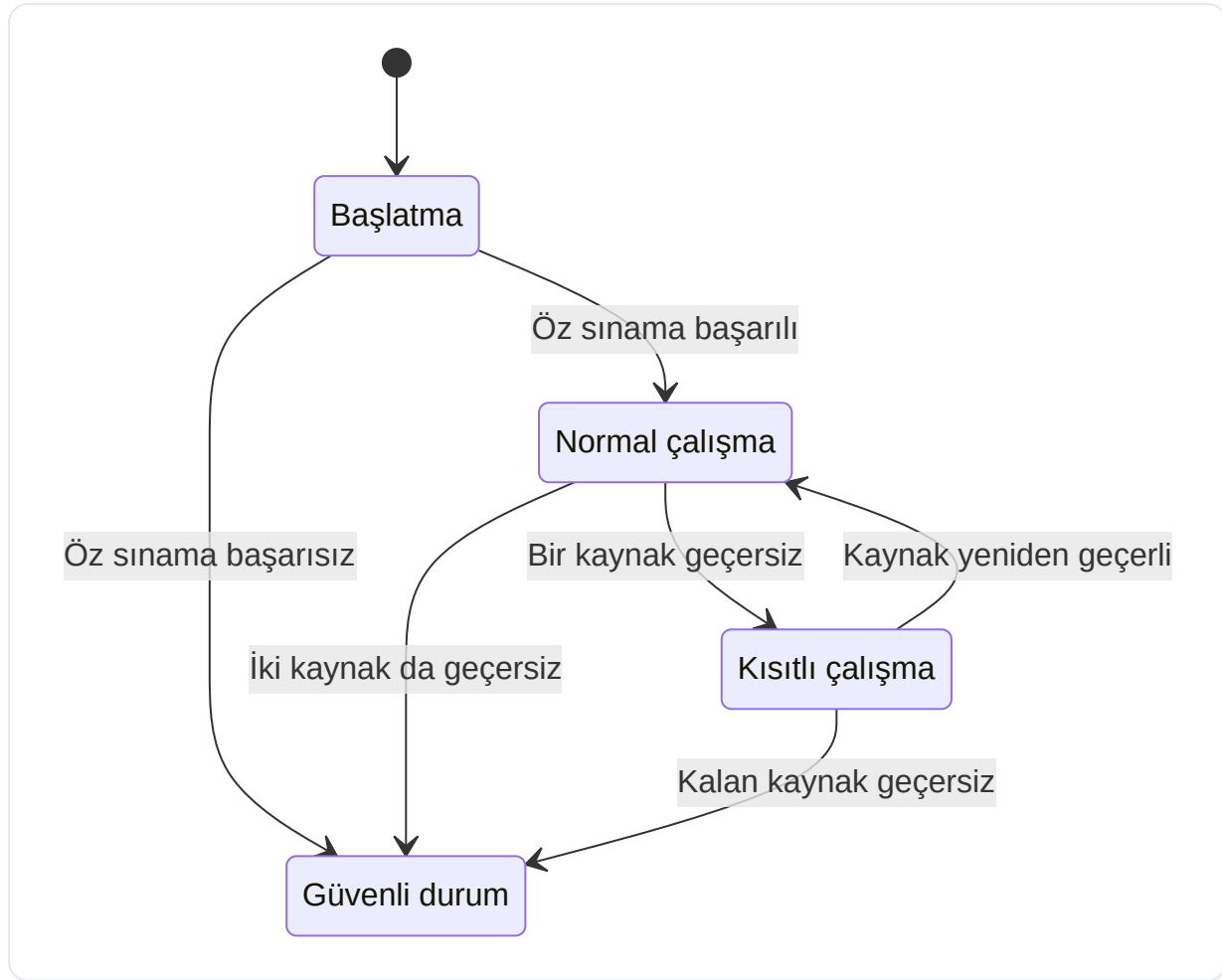
Arayüzler. Arayüz veri taşıdığı kadar sorumluluk sınırını da çizer. Net olmayan arayüz, bileşenlerin birbirinin iç ayrıntısına bağımlı hâle gelmesine yol açar. Entegrasyonda sık görülen hata türü de iki tarafın aynı veriyi farklı yorumlamasıdır: biri knot gönderir, öbürü metre/saniye bekler; biri "geçersiz" anlamında sıfır yazar, öbürü sıfırı ölçüm sanır. Bu yüzden her arayüz verisi için en azından şu öznitelikler tasarımda yazılı olmalıdır (değerler örnektir):

Öznitelik	Örnek: doğrulanmış havahızı
Üreten ve tüketen	Girdi doğrulama üretir; uyarı mantığı tüketir
Tip ve birim	32 bit kayan noktalı; knot
Geçerli aralık ve çözünürlük	0–450 knot; 0,1 knot
Güncelleme hızı	50 ms'de bir
Tazelik sınırı	150 ms'den eski örnek geçersiz sayılır
Geçerlilik bilgisi	Ayrı bir geçerli/geçersiz bayrağı; geçersizken değer kullanılmaz
Başlangıç değeri	İlk geçerli örnek gelene kadar geçersiz

Tabloda değerin kendisinden çok geçerlilik ve tazelik satırları belirleyicidir: tüketen bileşenin veriyi kullanmadan önce neye bakacağı ve veri kullanılamazsa ne yapacağı arayüz tanımının parçasıdır.

Kontrol akışı ve durumlar. Kontrol akışının en yalın biçimi sabit sıralı çevrimdir: örnekte her 50 ms'lik çevrimde sırasıyla sensör okuma, girdi doğrulama, durum yönetimi, uyarı mantığı ve çıktı üretimi çalışır. Sıra tasarımda yazılı olduğunda verinin tazeliği ve uçtan uca gecikme hesapla gösterilebilir; 200 ms'lik tepki süresinin kaç çevrime karşılık geldiği buradan okunur. Görevler, öncelikler ve kesmeler devreye girdiğinde aynı bilgi çizelgeleme tasarımıyla verilir.

Davranışın kipe bağlı olduğu yerde durum makinesi (state machine) kullanılır. Emniyet-kritik yazılımda en azından normal çalışma, kısıtlı çalışma (degraded mode) ve güvenli durum (safe state) ayrımının, aralarındaki geçiş koşullarıyla birlikte açıkça tanımlanması gerekir. İki havahızı kaynağından beslenen örnek için:



Durum diyagramının değeri eksikleri görünür kılmasıdır. Her durum için her olayın karşılığı sorulur ("kısıtlı çalışmadayken ikinci kaynak da bozulursa?") ve tanımsız kalan her çift ya bir düşük seviyeli gereksinimle doldurulur ya da bilerek yok sayıldığı yazılır. Diyagramdaki her geçiş bir düşük seviyeli gereksinime, o da en az bir test durumuna (test case) karşılık gelir; izin verilmeyen geçişler ise gürbüzlük (robustness) testlerinin konusudur.

Hata davranışının tasarımı

İyi tasarım, hata durumunu sonradan eklenmiş bir istisna gibi değil, beklenen bir işletim hâli gibi ele alır. Arıza karşısında yazılımdan ne beklendiği — hangi durumun güvenli sayıldığı dahil — yazılım ekibinin tercihi değildir; sistem gereksinimlerinden ve onların arkasındaki emniyet değerlendirmesinden gelir (bkz. [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#)). Tasarımın işi, bu beklentiye dört soruya ayrıntılı yanıt vererek gerçekleştirmektir. Havahızı kaynağının arızalandığı senaryoda yanıtlar şöyle olabilir:

Soru	Örnekteki tasarım kararı
Hata nasıl algılanır?	Aralık denetimi (0–450 knot dışı değer), tazelik denetimi (150 ms'den eski örnek) ve sensörün kendi durum bildirimi

Soru	Örnekteki tasarım kararı
Kim bilgilendirilir?	Girdi doğrulama geçerlilik bayrağını düşürür; hata yönetimi bileşeni arızayı kaydeder ve durum yönetimine bildirir
Hangi duruma geçilir?	Tek kaynak kaybında kalan kaynakla kısıtlı çalışma; iki kaynak da kaybedilirse güvenli durum (uyarı işlevi "kullanılmıyor" olarak bildirilir)
Geçiş nasıl sınırlanır?	Arıza art arda üç çevrimde görülmeden durum değişmez; normale dönüş için kaynak belirli bir süre geçerli kalmalıdır; güvenli durumdan yalnızca yeniden başlatmayla çıkılır

Bu konuda iki tasarım hatası sık görülür. İlki, "son geçerli değeri kullan" kuralını süre sınırı olmadan yazmaktır: veri akışı kesildiğinde yazılım bayat değerle normal çalışıyormuş gibi görünür. İkincisi, her bileşenin hataya kendi içinde, ötekilerden habersiz tepki vermesidir: aynı arıza bir yerde yok sayılır, başka bir yerde yeniden başlatmaya yol açar. Hata tepkisinin tek bir bileşende toplanması ve ortak bir geri dönüş yoluna bağlanması hem davranışı tutarlı kılar hem de sınanacak yol sayısını azaltır.

Tasarımda yazılan her algılama eşiği, teyit süresi ve geçiş koşulu aynı zamanda bir gürbüzlük testinin girdisidir. Bu değerler tasarımda yoksa test ekibi neyi sınavacağını koddan öğrenmek zorunda kalır; bu da gereksinim tabanlı testin mantığına aykırıdır.

İyi tasarımın nitelikleri

İyi tasarımın en klasik iki ölçütü, **bağlaşım** (coupling) ve **iç uyum** (cohesion) kavramlarıdır. Bağlaşım, iki modülün birbirine ne kadar bağımlı olduğunu; iç uyum, bir modülün içindeki parçaların ne kadar tek bir amaca hizmet ettiğini anlatır. Hedef her zaman aynıdır: **bağlaşım düşük, iç uyum yüksek** olmalıdır.

Bağlaşım düşükse bir modüldeki değişiklik komşularını daha az etkiler; değişiklik etki analizi ve yeniden doğrulama kapsamı küçülür. İç uyum yüksekse modülün ne yaptığı tek cümleyle anlatılabilir; gözden geçirme ve test tasarımı kolaylaşır.

Bağlaşımın en sorunlu biçimi, modüllerin ortak global değişkenler üzerinden örtük haberleşmesidir. Aşağıdaki C örneği farkı gösterir:

```

/* float32_t: projenin sabit genişlikli kayan nokta tipi. */

/* Sıkı bağlaşım: iki modül global durumu paylaşıyor. */
extern float32_t g_havahizi_knot;      /* sensör modülü yazar, herkes okur */

void uyari_kontrol_global(void)
{
    if (g_havahizi_knot > VNE_LIMIT) { /* kim, ne zaman güncelledi belli değil */
        uyari_ver(UYARI_ASIRI_HIZ);
    }
}

/* Gevşek bağlaşım: veri, açık bir arayüzden parametreyle taşınıyor. */
void uyari_kontrol(float32_t havahizi_knot)
{
    if (havahizi_knot > VNE_LIMIT) {
        uyari_ver(UYARI_ASIRI_HIZ);
    }
}

```

İkinci biçimde fonksiyonun tüm girdisi imzasında görünür; birim testi de sadece parametre vererek yazılabilir. Birinci biçimde ise test, global durumu kurmak ve başka modüllerin yan etkilerini dışlamak zorundadır.

Global veri yasak değildir; kesme ile görev arasında paylaşılan veri gibi durumlarda kaçınılmaz da olabilir. Ancak paylaşılan her verinin yazanı, okuyanları ve erişim koruması mimaride tanımlanmalıdır. Bileşenler arasındaki veri ve kontrol bağlaşımı tasarım verisinin parçasıdır; Seviye A, B ve C'de gereksinim tabanlı testlerin bu bağlaşımını ne ölçüde işlettiği ayrıca analiz edilir (bkz. [9. Yazılım Doğrulama](#)). Mimaride görünmeyen bir bağlaşım, o analizde de görünmez.

Bağlaşım ve iç uyumun ötesinde, emniyet-kritik tasarımda aranan nitelikler şunlardır:

- **Doğrulanabilirlik:** Her düşük seviyeli gereksinim, gözlemlenebilir bir davranışa karşılık gelmeli; "test edilemeyen tasarım" bir tasarım hatasıdır.
- **Kararlılık:** Küçük bir gereksinim değişikliği tasarımın büyük bölümünü devirmemeli; değişikliğin etkisi az sayıda modülle sınırlı kalmalıdır.
- **Değiştirilebilirlik:** Donanım farklılıkları ve konfigürasyon verileri, çekirdek mantıktan ayrı tutulmalı; taşıma işi yerelleşmelidir (bkz. [22. Konfigürasyon Verisi](#)).
- **Basitlik:** Aynı davranışı sağlayan iki tasarımdan analizi kolay olan tercih edilir; zekice ama izlenemez çözümler sertifikasyonda pahalıya mal olur.
- **Öngörülebilirlik:** En kötü durum yürütme süresi (worst-case execution time, WCET) ve bellek kullanımı tasarımdan kestirilebilmelidir; özyineleme ve dinamik bellek bu yüzden genellikle dışlanır.

Bu nitelikler ölçülebilir de kılınabilir. Pratikte kullanılan göstergeler:

Nitelik	Tipik gösterge
Bağlaşım	Modül başına dış bağımlılık sayısı, global veri erişim sayısı
İç uyum	Modülün tek sorumlulukla tarif edilebilmesi (gözden geçirme ölçütü)
Basitlik	Çevrimsel karmaşıklık (cyclomatic complexity) üst sınırı
Doğrulanabilirlik	Test edilemeyen düşük seviyeli gereksinim sayısı (hedef: sıfır)
Öngörülebilirlik	Statik olarak hesaplanabilen en derin çağrı zinciri ve yığın kullanımı

Bu eşikler proje standartlarında (tasarım ve kodlama standartları) sayısal olarak tanımlanır; gözden geçirmelerde ve statik analizde otomatik denetlenebilir hâle gelir.

Tasarımda ayrıca düşünülecekler

Bazı konular kodlamaya ya da doğrulamaya bırakıldığında pahalıya çözülür; tasarımda karara bağlanmaları gerekir. Ayrıntıları kitabın ilgili bölümlerindedir; burada tasarıma düşen pay özetlenmiştir.

- **Devre dışı bırakılmış kod:** Yazılımda yalnızca belirli konfigürasyonlarda çalışan ya da hiç çalıştırılması amaçlanmayan kod bulunacaksa, bu kodun etkin işlevleri olumsuz etkilemesini ve istem dışı etkinleşmesini önleyen mekanizma tasarımda tanımlanır; kullanılması amaçlanmayan ortamlarda kodun gerçekten devre dışı olduğu kanıtla gösterilir. Kodun kendisi de gereksinime izlenebilir tutulur ve etkin kodla aynı hedeflere göre geliştirilir (bkz. [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#)).
- **Kullanıcı tarafından değiştirilebilir yazılım:** Değiştirilebilen bileşenin değiştirilemeyen bileşeni etkilemesini önleyen koruma mimaride kurulur; koruma donanımıyla, yazılımla, değişikliği yapan araçla ya da bunların birleşimiyle sağlanabilir. Yazılımla sağlanıyorsa değiştirilemeyen bileşenin yazılım seviyesinde tasarlanır ve doğrulanır; araçla sağlanıyorsa araç, araç kalifikasyonu (tool qualification) kurallarına göre sınıflandırılır ve kalifiye edilir. Değiştirilebilen bileşenin yalnızca öngörülen yoldan değiştirilebildiği de gösterilir (bkz. [19. Kullanıcı Tarafından Değiştirilebilir Yazılım](#)).
- **Yazılım bölümlenmesi:** Farklı yazılım seviyesindeki bileşenler aynı işlemciyi paylaşacaksa bölümlenme yöntemi ve ihlali önleyen düzen mimarinin parçasıdır. Bölümlenme ya da başka bir mimari önlem bileşenlerin yazılım seviyesini etkiliyorsa bu bilgi türetilmiş gereksinim olarak sistem süreçlerine bildirilir (bkz. [21. Yazılım Bölümlenmesi](#)).
- **Çizelgeleme ve paylaşılan kaynaklar:** Görev yapısı, öncelikler, kesme kullanımı ve paylaşılan verinin korunması tasarım kararıdır; işletim sistemi kullanılıyorsa hangi hizmetlerinin kullanılacağı da burada sınırlandırılır (bkz. [20. Gerçek Zamanlı İşletim Sistemleri](#)).

Tasarımın doğrulanması

Tasarım aşamasının doğrulanması test ile değil, ağırlıklı olarak **gözden geçirme ve analiz** ile yapılır; çünkü ortada henüz çalıştırılacak kod yoktur. DO-178C bu değerlendirmeyi iki iş ürünü için ayrı ayrı bekler: mimari ve düşük seviyeli gereksinimler farklı sorulara farklı ölçütlerle tabi tutulur.

Mimari değerlendirmesinde tipik sorular şunlardır:

- Mimari, yüksek seviyeli gereksinimlerle uyumlu mu? Gereksinimlerin öngörmediği bir işlev ya da atlanmış bir işlev var mı?
- Bileşenler arası arayüzler tanımlı ve tutarlı mı? Bir tarafın gönderdiğini diğer taraf aynı biçimde mi bekliyor? Daha düşük yazılım seviyesindeki bir bileşenden veri alan bileşen, hatalı girdiye karşı kendini koruyor mu?
- Hata algılama ve güvenli duruma geçiş yolları mimaride görünüyor mu?
- Yazılım bölümlenmesi kullanılıyorsa bölümler arası koruma ihlal edilebiliyor mu?
- Mimari hedef bilgisayarla uyumlu mu? Başlatma, eşzamanlama, eşzamansız çalışma ve kesme kullanımı donanımın özellikleriyle çatışıyor mu; kaynak bütçeleri (işlemci zamanı, bellek, veri yolu) mimari düzeyde tutarlı mı?
- Mimari doğrulanabilir mi; sınırsız özyineleme gibi doğrulanamayan yapılar içeriyor mu?
- Tasarım standardına uyulmuş mu, sapmalar gerekçelendirilmiş mi?

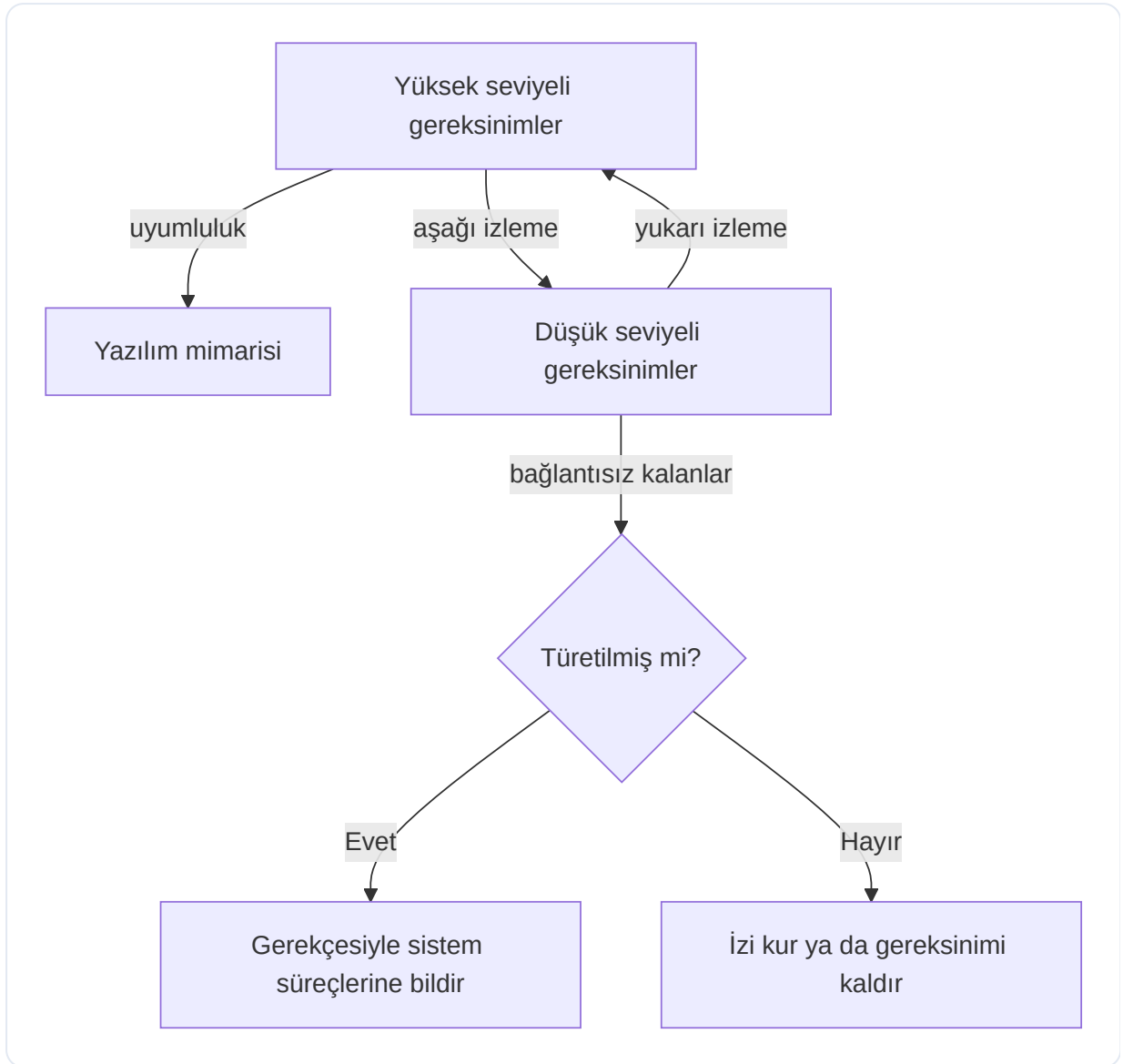
Düşük seviyeli gereksinimlerin değerlendirmesinde ise odak davranış tanımının kalitesidir:

- Her düşük seviyeli gereksinim tekil, belirsizlikten uzak ve test edilebilir mi?
- Yüksek seviyeli gereksinimlerle çelişki ya da boşluk var mı?
- Donanım/yazılım arayüzüne ve hedef bilgisayarın kısıtlarına uygun mu?
- Algoritmalar (filtreler, ölçekleme, sınır denetimleri) doğru ve kararlı mı?
- Tasarım standardındaki yazım ve ayrıntı düzeyi kurallarına uyulmuş mu?

Düşük seviyeli gereksinimlerin değerlendirmesinin omurgası **izlenebilirlik** kontrolüdür. İki yönde bakılır:

- **Aşağı yönde:** Her yüksek seviyeli gereksinim düşük seviyeli gereksinimlere iniyor mu? İnmeyen gereksinim, tasarımda unutulmuş demektir.
- **Yukarı yönde:** Her düşük seviyeli gereksinim bir yüksek seviyeli gereksinime bağlanıyor mu? Bağlanmayan gereksinim için üç yol vardır: eksik kalmış izi kurulur, **türetilmiş gereksinim** olarak gerekçesiyle işaretlenip emniyet değerlendirmesi dahil sistem süreçlerine bildirilir ya da gereksiz olduğu için kaldırılır.

Türetilmiş sayılmak için izin yokluğu şart değildir: histerezis örneğindeki gibi, üst seviyede belirtilenin ötesinde davranış tanımlayan gereksinim de türetilmiştir. Tanım ve geri bildirim yolu 6. bölümde anlatılmıştır.



Seviyeye göre beklentiler

Tasarımdan beklenen veri ve doğrulama yazılım seviyesine göre değişir:

Yazılım seviyesi	Tasarım verisi	Tasarımın doğrulanması
Seviye A	Mimari ve düşük seviyeli gereksinimler	On üç tasarım doğrulama hedefinin (objective) tamamı; altısı bağımsızlıkla (independence)
Seviye B	Mimari ve düşük seviyeli gereksinimler	On üç hedefin tamamı; bağımsızlık üç hedefte aranır
Seviye C	Mimari ve düşük seviyeli gereksinimler	Dokuz hedef; hedef bilgisayarla uyumluluk ve doğrulanabilirlik hedefleri uygulanmaz, bağımsızlık aranmaz
Seviye D	Mimari; düşük seviyeli gereksinimler için hedef yoktur	Yalnızca, bölümlere kullanılıyorsa bölümlere bütünlüğünün doğrulanması

Seviye B'de bağımsızlık üç hedefte aranır: düşük seviyeli gereksinimlerin yüksek seviyeli gereksinimlere uyumu; doğruluğu ve tutarlılığı; algoritmaların doğruluğu. Seviye A'da bunlara mimarinin yüksek seviyeli gereksinimlerle uyumluluğu, mimarinin tutarlılığı ve bölümlene bütünlüğü eklenir.

İki noktaya dikkat etmek gerekir. Seviye D'de düşük seviyeli gereksinimlerin sertifikasyon kanıtı olarak sunulması beklenmez; bu, tasarım yapılmayacağı anlamına gelmez, yalnızca o verinin DO-178C hedefleriyle doğrulanmadığı anlamına gelir. Bölümlene kullanılıyorsa mekanizmasının ayrıntısı bu seviyede de belgelenir. Bağımsızlığın aranmadığı seviye ve hedeflerde ise tasarımın yazarı dışında biri tarafından gözden geçirilmesi standart şartı değildir ama iyi uygulamadır; tasarım hatasını en ucuza yakalayan yöntem budur.

Pratikte tasarım gözden geçirmeleri kontrol listeleriyle yürütülür ve bulgular kayıt altına alınır: kim baktı, hangi sürüme baktı, hangi ölçüt geçti, hangi bulgu açık kaldı. Bu kayıtlar hem kalite güvencesinin hem de katılım aşaması (Stage of Involvement, SOI) denetimlerinin, özellikle geliştirme denetiminin ([SW SOI-2](#)) doğrudan girdisidir. Deneyimle sabittir ki tasarım aşamasında yakalanan bir tutarsızlık, kod ve test yazıldıktan sonra yakalanana göre kat kat ucuza düzeltilir; gözden geçirmeye ayrılan zaman bu yüzden bir maliyet değil, yatırımdır.

Tasarımdan kodlamaya hangi koşulla geçileceği planlardaki geçiş kriterleriyle belirlenir; örnekleri [Ek A: Örnek Geçiş Kriterleri](#) sayfasındadır. Tasarımın koda dökülmesi ise [8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#) bölümünün konusudur.

Bu bölümden akılda kalması gerekenler

- Tasarımın çıktısı tasarım tanımıdır: yazılım mimarisi ve düşük seviyeli gereksinimler. Düşük seviyeli gereksinimler yüksek seviyeli gereksinimlere izlenebilir, mimari onlarla uyumlu olmalıdır.
- Düşük seviyeli gereksinim, kodun başka bilgiye gerek kalmadan yazılabileceği kadar ayrıntılıdır ama kodun kopyası değildir; ayrıntı düzeyi tasarım standardında tanımlanır.
- Tasarımda doğan ve üst seviyede belirtilenin ötesine geçen her karar türetilmiş gereksinimdir; gerekçesiyle kaydedilir ve sistem süreçlerine bildirilir.
- Arayüz tanımı birim, aralık, güncelleme hızı ve geçerlilik bilgisini içerir; normal çalışma, kısıtlı çalışma ve güvenli durum arasındaki geçişler ile hata algılama ve tepki kuralları tasarımda yazılıdır.
- Bağlaşım düşük, iç uyum yüksek tutulur; doğrulanabilirlik, basitlik ve öngörülebilirlik proje standartlarında ölçülebilir eşiklere bağlanır.
- Tasarım, gözden geçirme ve analizle doğrulanır; beklenti yazılım seviyesine göre değişir ve Seviye D'de tasarım doğrulamasından yalnızca bölümlene bütünlüğü kalır.
- İyi tasarım doğrulamayı kolaylaştırır; tasarımda yakalanan hata en ucuz hatadır.

8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon

Bu bölüm, tasarımın kaynak koda dönüştürüldüğü kodlama sürecini ve kaynak koddan uçağa yüklenen çalıştırılabilir nesne kodunun üretildiği entegrasyon sürecini anlatır. Dil ve derleyici seçimi, kodlama standardının uygulanması, kaynak kodun doğrulanması, derleme, bağlama ve hedefe yükleme bu sırayla ele alınır.

Amaç yalnızca derlenen bir yazılım üretmek değildir; okunabilir, test edilebilir, gereksinimlerine izlenebilir ve yıllar sonra aynı biçimde yeniden üretilebilir bir gerçekleştirme ortaya koymaktır.

Kodlama sürecinin girdileri, çıktıları ve kodlama standardı

Kodlama ve entegrasyon, DO-178C'deki geliştirme süreçlerinin son ikisidir. İkisi de kendi başına karar veren süreçler değildir: ne üretileceğini önceki süreçlerin çıktıları, nasıl üretileceğini planlar ve standartlar belirler.

Süreç	Girdiler	Çıktılar
Kodlama	Düşük seviyeli gereksinimler (low-level requirements), yazılım mimarisi (software architecture), kodlama standardı, yazılım geliştirme planı (Software Development Plan, SDP)	Kaynak kod (source code); kaynak kod ile düşük seviyeli gereksinimler arasındaki iz verisi (trace data)
Entegrasyon	Kaynak kod, yazılım mimarisi	Nesne kodu (object code) ve çalıştırılabilir nesne kodu (executable object code); varsa parametre verisi ögesi (parameter data item, PDI) dosyaları; derleme, bağlama ve yükleme verisi (tipik olarak derleme ve bağlama betikleri ile seçenekleri, bağlayıcı haritası (linker map), derleme kaydı ve yükleme talimatı)

Düşük seviyeli gereksinimlerin ve mimarinin nasıl ortaya çıktığı [7. Yazılım Tasarımı](#) bölümünde anlatıldı. Tablodan iki kural çıkar. Birincisi, kod yalnızca girdilerinde yazanı gerçekleştirir: kodlayan kişi gereksinimde olmayan bir davranışa ihtiyaç duyuyorsa onu sessizce koda gömmez; eksik, tasarım sürecine bildirilir ve gerekiyorsa türetilmiş gereksinim (derived requirement) olarak kaydedilir. İkincisi, kodlama ya da entegrasyon sırasında girdilerde fark edilen her belirsizlik ve hata, kaynaklandığı sürece geri beslenir. Kodda "etrafından dolaşılan" bir gereksinim hatası gereksinimde durmaya devam eder ve o gereksinimi okuyan bir sonraki kişiyi — testi yazanı da — yanıltır.

Kod, tasarımın somut biçimidir. Kodlama süreci disiplinsiz yürürse doğru tasarım bile beklenmeyen davranış üretebilir; bu yüzden kodlama standardı (coding standard) bir stil kılavuzundan çok, davranışı öngörülebilir kılan kurallar bütünüdür. Standart planlama aşamasında yazılır; içeriğinin ayrıntısı [5. Yazılım Planlama](#) bölümündedir. Tipik konu başlıkları ve her birinin kurala bağlanma nedeni şöyle özetlenebilir:

Konu	Neden kurala bağlanır?
Dil alt kümesi ve yasaklı yapılar	Tanımsız davranışı (undefined behavior) ve derleyiciye bağlı davranışı kodun dışında tutmak
Veri tipleri ve dönüşümler	Sabit genişlikli tiplerle taşınabilirliği sağlamak; örtük dönüşümlerden doğan işaret ve taşma hatalarını önlemek
Hata dönüşleri	Her dönüş değerinin denetlenmesini, hatanın sessizce yutulmamasını sağlamak
Sabitler ve değişkenler	Anlamı belirsiz sayılar yerine adlandırılmış sabit kullanmak; global veriyi sınırlayarak bileşenler arası veri bağlaşımını (data coupling) görünür tutmak
Döngü, koşul ve karmaşıklık sınırları	Kontrol akışını gözden geçirilebilir ve test edilebilir tutmak
Adlandırma, dosya düzeni ve yorum	Gözden geçirmeyi hızlandırmak; izlenebilirlik etiketinin nereye ve nasıl yazılacağını belirlemek

Bu kuralların hedeflediği nitelikler bellidir: emniyet-kritik kod anlaşılır olmalı, kolay gözden geçirilebilmeli, yan etkilerini sınırlı tutmalı, sınır durumlarında açık davranmalı ve gereksiz karmaşıklık taşımamalıdır. Bir kod parçası çalışıyor olabilir; ancak okunamıyor, test edilemiyor ya da bakımı yapılamıyorsa sertifikasyon açısından zayıftır, çünkü doğruluğu kimseye gösterilemez.

Emniyet-kritik yazılımda diller ve derleyici seçimi

Programlama dili seçimi, projenin en kalıcı kararlarından biridir. Dil bir kez seçildiğinde kodlama standardı, statik analiz araçları, derleyici, test ortamı ve ekip eğitimi hep bu seçimin etrafında şekillenir. Aviyonikte bu seçim çoğunlukla dört dil arasında yapılır: çevirici dili, Ada, C ve C++.

Çevirici dili (assembly language), işlemciye en yakın dildir. Donanım yazmaçlarına (register) doğrudan erişim, kesin zamanlama ve başlatma (boot) kodu gibi yerlerde hâlâ vazgeçilmezdir. Ancak taşınabilirliği yoktur, okunması ve gözden geçirilmesi zordur; bu yüzden modern projelerde yalnızca donanıma dokunan dar bir katmanla sınırlandırılır.

Ada, emniyet-kritik sistemler düşünülerek tasarlanmış bir dildir. Güçlü tip denetimi, aralık kontrolü ve görev (tasking) modeli sayesinde birçok hata sınıfını daha derleme aşamasında yakalar. SPARK gibi biçimsel yöntemler (formal methods) destekli alt kümeleri, bazı özelliklerin matematiksel olarak kanıtlanmasına da imkân verir; bu kanıtların doğrulamada nasıl

kullanılabileceği [16. DO-333 ve Biçimsel Yöntemler](#) bölümünün konusudur. Dil ve araçları geliştirilmeye devam etmektedir; buna karşılık ekosistem C'ye göre dardır, az sayıda tedarikçiye bağlıdır ve deneyimli ekip bulmak daha zordur.

C, bugün aviyonikte en yaygın dildir. Derleyici ve araç desteği geniştir, gömülü donanımların hemen hepsinde olgun bir araç zinciri vardır. Ancak dilin kendisi emniyetli değildir: tanımsız davranışlar, serbest işaretçi aritmetiği ve zayıf tip denetimi ciddi tuzaklar barındırır. Bu yüzden C, hemen her projede **MISRA C** gibi bir alt küme (subset) ile birlikte kullanılır; alt küme, dilin tehlikeli özelliklerini yasaklayarak kalan kısmı öngörülebilir kılar. Kodlama standardında temel alınan kural kümesinin sürümü ve kullanılan C dil standardı açıkça yazılır; bu satırların yazıldığı tarihte güncel sürüm MISRA C:2025'tir, yürüyen projelerde MISRA C:2012 ve sonraki düzeltmeleri de yaygındır.

C++, özellikle büyük uygulama katmanlarında kullanımı artan bir dildir. Soyutlama gücü büyük kod tabanlarını yönetmeyi kolaylaştırır; ancak kalıtım, dinamik bağlama (dynamic dispatch), şablonlar, istisnalar ve dinamik bellek yönetimi kaynak kod ile nesne kodu arasındaki mesafeyi açar ve doğrulamaya yeni sorular getirir. Bu yüzden C++ da MISRA C++ gibi kısıtlayıcı bir kural kümesiyle kullanılır; nesne yönelimli özellikler devredeyse DO-332'nin ek hedefleri (objective) ve analizleri gündeme gelir (bkz. [15. DO-332 ve Nesne Yönelimli Teknoloji ve İlgili Teknikler](#)).

Ölçüt	Çevirici dili	Ada / SPARK	C (+MISRA C)	C++ (kısıtlı alt küme)
Donanım denetimi	Çok yüksek	Orta	Yüksek	Yüksek
Tip güvenliği	Yok	Çok güçlü	Zayıf (alt kümeyle iyileşir)	Orta (alt kümeyle iyileşir)
Araç ve derleyici ekosistemi	Dar	Dar ama sürdürülüyor	Çok geniş	Geniş
Ekip bulma kolaylığı	Zor	Zor	Kolay	Orta
Statik analiz olgunluğu	Sınırlı	İyi	Çok iyi	İyi
Doğrulamada dikkat isteyen yan	Gözden geçirme maliyeti, taşınamazlık	Çalışma zamanı denetimleri ve kütüphanesi	Tanımsız davranışlar	Nesne yönelimli özelliklerin ek analizleri

Rust gibi bellek güvenliğini derleme zamanında denetleyen yeni diller de gündemdedir. Emniyet standartlarına göre kalifiye edilmiş Rust derleyicileri önce otomotiv ve endüstriyel alanlarda ortaya çıkmıştır; havacılıkta araç zinciri, yerleşik kodlama kuralları ve sertifikasyon deneyimi bu satırların yazıldığı tarihte henüz sınırlıdır. Böyle bir dili seçen proje, bunu planlama aşamasında otoriteyle konuşulacak bir risk olarak ele almalıdır.

Dil kadar **derleyici seçimi** de önemlidir; çünkü yazılım açısından uyumun gösterildiği nihai ürün kaynak kod değil, uçakta çalışan çalıştırılabilir nesne kodudur. Derleyici değerlendirilirken şu ölçütlere bakılır:

- **Belirlenimcilik (determinism):** aynı kaynak ve aynı seçeneklerle her derlemede aynı çıktının üretilmesi; agresif eniyileme (optimization) seviyelerinden kaçınılması.
- **Hedef işlemci desteği:** kullanılan işlemci ve çalışma ortamı için kanıtlanmış, hatası bilinen ve belgelenmiş bir sürümün bulunması.
- **Hata geçmişi:** derleyici üreticisinin bilinen hata listesi yayımlaması ve projenin bu listeyi izleyip etkilenen yapıları yasaklayabilmesi.
- **Ekip deneyimi:** dil ve derleyiciyle daha önce sertifikasyon geçirmiş mühendislerin varlığı; deneyimsiz ekip, en iyi araçla bile riskli sonuç üretir.
- **Doğrulama stratejisiyle uyum:** nesne kodu ile kaynak kod arasındaki ilişkinin izlenebilir olması. Derleyiciler sıklıkla kaynak satırlarına doğrudan izlenemeyen kod üretir (sınır denetimleri, örtük başlatma, yardımcı işlev çağrıları). Seviye A'da bu kodun belirlenmesi ve doğruluğunun ek doğrulamayla gösterilmesi beklenir; yani aranan, böyle bir kodun yokluğunu kanıtlamak değil, var olanı bulup doğrulamaktır. Kodun hangi yöntemle saptanacağı planlama aşamasında belirlenir ve ilgili plana yazılır. Bu işi kolaylaştıran ya da zorlaştıran, seçilen derleyici ve seçenekleridir (bkz. [9. Yazılım Doğrulama](#) ve [SW SOI-3](#)).

Derleyici çoğu projede kalifiye edilmez: ürettiği nesne kodu gereksinim tabanlı testlerle hedef ortamda zaten doğrulandığı için araç kalifikasyonu (tool qualification) koşulu oluşmaz (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)). DO-178C de derleyiciyi, yazılım ürününün doğrulaması başarıyla tamamlandığında kabul edilebilir sayar; ama yalnızca o ürün için. Derleyicinin, bağlayıcının ve yükleyicinin doğruluğuna ilişkin varsayımlar da yazılım doğrulama planında (Software Verification Plan, SVP) yazılır. Bu yaklaşımın bedeli şudur: doğrulama, o derleyicinin o seçeneklerle ürettiği koda bağlıdır. Bu nedenle derleyici sürümü ve derleme seçenekleri projenin ömrü boyunca **dondurulur** ve konfigürasyon yönetimi (configuration management) altında tutulur. Proje ortasında "yeni derleyici sürümüne geçelim" kararı masum görünse de, üretilen nesne kodunu değiştirdiği için yapılmış doğrulamanın önemli bir kısmını geçersiz kılabilir.

Kodlamaya ilişkin özel konular

Kodlama standardının uygulanması

Kodlama standardı, planlama aşamasında yazılır ama değerini kodlama aşamasında gösterir. Standardın raflarda kalmaması için iki şey gerekir: kuralların **otomatik denetlenebilir** olması ve istisnaların **kayıt altına alınması**. Bir kural statik analiz aracıyla denetlenemiyorsa gözden geçirme kontrol listesine (checklist) girer; haklı bir gerekçeyle ihlal edilmesi gerekiyorsa sapma (deviation) kaydı açılır ve gerekçesiyle birlikte onaylanır. Sessizce ihlal edilen kural, standardın tamamına olan güveni zedeler.

Tipik bir standardın davranışa dokunan kuralları şöyle örneklenebilir:

```
#include <stdint.h>

/* Kurallar: tek çıkış noktası; if ... else if zinciri else ile kapanır;
   sabit genişlikli tipler kullanılır.
   Ön koşul: alt <= ust; çağırın taraf sağlar (LLR-31-4). */
int32_t hiz_sinirla(int32_t hiz, int32_t alt, int32_t ust)
{
    int32_t sonuc;

    if (hiz < alt)
    {
        sonuc = alt;
    }
    else if (hiz > ust)
    {
        sonuc = ust;
    }
    else
    {
        sonuc = hiz;
    }

    return sonuc;
}
```

Örnekteki kurallar MISRA C:2012 numaralandırmasıyla Kural 15.5 (tek çıkış noktası), Kural 15.7 (if ... else if zincirinin else ile kapanması) ve Yönerge 4.6'ya (boyutu ve işareti belli tipler) karşılık gelir. Bunlar stil tercihi değildir; gözden geçirmeyi kolaylaştırır ve yapısal kapsam analizinde (structural coverage analysis) belirsizlikleri azaltır. MISRA kuralları bağlayıcılığına göre sınıflandırılır: sapma kabul etmeyenler (mandatory), ancak kayıtlı ve gerekçeli sapmayla ihlal edilebilenler (required) ve tavsiye niteliğindeki (advisory). Tek çıkış kuralı son gruptandır; tavsiye kurallarından hangilerinin projede bağlayıcı olduğunu kodlama standardı söyler.

İki ayrıntı daha gözden geçirenin işini kolaylaştırır. `sonuc` her dalda atandığı için tanımda ayrıca başlatılmamıştır; fazladan bir ilk atama, statik analizin "kullanılmayan atama" diye işaretleyeceği bir bulgu olurdu. `alt <= ust` ön koşulu ise kodlama kuralının değil tasarımın konusudur: ya çağırın tarafın bunu sağladığı gereksinimde yazılır ya da işlev kendisi denetler. Yorumdaki tek satır, hangi seçimin yapıldığını ve dayanağını gösterir.

Derleyici kütüphaneleri

Kod yalnızca sizin yazdığınız satırlardan oluşmaz. Derleyici, bölme, kayan nokta işlemleri veya `memcpy` benzeri işlevler için kendi **çalışma zamanı kütüphanelerinden** (runtime library) kod ekler. Bu kod da uçakta çalışır; dolayısıyla yazılımın geri kalanıyla aynı seviyenin hedeflerine tabidir. Pratikte üç yaklaşım görülür:

- kullanılan kütüphane işlevlerini belirleyip her birini gereksinimlendirerek test etmek,

- sertifikasyona hazır (önceden doğrulanmış) bir kütüphane paketi tedarik etmek,
- kütüphane kullanımını tamamen yasaklayıp gereken işlevleri projede yazmak.

Hangi yol seçilirse seçilsin, bağlanan her nesnenin kaynağı ve doğrulama durumu hesap verilebilir olmalıdır; "derleyici ekledi, bizden sorulmaz" diyebileceğiniz bir kod parçası yoktur. Bağlandığı hâlde hiç çağrılmayan kütüphane işlevleri de bu hesabın içindedir. DO-178C kullanılmayan kütüphane işlevlerini devre dışı bırakılmış kod (deactivated code) örnekleri arasında sayar; ancak bu statü kendiliğinden doğmaz: sınıfı belirleyen, işlevlerin bir gereksinime izlenmesi ve tasarım gereği çalıştırılmamasıdır. Bu sınıfa giren kod ayrıca planlarda beyan edilir ve istem dışı çalışmasının önleniği gösterilir. Hiçbir gereksinime izlenemeyen işlev ise gereksiz koddur (extraneous code) ve beklenen çözüm kaldırılmasıdır (bkz. [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#)).

Otomatik kod üreticileri

Model tabanlı geliştirme (model-based development) ortamlarında kaynak kodun bir kısmı otomatik kod üretici (automatic code generator) tarafından üretilir. Bu, kodlama hatalarını azaltabilir ama doğrulama yükünü ortadan kaldırmaz; yalnızca yerini değiştirir. İki temel strateji vardır:

- **Üretilen kodu elle yazılmış kod gibi ele almak:** üretilen kod gözden geçirilir, statik analize sokulur ve gereksinimlere karşı test edilir. Araç için ek bir kanıt gerekmez, ama üretilen kodun okunabilirliği gözden geçirme yükünü artırabilir.
- **Araç kalifiye etmek:** araç kalifikasyonu ile kod üreticinin çıktısına güvenilebileceği gösterilir ve bazı doğrulama adımları azaltılır. Bu, aracın kendisinin ciddi bir kanıt paketiyle desteklenmesini gerektirir: çıktısı uçuş yazılımının parçası olduğu için kod üretici Ölçüt 1'e girer; araç kalifikasyon seviyesi (tool qualification level, TQL) yazılım seviyesine göre Seviye A'daki TQL-1 ile Seviye D'deki TQL-4 arasında belirlenir. Her yazılım seviyesinde en sıkı kalifikasyonu bu ölçüt getirir.

Hangi strateji seçilirse seçilsin, kod üreticinin kullanımı planlarda tanımlanan seçeneklere ve kısıtlara uymak zorundadır.

Modelin gereksinim mi tasarım mı sayıldığı ve simülasyondan ne kadar kredi alınabileceği [14. DO-331 ve Model Tabanlı Geliştirme ve Doğrulama](#) bölümünde, kalifikasyonun kendisi 13. bölümde anlatılıyor.

Karma projelerde (üretilen kod ve elle yazılmış kod birlikte) en sık yapılan hata, ikisinin arayüzünü belirsiz bırakmaktır. Hangi dosyalara elle dokunulmasının yasak olduğu, üretim adımının derleme sürecinin neresinde çalıştığı ve model sürümü ile kod sürümü arasındaki izlenebilirlik (traceability) baştan tanımlanmalıdır.

Kaynak kodun doğrulanması

Kaynak kodun doğrulanması testle başlamaz; testten önce kodun **gözden geçirilmesi** ve **analiz edilmesi** gelir. Test, kodun ne yaptığını gösterir; gözden geçirme ve analiz ise kodun neden öyle yazıldığını sorgular. İkisi birbirinin yerine geçmez. Yöntemlerin ayrıntısı [9. Yazılım Doğrulama](#) bölümündedir; burada koda özgü sorular ele alınır.

Kod gözden geçirmesinde (code review) yanıtlanması beklenen temel sorular şunlardır:

- Kod, izlendiği **düşük seviyeli gereksinimleri** doğru ve eksiksiz gerçekleştiriyor mu? Gereksinimde olmayan bir davranış eklenmiş mi?
- Kod, **yazılım mimarisine** uyuyor mu; tanımlı arayüzlerin dışına çıkan bir erişim var mı?
- Kod **doğrulanabilir** mi; test edilemeyen ya da sonucu gözlemlenemeyen yapılar içeriyor mu, test edebilmek için kodu değiştirmek gerekiyor mu?
- **Kodlama standardına** uyuluyor mu; sapmalar kayıtlı ve gerekçeli mi?
- Kod **izlenebilir** mi; her düşük seviyeli gereksinim kodda karşılığını bulmuş mu, her fonksiyon en az bir gereksinime bağlanabiliyor mu? Bağlanamayan kod, ölü kodu (dead code) da içeren gereksiz kod şüphesidir ve açıklanmak zorundadır; ayrımları 17. bölümde yapılıyor.
- Kod **doğru ve tutarlı** mı: taşma, sıfıra bölme, başlatılmamış değişken, yığın (stack) kullanımı, en kötü durum yürütme süresi (worst-case execution time, WCET) gibi konular ele alınmış mı?

Gözden geçirmenin işe yaraması için birkaç pratik koşul vardır: gözden geçirme bir **kontrol listesine** dayanmalı, belirli bir kod sürümü üzerinde yapılmalı, bulgular kayıt altına alınıp kapanışları izlenmelidir. "Omuz üstünden bakıp onayladım" tarzı gözden geçirmeler kanıt üretmez; denetimde de savunulamaz. Gözden geçirmeyi kodu yazan kişi dışında birinin yapması da bu koşullardandır, ancak standart açısından statüsü seviyeye bağlıdır.

SEVİYEYE GÖRE BEKLENTİ

Kaynak koda ilişkin doğrulama hedeflerinin kapsamı yazılım seviyesine (software level) göre değişir. Seviye A'da üç hedef — kodun düşük seviyeli gereksinimlere uyumu, mimariye uyumu ve doğruluk ile tutarlılığı — bağımsızlıkla (independence) karşılanır; yani bu hedeflerde gözden geçiren, kodun yazarı olamaz. Seviye B'de bağımsızlık yalnızca düşük seviyeli gereksinimlere uyumda aranır. Seviye C'de doğrulanabilirlik dışındaki hedefler geçerliliğini korur, ancak bağımsızlık aranmaz. Seviye D'de kaynak kodun gözden geçirilmesine ve analizine ilişkin hedef yoktur. Standart yazar dışı gözden geçirmeyi yalnızca belirli seviye ve hedeflerde şart koşar; yine de her seviyede iyi uygulamadır.

Statik analiz, insan gözünün sistematik olarak kaçırdığı hata sınıflarını yakalar ve gözden geçirmeyi tamamlar:

Analiz türü	Yakaladığı tipik sorunlar
Kodlama standardı denetimi	MISRA ihlalleri, yasaklı yapılar
Veri akışı analizi	Başlatılmamış değişken, kullanılmayan atama
Kontrol akışı analizi	Erişilemeyen kod, sonsuz döngü riski
Değer aralığı analizi	Taşma, dizi sınırı aşımı, sıfıra bölme
Kaynak analizi	Yığın kullanımı, özyineleme, bellek sınırları

Statik analiz aracının çıktısı da tek başına kanıt değildir: her bulgu bir mühendis tarafından değerlendirilir; gerçek hata ise düzeltilir, yanlış alarm (false positive) ise gerekçesiyle kapatılır. Doğrulama faaliyetinin yerine geçen bir araç kullanılıyorsa (örneğin bir kuralın denetimini tamamen araca bırakmak ve çıktısını ayrıca doğrulamamak), aracın kalifikasyonu gündeme gelir.

Kaynak kod doğrulaması, sonuç olarak üç şeyi görünür kılmalıdır: kodun gereksinimlere uyumu, standarda uygunluğu ve doğruluk/tutarlılık analizlerinin yapıldığı. Bu üç kanıt tamamlanmadan koda "doğrulandı" damgası vurulmaz; test bu kanıtların üzerine inşa edilir.

Entegrasyon: derleme, bağlama ve yükleme

Gündelik dilde entegrasyon, "parçaları birleştirip denemek" anlamına gelir. DO-178C'de ise entegrasyon süreci bir üretim sürecidir: kaynak koddan çalıştırılabilir nesne kodunu üretir ve onu hedef bilgisayara yükler. Parçaların birlikte doğru davrandığını göstermek, entegrasyon testlerinin, yani doğrulamanın işidir. İkisi iç içe yürür ama kanıtları ayrıdır: biri "doğru ürün, doğru biçimde üretildi mi", diğeri "üretilen ürün gereksinimlerini karşılıyor mu" sorusunu yanıtlar.

Yazılım entegrasyonu ve donanım/yazılım entegrasyonu

Süreç iki adımdan oluşur. **Yazılım entegrasyonu (software integration)**, kaynak dosyaların derlenmesini, nesne dosyalarının bağlanmasını (link) ve kod ile verinin bellek bölgelerine yerleştirilmesini kapsar; çıktısı çalıştırılabilir nesne kodudur. **Donanım/yazılım entegrasyonu (hardware/software integration)**, bu imajın (image) hedef bilgisayara yüklenmesidir; yazılım ilk kez gerçek işlemci, gerçek bellek ve gerçek giriş/çıkış ile karşılaşır.

Parametre verisi ögesi kullanan projelerde PDI dosyaları da bu süreçte üretilir. Her dosya ayrı bir konfigürasyon ögesidir ve birlikte çalışacağı çalıştırılabilir nesne koduyla uyumlu olmalıdır; koşulları ve doğrulaması [22. Konfigürasyon Verisi](#) bölümünde anlatılıyor.

Entegrasyon stratejisi ve sırası

Bütün bileşenleri en sonda tek seferde birleştirmek, hata çıktığında nereye bakılacağını belirsiz bırakır. Bunun yerine **artımlı entegrasyon (incremental integration)** uygulanır: her adımda az sayıda bileşen eklenir ve eklenen parça, altındaki çalışan temelin üzerinde denenir. Gömülü bir aviyonik birimde tipik sıra aşağıdan yukarıdır: önce başlatma kodu ve donanım dokunan sürücüler, sonra işletim sistemi ve çizelgeleme, ardından haberleşme ve giriş/çıkış katmanı, en sonda uygulama işlevleri. Sıra ve her artımın içeriği önceden yazılır; her artım, konfigürasyonu belli bir derleme sürümüdür. Henüz hazır olmayan bileşenlerin yerini tutan yer tutucu kod (stub) ve test sürücülerini uçuş koduyla aynı yerde tutulmaz ve nihai imaja girmedeği gösterilir.

Entegrasyonun ortaya çıkardığı uyumsuzluklar, tek tek bileşenlere bakılarak görülemeyen türdendir:

Konu	Entegrasyonda görülen tipik uyumsuzluk
Arayüz uyumu	Birim, ölçek, bayt sırası ya da parametre sırası farkı
Veri sırası ve zamanlama	Bir bileşenin, diğerinin o çerçevede henüz güncellemediği veriyi okuması
Başlatma sırası	Henüz başlatılmamış sürücünün ya da verinin kullanılması
Hata yayılımı	Alt katmanın bildirdiği hata kodunun üst katmanda yok sayılması
Sürüm uyumu	Farklı temel çizgilerden (baseline) alınmış bileşenlerin aynı imajda buluşması

Bu tür bir bulgunun düzeltilmesi çoğu kez kodda değil, arayüz tanımında ya da gereksinimdedir. Entegrasyon masasında koda hızlıca eklenen bir dönüşüm sorunu o gün çözer, ama tasarım verisi yanlış kalır; doğru yol, bulguyu kaynaklandığı sürece geri beslemektir.

Derleme sürecinin tekrarlanabilirliği

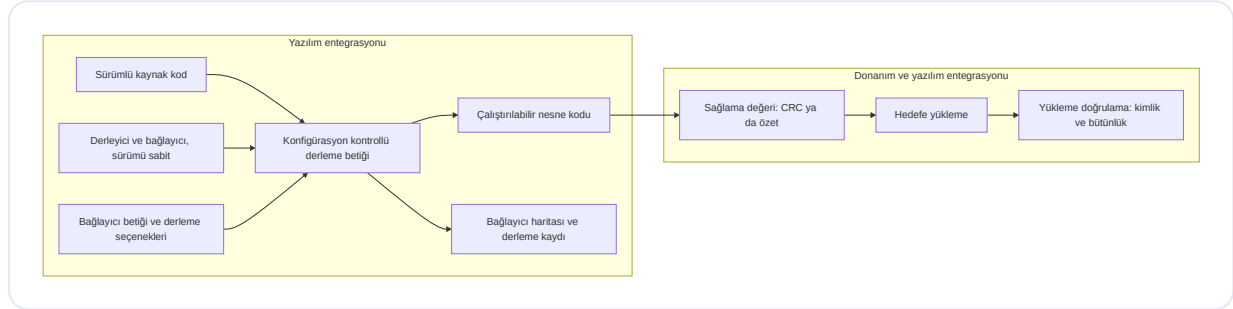
Uçakta çalışan nesne kodu olduğuna göre, kaynak kodu nesne koduna çeviren derleme (build) süreci de mühendislik ürünüdür ve doğrulamanın parçasıdır. Temel beklenti tek cümledir: **aynı kaynaktan, aynı araçlarla, her zaman aynı ikili (binary) üretilebilmelidir**. İkili birebir aynı çıkmıyorsa — örneğin imaja gömülen bir zaman damgası yüzünden — farkın kaynağı bilinmeli, açıklanmalı ve tercihen giderilmelidir.

Bunu sağlamak için derleme sürecine giren her şey konfigürasyon yönetimi altında tutulur:

- kaynak dosyaların sürümleri,
- derleme betikleri (makefile vb.) ve bağlayıcı (linker) betikleri,
- derleyici, bağlayıcı ve yardımcı araçların tam sürümleri,
- derleme seçenekleri ve tanımlı makrolar,
- derleme ortamının kendisi (işletim sistemi, kurulum paketleri).

Bu bilgilerin resmî adresi iki yaşam döngüsü verisidir: ortamı yazılım yaşam döngüsü ortam konfigürasyon indeksi (Software Life Cycle Environment Configuration Index, SECI), üretilen sürümün bileşenlerini ve yeniden derleme talimatını yazılım konfigürasyon indeksi (Software Configuration Index, SCI) kayıt altına alır (bkz. [10. Yazılım Konfigürasyon Yönetimi](#)).

Derleme seçenekleri özellikle kritiktir: bir eniyileme bayrağının değişmesi, test edilmiş nesne kodundan farklı bir nesne kodu üretir. Bu nedenle seçenekler planlarda belgelenir, betiğe gömülür ve elle geçersiz kılınamaz hâle getirilir. Mühendisin kendi makinesinde "elle derleyip" ürettiği ikili, ne kadar doğru görünürse görünsün, resmî bir konfigürasyon değildir.



Resmî derlemeler tipik olarak temiz bir ortamda, sıfırdan (full build) alınır ve bir **derleme kaydı** üretilir: hangi sürümlerden, hangi araçlarla, hangi seçeneklerle üretildiği bu kayıtta yer alır. Tekrarlanabilirlik, sertifikasyondan yıllar sonra bir hata düzeltilmesi gerektiğinde aynı ortamı yeniden kurabilmek için de gereklidir; bu yüzden derleme ortamının arşivlenmesi (hatta sanal makine olarak saklanması) yaygın bir uygulamadır.

Entegrasyon çıktılarının gözden geçirilmesi

Derlemenin hatasız bitmesi, çıktının doğru olduğunu göstermez. Entegrasyon çıktıları da gözden geçirilir ve analiz edilir; aranan hatalar yanlış donanım adresi, çakışan ya da taşan bellek bölgeleri ve eksik ya da fazladan bileşendir. Başlıca kaynak, derleyici ve bağlayıcı uyarıları ile bağlayıcı haritasıdır. Çıktıların eksiksiz ve doğru olduğunu gösterme hedefi Seviye A, B ve C'de geçerlidir ve hiçbirinde bağımsızlık aranmaz; Seviye D'de uygulanmaz. Sadeleştirilmiş bir bellek özeti şöyle görünebilir:

Bölge	Başlangıç	Bitiş	Uzunluk	Kullanılan
ONYUKLEME	0x08000000	0x08007FFF	0x008000	0x0061C0
UYGULAMA	0x08008000	0x08077FFF	0x070000	0x04A2C0
PARAMETRE	0x08078000	0x0807FFFF	0x008000	0x001200
RAM_VERI	0x20000000	0x2000DFFF	0x00E000	0x006F40
RAM_YIGIN	0x2000E000	0x2000FFFF	0x002000	(analizle)

Böyle bir çıktının karşısında sorulan sorular şunlardır:

- Bölgelerin adresleri, hedef donanımın gerçek adres haritasıyla ve tasarımda tanımlı yerleşimle örtüşüyor mu?

- Bölgeler birbirine giriyor mu; kullanılan miktar bölgenin uzunluğunu aşıyor mu; planlarda öngörülen bellek payı korunuyor mu?
- Haritada beklenmeyen bir nesne dosyası ya da kütüphane işlevi var mı; beklenen bir bileşen eksik mi; çözülmemiş sembol kalmış mı?
- Yığın bölgesi, yığın kullanımı analizinin en kötü durum sonucunu karşılıyor mu?
- Derleyici ve bağlayıcı uyarıları sıfırlanmış ya da her biri gerekçesiyle kapatılmış mı?

Bu soruların yöntem tarafı, yani bellek haritası analizi ile bağlantı ve yükleme analizi, 9. bölümde anlatılıyor. Analiz belirli bir derleme sürümü üzerinde yapılır; sürüm değiştiğinde yinelenir ya da değişikliğin sonucu etkilemediği gerekçelendirilir.

Yamalar

Yama (patch), kaynak kodu değiştirip yeniden derlemek yerine çalıştırılabilir nesne kodunun doğrudan değiştirilmesidir; planlı derleme ve bağlama adımlarından en az biri atlanır. İmaja gömülen parça numarası, sürüm kimliği ya da sağlama değeri gibi kimlik bilgileri bu tanımın dışındadır. Yama, emniyet-kritik yazılımda kural değil, dar bir istisnadır; nedeni kanıt zinciridir. Gözden geçirme, izlenebilirlik ve kapsam analizi kaynak kod üzerinden yapılmıştır; yamalanmış ikili ise artık arşivlenen kaynaktan yeniden üretilemez. Bu yüzden gereksinim ya da mimari değişiklikleri ve doğrulamada bulunan hatalar yamayla kapatılmaz; kaynakta düzeltilir, yeniden derlenir ve etkilenen doğrulama yinelenir.

Yamanın savunulabildiği durum, geliştirme ortamından kaynaklanan bilinen bir kusurun — örneğin belgelenmiş bir derleyici hatasının — başka türlü aşılabilmesidir. O zaman da üç şey gösterilir: yamanın konfigürasyon yönetimi altında tanımlandığı ve izlendiği, yamalı yazılımın uygulanabilir bütün hedefleri hâlâ karşıladığını gösteren analiz ve yazılım başarı özetinde (Software Accomplishment Summary, SAS) yamanın gerekçesi.

Hedefe yükleme

Yükleme süreci de aynı disiplinle ele alınır. Nesne kodunun hedef donanıma aktarılması sırasında iki soru yanıtlanmalıdır:

- **Doğru yazılım mı yüklendi?** Parça numarası ve sürüm kimliği, yüklenen imaj ile belgelenen konfigürasyon arasında birebir eşleşmelidir.
- **Eksiksiz ve bozulmadan mı yüklendi?** Sağlama toplamı (checksum) veya döngüsel artıklık denetimi (cyclic redundancy check, CRC) gibi bütünlük kontrolleri, aktarım sırasında bozulma olmadığını göstermelidir.

Sahada yüklenebilir yazılım (field-loadable software) söz konusuysa yükleme mekanizması — hedefteki yükleme yazılımı ve bütünlük kontrolleri — ile yükleme prosedürü de doğrulanır; yarım kalan yüklemede sistemin güvenli bir durumda kalması (örneğin önceki geçerli imaja dönmesi veya kendini geçersiz sayması) ayrıca gösterilir. Bütünlük güvencesi hedefteki bu kontrollere

değil yer tarafındaki yükleyiciye dayanıyorsa, yükleyicinin kendisinin güvenceye alınması (araç kalifikasyonu ya da DO-178C'ye göre geliştirme) gündeme gelir; ayrıntısı [18. Sahada Yüklenebilir Yazılım](#) bölümündedir.

Gereksinimden koda: bir gözden geçirme örneği

Tasarımdan gelen düşük seviyeli gereksinim şöyle olsun:

LLR-73-2: Ortalama sıcaklık, örnek toplamının geçerli örnek sayısına bölümüdür. Geçerli örnek sayısı sıfırda son geçerli ortalama korunur ve sonuç geçersiz olarak işaretlenir.

Kodun gözden geçirmeye gelen ilk sürümü:

```
/* İz: LLR-73-2 */
int32_t ortalama_sicaklik(int32_t toplam, int32_t ornek_sayisi)
{
    return toplam / ornek_sayisi;
}
```

Kod derlenir ve olağan girdilerle doğru sonuç verir. Kontrol listesiyle yapılan gözden geçirme ise üç bulgu üretir:

1. **Gereksinim eksik gerçekleştirilmiş.** Örnek sayısının sıfır olduğu durum kodda yoktur; geçerlilik bilgisi üretilmemektedir.
2. **Tanımsız davranış.** C'de tamsayının sıfıra bölünmesi tanımsız davranıştır; sonuç derleyiciye, seçeneklere ve işlemciye göre değişebilir. Örnek sayısının işaretli tipte olması da anlamsız negatif değerlere kapı açar.
3. **Gereksinim belirsiz.** Bölmenin nasıl yuvarlanacağı yazılmamıştır. Bu, kodun değil gereksinimin eksikliğidir.

İlk iki bulgu kodda kapatılır. Üçüncüsü tasarım sürecine geri beslenir: gereksinime "sonuç sıfırda doğru kesilir" ifadesi eklenir ve kod, güncellenen gereksinime göre yeniden gözden geçirilir.

```
#include <stdbool.h>
#include <stdint.h>

typedef struct
{
    int32_t deger;    /* 0,1 °C çözünürlük */
    bool gecerli;
} sicaklik_t;

/* İz: LLR-73-2 */
sicaklik_t ortalama_sicaklik(int32_t toplam, uint16_t ornek_sayisi,
                             sicaklik_t onceki)
{
    sicaklik_t sonuc;

    if (ornek_sayisi > 0U)
    {
        sonuc.deger = toplam / (int32_t)ornek_sayisi;
        sonuc.gecerli = true;
    }
    else
    {
        sonuc.deger = onceki.deger;    /* son geçerli değer korunur */
        sonuc.gecerli = false;
    }

    return sonuc;
}
```

Düzeltilmiş sürümde gereksinimin iki cümlesi kodun iki dalında bulunabilir, iz etiketi ikisini birbirine bağlar ve gürbüzlük (robustness) testi sıfır örnek için beklenen sonucu tahmin etmek yerine gereksinimden okur. Bulgular, kaynakları ve kapanışları gözden geçirme kaydında kalır; geliştirme denetiminde ([SW SOI-2](#)) örnekleme yapan denetçi tam da bu zinciri izler: gereksinim, kod, kayıt, kapanış.

Gerçekleştirmede tipik riskler

Risk	Belirtisi	Önlem
Tasarımdan sapma	Kodda gereksinime izlenemeyen işlev; mimaride tanımlı olmayan çağrı ya da veri erişimi	İz verisini kodla birlikte güncellemek; gözden geçirmede mimari uyumu ayrı madde olarak sormak
Arayüzü yanlış yorumlama	Bileşen testleri geçer, entegrasyon testinde birim, ölçek ya da sıra uyumsuzluğu çıkar	Arayüzü tip, birim ve aralıkla tanımlamak; erken ve artımlı entegrasyon
Hata durumunu eksik ele alma	Denetlenmeyen dönüş değerleri; gürbüzlük testlerinde beklenmeyen davranış	Dönüş değeri denetimini kural yapıp statik analizle denetlemek; hata davranışını gereksinimde tanımlamak

Risk	Belirtisi	Önlem
Tanımsız davranışa dayanan kod	Eniyileme seviyesi ya da derleyici değişince davranışın değişmesi	Dil alt kümesi, değer aralığı analizi, dondurulmuş derleyici ve seçenekler
Belgelenmemiş entegrasyon ve elle derleme	Aynı kaynaktan farklı ikili; yalnız bir mühendisin makinesinde üretilebilen sürüm	Betikle derleme, derleme kaydı, SECI ve SCI ile kayıt altına alınmış ortam
Gereksiz karmaşık kod	Aşılan karmaşıklık sınırı; uzayan gözden geçirmeler; yapısal kapsamı tamamlamanın zorlaşması	Karmaşıklık sınırını otomatik denetlemek; işlevi bölmek, gerekiyorsa tasarıma dönmek

Bu risklerin ortak yanı, geç fark edildiklerinde bedellerinin doğrulama aşamasında ödenmesidir: aynı hata gözden geçirmede bir satırlık bulgu, entegrasyon testinde günler süren bir hata ayıklamadır.

Bu bölümden akılda kalması gerekenler

- Kodlama, tasarımın kontrollü ifadesidir: kod yalnızca düşük seviyeli gereksinimleri ve mimariyi gerçekleştirir; girdilerde bulunan eksik, kaynaklandığı sürece geri beslenir.
- Dil ve derleyici seçimi kalıcıdır; C ve C++ kısıtlayıcı bir alt kümeyle kullanılır, derleyici sürümü ve seçenekleri dondurulur. Seviye A'da kaynak koda izlenemeyen nesne kodu belirlenir ve ayrıca doğrulanır.
- Derleyici kütüphaneleri ve otomatik üretilen kod da uçakta çalışır; ikisi de doğrulama kapsamının dışında bırakılamaz.
- Kaynak kod doğrulaması testten önce gelir: gözden geçirme ve statik analiz, gereksinimlere uyumu, standarda uygunluğu ve doğruluk/tutarlılığı gösterir. Bağımsızlık seviyeye ve hedefe göre aranır; yazar dışı gözden geçirme her seviyede iyi uygulamadır.
- Entegrasyon süreci çalıştırılabilir nesne kodunu üretir ve hedefe yükler; artımlı yürütülür, bağlayıcı haritası gibi çıktıları da gözden geçirilir.
- Derleme tekrarlanabilir ve konfigürasyon kontrollü olmalıdır; yama dar bir istisnadır ve gerekçesi SAS'ta yer alır.
- Yükleme, kimlik ve bütünlük kontrolleriyle doğrulanır; güvencenin hedefteki kontrollere mi yükleyiciye mi dayandığı, neyin doğrulanacağını belirler.

9. Yazılım Doğrulama

Doğrulama (verification), yazılımın gereksinimlerini doğru ve eksiksiz karşıladığını gösteren kanıtı üretir; bu kanıt gözden geçirme, analiz ve testle oluşur. Bu bölüm bağımsızlığı, bu üç yöntemi, gereksinim ve yapısal kapsam analizlerini ve bulguların problem raporlarıyla yönetimini anlatır.

Kritik nokta, doğrulamanın sonradan eklenen bir adım olmamasıdır. Doğrulama, geliştirmeye eşzamanlı yürüyen bütünüleyici süreçlerden (integral processes) biridir; tasarım ve gereksinimlerdeki belirsizlikler çoğu zaman doğrulama sırasında görünür hâle gelir.

Doğrulamanın amacı

Doğrulama, "çalışıyor mu?" sorusundan daha geniştir. Asıl soru, yazılımın tanımlanan koşullarda, beklenen sınırlar içinde ve istenen emniyet düzeyinde davranıp davranmadığıdır.

Bu yüzden doğrulama sadece hata bulma faaliyeti değildir; aynı zamanda gereksinimlerin, tasarımın ve kodun birbirini gerçekten desteklediğini gösterme faaliyetidir.

Bağımsızlık

Doğrulama bağımsızlığı (verification independence), bir iş ürününü doğrulayan kişinin o ürünü geliştiren kişiden farklı olması demektir. Buradaki amaç basit bir psikolojik gerçeğe dayanır: kendi yazdığımız gereksinimi veya kodu okurken, yazarken yaptığımız varsayımları farkında olmadan tekrar yaparız. Aynı kör noktaya iki kez bakan göz, oradaki hatayı iki kez kaçırır. Bağımsız bir göz ise ürünü kendi varsayımları olmadan, yalnızca yazılı olana bakarak değerlendirir.

Bağımsızlıkla ilgili en yaygın yanlış anlama, bunun **örgütsel** bir ayrım gerektirdiği düşüncesidir. DO-178C bağlamında bağımsızlık **kişi düzeyindedir**: doğrulamayı yapan kişinin ayrı bir bölümde, ayrı bir şirkette veya ayrı bir binada olması gerekmez. Aynı takımdaki iki mühendis, birbirinin ürünlerini çapraz doğrulayarak bağımsızlığı sağlayabilir. Önemli olan, doğrulayanın doğruladığı ürünün üreticisi olmamasıdır. Standart bunun bir sonucunu ayrıca vurgular: düşük seviyeli gereksinimlere (low-level requirements) dayalı test durumlarını yazan kişi, aynı gereksinimlerden kaynak kodu geliştiren kişi olmamalıdır. Kalite güvencesinde (quality assurance) bağımsızlık buna ek olarak düzeltici faaliyetin (corrective action) yapılmasını sağlama yetkisini de kapsar; uygulamada bu çoğunlukla ayrı bir raporlama hattıyla sağlanır. Ayrıntısı [11. Yazılım Kalite Güvencesi](#) bölümündedir.

Bağımsızlığın ikinci yolu **araç desteğidir**: insan eliyle yapılacak doğrulama faaliyetine denk bir güvence sağlayan araç da "bağımsız göz" rolünü üstlenebilir. Örneğin test çıktılarını beklenen sonuçlarla karşılaştıran ya da yapısal kapsamı ölçen bir araç bu amaçla kullanılabilir; çıktısı ayrıca

doğrulanmıyorsa araç kalifikasyonu (tool qualification) koşullarını sağlaması gerekir (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)).

Bağımsızlık her hedefte (objective) aranmaz: DO-178C'nin hedef tablolarında hangi hedefin hangi yazılım seviyesinde (software level) bağımsızlıkla karşılanacağı ayrıca işaretlidir. Kaba çizgileriyle:

Yazılım seviyesi	Doğrulama hedeflerinde bağımsızlık
Seviye A	En geniş kapsam: 71 hedefin 30'u bağımsızlıkla karşılanır. Kaynak kodun gereksinimlere ve mimariye uygunluğu, test sonuçlarının değerlendirilmesi ve kapsam analizleri dahil pek çok doğrulama hedefi bu gruptadır
Seviye B	Daha dar: 69 hedefin 18'i. Gereksinimlerin bir üst seviyeye uygunluğu ve doğruluğu, kaynak kodun düşük seviyeli gereksinimlere uygunluğu ve yapısal kapsam gibi hedefler bağımsız kalır; mimari, test prosedürleri ve sonuçları ile gereksinim test kapsamı gibi hedeflerde bağımsızlık yalnızca Seviye A'da aranır
Seviye C	Doğrulama hedeflerinde bağımsızlık aranmaz; 62 hedeften bağımsızlıkla işaretli 5'i kalite güvencesi hedefleridir
Seviye D	Doğrulama hedeflerinde bağımsızlık aranmaz; 26 hedeften bağımsızlıkla işaretli 2'si kalite güvencesi hedefleridir

Seviye A ve B'deki sayılara kalite güvencesi hedefleri de dahildir. Bağımsızlık aranmayan hedeflerde de gözden geçirme ya da analiz yapılır; yalnızca bunu yazarın kendisinin yapması standartça yasak değildir. Yine de yazar dışı bir gözün bakması her seviyede iyi uygulamadır.

Pratikte bağımsızlık genellikle şöyle organize edilir:

- **Çapraz doğrulama:** Aynı takım içinde, A modülünü yazan mühendis B modülünün testlerini gözden geçirir; B'yi yazan da A'ninkileri. Küçük takımlarda en yaygın ve en ekonomik modeldir.
- **Ayrı doğrulama ekibi:** Büyük projelerde test geliştirme ve gözden geçirme işini yalnızca doğrulamaya ayrılmış bir ekip üstlenir. İzlenebilirlik ve kayıt disiplini daha kolay yönetilir; ancak alan bilgisi aktarımı için ek çaba gerekir.
- **Rol bazlı ayırım:** Aynı kişi projenin farklı bileşenlerinde farklı roller alabilir; kritik olan, bağımsızlığın arandığı hedeflerde aynı iş ürününü geliştiren ile doğrulayanın aynı kişi olmamasıdır.

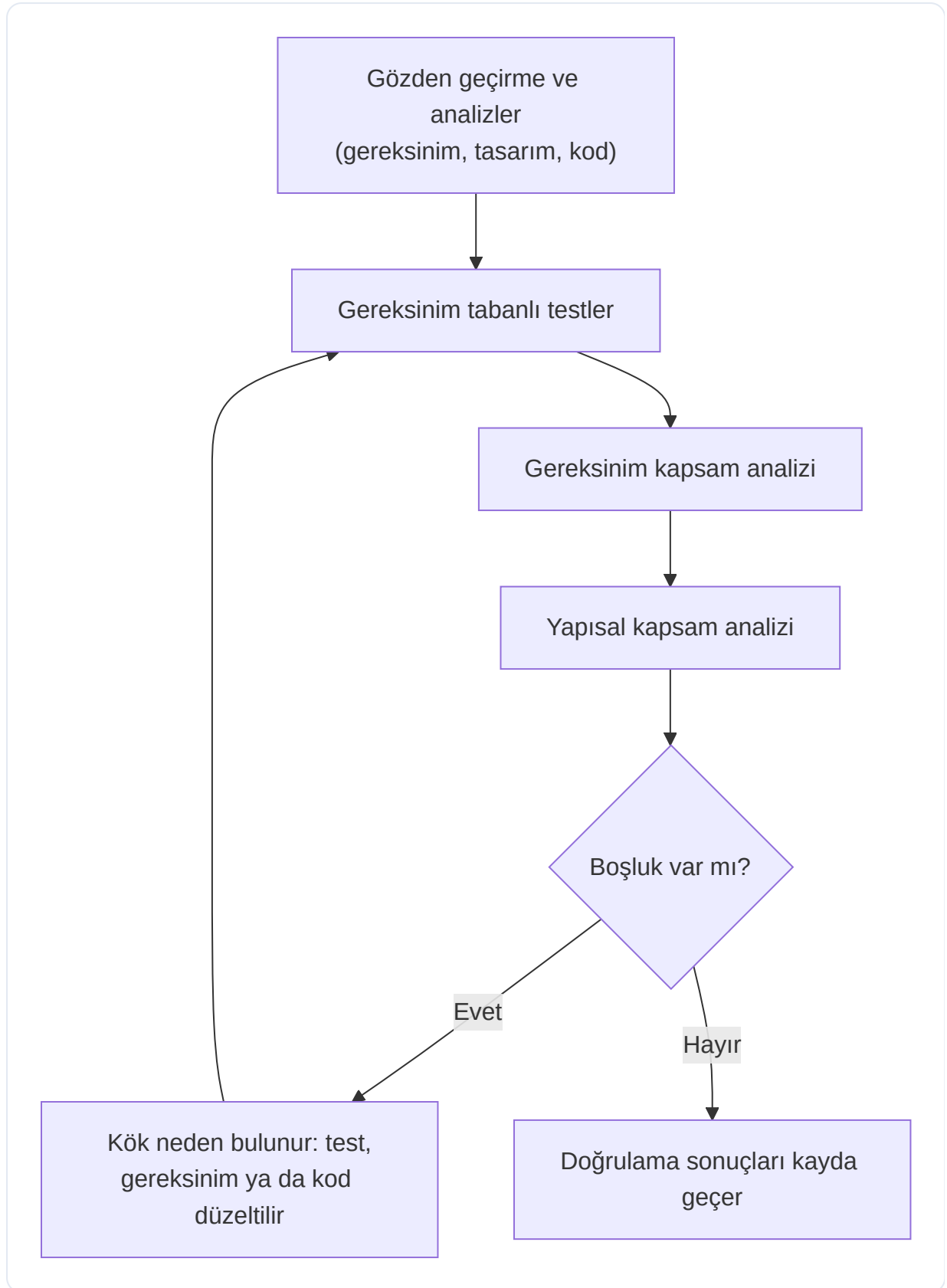
Bağımsızlığın kanıtı kayıtlardır: gözden geçirme tutanaklarında ve test sonuç kayıtlarında geliştiren ile doğrulayanın adları ayrı ayrı görünmelidir. Katılım aşaması (Stage of Involvement, SOI) denetimlerinde ilk bakılan şeylerden biri budur; "bağımsız yaptık" demek yetmez, kimin neyi yaptığı kayıttan okunabilmelidir.

Doğrulama yöntemleri

Doğrulama üç yöntemle yürür; üçü birbirinin yerine değil, birbirini tamamlayacak biçimde kullanılır:

Yöntem	Ne üretir	Tipik kullanım
Gözden geçirme (review)	Bir iş ürününün doğruluğuna ilişkin, çoğunlukla kontrol listesiyle yürütülen nitel değerlendirme	Gereksinim, tasarım, kod ve testlerin standartlara ve bir üst seviyeye uygunluğu
Analiz (analysis)	Yinelenabilir, çoğu zaman araç destekli kanıt	Zamanlama, yığın ve bellek kullanımı; kapsam ve bağlaşım analizleri
Test	Çalışan kodun davranışının gözlemi	Gereksinimlerin karşılandığının ve istenmeyen davranış bulunmadığının gösterilmesi

Gözden geçirme ve analiz testi beklemes, geliştirmeye birlikte yürür; çünkü bazı hatalar kodu çalıştırarak değil, okuyarak daha erken ve daha ucuza yakalanır. Sınır değer ya da durum geçişi gibi adlar ise ayrı yöntem değil, test durumu (test case) türetme teknikleridir. Akışın tamamı şöyle özetlenebilir:



Bu faaliyetlerin nasıl yürütüleceği yazılım doğrulama planında (Software Verification Plan, SVP) tanımlanır (bkz. [5. Yazılım Planlama](#)). Gözden geçirme ve analiz prosedürleriyle test durumları ve prosedürleri, yazılım doğrulama durumları ve prosedürleri (Software Verification Cases and

Procedures, SVCP) verisini; bunların uygulanmasıyla elde edilen kayıtlar yazılım doğrulama sonuçları (Software Verification Results, SVR) verisini oluşturur.

Gözden geçirmeler

Gözden geçirme, bir iş ürününün başka gözler tarafından sistematik biçimde incelenmesidir. En ucuz doğrulama yöntemidir, çünkü hatayı ürün henüz kağıt üzerindeyken yakalar: gereksinimdeki bir belirsizlik gözden geçirmede on dakikada düzeltilir; aynı belirsizlik teste kadar yaşarsa kod, test ve izlenebilirlik zinciri birlikte değişir.

Her geliştirme çıktısının kendine özgü bir gözden geçirme odağı vardır:

- **Gereksinim gözden geçirmesi:** Her gereksinim tekil, doğrulanabilir ve belirsizlikten arınmış mı? "Hızlı", "uygun", "yeterli" gibi ölçülemeyen ifadeler var mı? Sistem gereksinimlerine izlenebilirlik (traceability) kurulmuş mu? Hata ve sınır durumları tanımlanmış mı?
- **Tasarım gözden geçirmesi:** Yazılım mimarisi gereksinimleri karşılıyor mu? Düşük seviyeli gereksinimler yüksek seviyeli gereksinimlerle (high-level requirements) tutarlı mı? Arayüzler, veri akışı ve kontrol akışı tanımlı mı? Tasarım kararlarının gerekçeleri kayıtlı mı?
- **Kod gözden geçirmesi:** Kaynak kod düşük seviyeli gereksinimleri eksiksiz ve doğru gerçekleştiriyor mu? Kodlama standardına uyuluyor mu? Gereksinime izlenmeyen kod parçası (potansiyel gereksiz kod) var mı? Sınır koşulları, taşma ve hata yolları ele alınmış mı?
- **Test gözden geçirmesi:** Test durumları gereksinimdeki her koşulu — normal ve gürbüzlük (robustness) — kapsıyor mu? Beklenen sonuçlar gereksinimden mi türetilmiş, yoksa kodun mevcut davranışından mı kopyalanmış? Test prosedürleri tekrarlanabilir mi?

Kontrol listeleri ve kayıt

Etkili gözden geçirmenin iki dayanağı vardır: **kontrol listesi** (checklist) ve **kayıt**. Kontrol listesi, gözden geçirmenin kişisel deneyime bırakılmasını önler; her gözden geçiren aynı asgari soruları sorar. İyi bir kontrol listesi kısa tutulur (20-30 madde), projede sık görülen hata türleriyle güncellenir ve "evet/hayır" ile yanıtlanabilen sorulardan oluşur.

Kayıt ise sertifikasyon kanıtıdır. Tipik bir gözden geçirme kaydında şunlar bulunur:

Alan	İçerik
Gözden geçirilen ürün	Doküman/dosya adı ve konfigürasyon sürümü
Katılımcılar	Gözden geçiren(ler) ve yazar — bağımsızlık buradan izlenir
Kullanılan kontrol listesi	Sürümüyle birlikte
Bulgular	Her bulgu için sınıf (büyük/küçük/öneri) ve karar
Kapanış	Bulguların giderildiğinin teyidi ve yeniden gözden geçirme kararı

Etkili akran gözden geçirmesi için pratikler

Deneyimin gösterdiği birkaç basit kural, gözden geçirmenin verimini belirgin artırır:

- Malzemeyi toplantıdan **önce** dağıtın; toplantı bulguları tartışmak içindir, ilk kez okumak için değil.
- Oturumları kısa tutun; bir saatten sonra bulgu bulma oranı hızla düşer. Büyük dokümanları parçalara bölün.
- Yazarı değil ürünü eleştirin; savunma refleksi başlayan bir gözden geçirme bulgu üretmeyi bırakır.
- Bulguları toplantıda **çözmeye çalışmayın**; kaydedin, sahiplendirin, kapanışını ayrıca takip edin.
- "Gözden geçirildi" damgasını, bulgular kapanmadan basmayın; açık bulgusu olan ürün konfigürasyon açısından hâlâ olgunlaşmamıştır.

Analizler

Analiz, bir özelliğin **tüm** koşullar için geçerli olduğunu akıl yürütme ile göstermeye çalışır; test ise yalnızca denenen durumlar için kanıt üretir. Bu yüzden "her zaman doğru olmalı" türünden özellikler — zamanlama sınırları, bellek sınırları, kaynak tüketimi — yalnız testle gösterilemez; analiz gerekir ve ölçümle desteklenir. Aviyonik projelerde en sık karşılaşılan analizler şunlardır:

En kötü durum yürütme süresi analizi

En kötü durum yürütme süresi (worst-case execution time, WCET) analizi, her kritik görevin en olumsuz koşulda bile ayrılan zaman dilimini aşmadığını gösterir. Ortalama süre yanıltıcıdır; önemli olan ön bellek ıskalamaları, kesmeler ve en uzun kod yolunun üst üste geldiği durumdur. İki temel yaklaşım vardır:

- **Ölçüme dayalı:** Kod hedef donanımda çalıştırılır, en uzun gözlenen süreye bir emniyet payı eklenir. Basittir ama en kötü yolun gerçekten tetiklendiğinden emin olmak zordur.
- **Statik analiz:** Kodun kontrol akışı ve işlemci modeli üzerinden üst sınır hesaplanır. Kanıt gücü yüksektir; modern işlemcilerde modelleme güçlüğü nedeniyle çoğu proje iki yaklaşımı birleştirir.

WCET sonuçları zamanlama bütçesiyle karşılaştırılır ve kalan pay (margin) raporlanır; sertifikasyon otoriteleri tükenmek üzere olan zaman bütçelerini yakından sorgular.

Çok çekirdekli işlemcilerde iş daha da zorlaşır: çekirdekler ön bellek, veri yolu ve bellek denetleyicisi gibi kaynakları paylaştığından bir çekirdekteki yük, diğerindeki görevin süresini uzatabilir. Bu girişim (interference) kanalları belirlenip sınırlandırılmadan WCET savunulamaz;

otoritelerin beklentisi FAA AC 20-193 ve EASA AMC 20-193'te toplanmıştır. Konunun işletim sistemi tarafı için [20. Gerçek Zamanlı İşletim Sistemleri](#) bölümüne ve [Ek B: Gerçek Zamanlı İşletim Sistemlerinde Endişe Alanları](#) sayfasına bakılabilir.

Yığın kullanımı analizi

Yığın (stack) taşması, emniyet-kritik sistemlerde sinsi bir hata kaynağıdır: taşma anında değil, bozduğu veriler kullanıldığında belirti verir. Yığın analizi, en derin çağrı zinciri ile kesme yükü üst üste geldiğinde bile ayrılan yığın alanının aşılmadığını gösterir. Statik çağrı grafiği üzerinden hesap yapılır; işlev işaretçileri ve özyineleme bu hesabı zorlaştırdığından çoğu kodlama standardı özyinelemeyi zaten yasaklar. Hesap, hedefte doldurulmuş desenle (stack painting) yapılan ölçümle çapraz kontrol edilir.

Bellek haritası analizi

Bellek haritası analizi, bağlayıcı (linker) çıktısındaki yerleşimin tasarımıyla uyumunu denetler: kod, sabit veri ve değişken veri doğru bölgelere mi yerleşmiş; bölgeler taşıyor mu ya da birbirine giriyor mu; korumalı bölgelere (örneğin bir bölümlenme sınırının ötesine) beklenmedik bir yerleşim var mı? Yazılım bölümlenmesi (software partitioning) kullanan sistemlerde bu analiz, bölümler arası izolasyon iddiasının temel dayanaklarından biridir (bkz. [21. Yazılım Bölümlenmesi](#)).

Bağlantı ve yükleme analizi

Bağlantı (link) analizi, çalıştırılabilir nesne kodunun doğru bileşen sürümlerinden, çözülmemiş sembol kalmadan ve beklenmeyen kütüphane kodu çekilmeden oluştuğunu gösterir. Yükleme (load) analizi ise imajın (image) hedef donanıma bütünlüğü bozulmadan yüklendiğini doğrular: bellek aralığı uygunluğu ile sağlama toplamı (checksum) ya da döngüsel artıklık denetimi (cyclic redundancy check, CRC) bu kapsamdadır. Sahada yüklenebilir yazılım (field-loadable software) içeren sistemlerde yükleme prosedürünün doğruluğu ve yükleme işlevinin istem dışı etkinleşmesine (inadvertent enabling) karşı korumalar da doğrulanır; ayrıntısı [18. Sahada Yüklenebilir Yazılım](#) bölümündedir.

Tüm bu analizlerin ortak disiplini şudur: analiz **konfigürasyonu belli** bir derleme sürümü (build) üzerinde yapılır ve o sürüm değiştiğinde tekrarlanır ya da değişikliğin analizi etkilemediği gerekçelendirilir. Son sürümde unutulmuş eski bir WCET raporu, denetimde kanıt değil bulgu olur.

Testin rolü

Test, yazılımın somut davranışını gözlemler. Bir gereksinim ne kadar iyi yazılmış olursa olsun, test ya da gerekçelendirilmiş bir analizle desteklenmiyorsa sertifikasyon açısından eksik kalır.

DO-178C'de testler **gereksinim tabanlıdır** (requirements-based testing): her test durumu bir gereksinime dayanır ve iki tür test durumu beklenir. **Normal aralık** (normal range) test durumları, geçerli eşdeğerlik sınıflarını ve geçerli sınır değerleri kullanarak yazılımın isteneni yaptığını gösterir. **Gürbüzlük** test durumları ise geçersiz değerler, anormal başlatma, bozuk veri, aşılın çerçeve süresi ve izin verilmeyen durum geçişi gibi anormal girdi ve koşullarda yazılımın tanımlı biçimde davrandığını gösterir. İkisi de gereksinime dayanır: geçersiz girdide ne olacağı gereksinimde yazmıyorsa, eksik olan testten önce gereksinimdir.

Test stratejileri

Gereksinim tabanlı test, gereksinimi okuyup akla ilk gelen senaryoyu yazmak değildir; girdi uzayını sistematik tekniklerle taramaktır. Yerleşik teknikler şunlardır:

- **Eşdeğerlik sınıfı ayrımı (equivalence class partitioning):** Girdi uzayı, yazılımın aynı biçimde davranması beklenen sınıflara bölünür ve her sınıftan en az bir temsilci test edilir. Örneğin 0–350 knot aralığında geçerli bir hız girdisi için üç sınıf vardır: aralığın altı, aralığın içi, aralığın üstü.
- **Sınır değer testi (boundary value testing):** Hatalar sınıfların ortasında değil, kenarlarında yoğunlaşır (< yerine <= yazmak gibi). Her sınırın kendisi, bir altı ve bir üstü test edilir.
- **Durum geçiş testi (state transition testing):** Durum makinesi içeren yazılımlarda her geçerli geçiş en az bir kez tetiklenir; geçersiz geçiş denemelerinin reddedildiği de ayrıca gösterilir (gürbüzlük).
- **Karar tablosu testi (decision table testing):** Birden çok koşulun birleşimine göre davranan mantık için koşul kombinasyonları tablo hâlinde yazılır ve her anlamlı satır test edilir. Bu teknik, MC/DC hedefiyle doğal olarak örtüşür.

Küçük bir C örneği üzerinden sınır değer düşünelim:

```
#include <stdbool.h>
#include <stdint.h>

#define HIZ_ALT_SINIR_KNOT (0)
#define HIZ_UST_SINIR_KNOT (350)

/* Gereksinim: Hız 0-350 knot aralığındaysa geçerli kabul edilir. */
bool hiz_gecerli_mi(int32_t hiz_knot)
{
    return (hiz_knot >= HIZ_ALT_SINIR_KNOT) && (hiz_knot <= HIZ_UST_SINIR_KNOT);
}
```

Bu işlev için 0, 1, 349 ve 350 geçerli sınıfın sınır değerleri, 175 ise temsilcisidir; bunlar normal aralık test durumlarıdır. -1 ve 351 geçersiz sınıfların sınır değerleri, INT32_MIN ve INT32_MAX türün uç değerleridir; bunlar gürbüzlük test durumlarıdır. İşlevin görevi zaten geçerlilik denetimi olduğundan geçersiz girdide beklenen sonuç (yanlış) gereksinimden okunabilir: gürbüzlük testi beklenen sonucu tahmin etmez, gereksinimden alır.

Test seviyeleri

Gereksinim tabanlı testler üç seviyede yürütülür; her seviye başka tür hataları yakalamaya uygundur:

Test seviyesi	Ne gösterir	Tipik yakaladığı hatalar
Donanım/yazılım entegrasyon testi (hardware/software integration testing)	Yazılımın hedef bilgisayar ortamında yüksek seviyeli gereksinimleri karşıladığını	Kesme işleme ve zamanlama hataları, donanım arayüzü uyumsuzluğu, yığın ve bellek sorunları, donanım arızasına yanlış tepki
Yazılım entegrasyon testi (software integration testing)	Bileşenlerin birbiriyle ve yazılım mimarisiyle uyumlu çalıştığını	Arayüz ve birim uyumsuzluğu, yanlış başlatılmış veri, parametre ya da çağrı sırası hatası, veri bozulması
Düşük seviyeli test (low-level testing)	Her bileşenin düşük seviyeli gereksinimlerini karşıladığını	Algoritma ve döngü hataları, taşma, yanlış mantık kararı, geçersiz girdinin işlenmemesi

Seviyeler zorunlu bir sıra değildir; amaç her gereksinimi, hatasının ortaya çıkabileceği yerde sınamaktır. Birim düzeyinde kusursuz iki modül, biri metre diğeri feet konuşuyorsa birlikte yanlıştır; bunu ancak entegrasyon testi gösterir. Öte yandan üst seviyede koşulan bir test düşük seviyeli gereksinimi ve ilgili yapısal kapsamı da karşılıyorsa aynı testi alt seviyede yinelemek gerekmez.

Zamanlama, iş çıkarma (throughput) ve kaynak kullanımı gereksinimleri de test edilir: performans testleri bu gereksinimlerin hedef donanımda karşılandığını gösterir ve WCET analiziyle birbirini tamamlar.

Bazı hatalar yalnız hedef bilgisayar ortamında ortaya çıkar; bu yüzden tercih edilen test ortamı hedefe yüklenmiş yazılımdır ve seçilmiş testler her durumda entegre hedef ortamında koşulur. Emülatör ya da ana bilgisayar simülasyonundan kredi alınacaksa, ortamın hedeften farklarının hata yakalama yeteneğine etkisi değerlendirilir; o ortamda yakalanamayacak hataların hangi başka doğrulama faaliyetiyle yakalanacağı SVP'de belirtilir. Emülatör ya da simülatörün kendisi de araç kalifikasyonu gerektirebilir.

Test planlama ve test geliştirme

Test faaliyeti üç ayrı iş ürünü üretir ve bu ayrım önemlidir:

İş ürünü	İçerik	Sorusu
Test durumu (test case)	Girdi, ön koşullar, beklenen sonuç ve geçti/kaldı ölçütü; testin amacı ve izlendiği gereksinim	Ne test edilecek?
Test prosedürü (test procedure)	Test durumlarını kurup çalıştırmak ve sonucu değerlendirmek için adım adım talimat veya betik; kullanılacak test ortamı	Nasıl çalıştırılacak?

İş ürünü	İçerik	Sorusu
Test sonucu (test result)	Koşulan yazılım, prosedür ve ortam sürümü; gözlenen sonuç ve geçti/kaldı kararı	Ne gözlemlendi?

Test durumları gereksinimden türetilir ve gereksinim değişmedikçe kararlıdır; test prosedürleri ise test ortamına (yazılım entegrasyon ortamı, hedef donanım) bağlıdır. Planlama aşamasında hangi testin hangi ortamda ve hangi seviyede koşacağı ve sonuçların nasıl kaydedileceği belirlenir.

Geliştirme boyunca testler defalarca koşulur; uyum kanıtı sayılan ise **kredi amaçlı (for-credit) koşudur**: konfigürasyon kontrolündeki yazılım sürümü, gözden geçirilmiş test durumları ve prosedürleri ve kayıtlı bir test ortamıyla yapılan, sonuçları SVR'ye giren koşu. Hedef donanımda ya da hedefe denkliği gerekçelendirilmiş bir ortamda yapılır. Bu koşuya hangi koşullarla girileceği geçiş kriterlerinde tanımlanır (bkz. [Ek A: Örnek Geçiş Kriterleri](#)); otoritenin doğrulama verisinde baktığı noktalar ise [SW SOI-3](#) kontrol listesindedir.

İyi bir testin nitelikleri

- **Gereksinime izlenir:** Hangi gereksinimi doğruladığı testin üzerinde yazar; "faydalı görünen" ama hiçbir gereksinime bağlanmayan test, gereksinim eksikliğinin işaretidir.
- **Beklenen sonucu gereksinimden alır:** Beklenen değer koddan kopyalanırsa test, kodun kendi kendine eşit olduğunu kanıtlar — hatayı değil.
- **Tekrarlanabilir:** Aynı derleme sürümü ve aynı ortamda her koşuda aynı sonucu verir; zamanlamaya veya koşu sırasına gizlice bağımlı test güven vermez.
- **Başarısız olmayı bilir:** Sonucu otomatik ve net biçimde geçti/kaldı olarak değerlendirir; "çıktıya göz atın" diyen prosedür kanıt üretmez.
- **Gürbüzlüğü unutmaz:** Geçersiz girdi, sınır aşımı ve hata yolları da normal senaryolar kadar testin konusudur.

Doğrulamanın doğrulanması

Testlerin kendisi de bir iş ürünüdür ve doğrulanır: test prosedürleri test durumlarını doğru yansıtıyor mu, sonuçlar doğru mu ve beklenenle gözlenen arasındaki her fark açıklanmış mı; testler gereksinimleri gerçekten kapsıyor mu, kodun yeterince derinine iniyor mu? DO-178C bu soruları ayrı bir hedef tablosunda toplar.

Gereksinim kapsam analizi

Gereksinim kapsam analizi (requirements-based test coverage analysis) şu soruyu yanıtlar: **her gereksinim, en az bir test durumu ile karşılanıyor mu ve bu testler gereksinimin tamamını mı sınıyor?** İzlenebilirlik verisi üzerinden yürütülür: gereksinimden test durumuna, test durumundan test prosedürüne ve koşu sonucuna giden zincir iki yönde de uçtan uca takip edilir.

Analizin iki düzeyi vardır ve ikisi de gereklidir:

- 1. Eşleme kontrolü:** Her yüksek ve düşük seviyeli gereksinimin karşısında en az bir test durumu var mı? Bu, iz verisinden (biçimi ne olursa olsun) mekanik olarak okunabilir.
- 2. Yeterlilik kontrolü:** Eşlenen testler gereksinimin *tüm* koşullarını — aralıkları, kipleri, hata durumlarını, gürbüzlük beklentilerini — kapsıyor mu? Bu kısım mekanik değildir; testleri gereksinimle yan yana okuyan bir insan ister. "Gereksinim başına bir test var" demek, gereksinim üç kip tanımlıyorsa ve test yalnızca birini deniyorsa yeterlilik açısından boşluktur.

Seviye D'de bu analiz yalnızca yüksek seviyeli gereksinimler için aranır; düşük seviyeli gereksinimlere dayalı test ve test kapsamı hedefleri Seviye C ve üstünde geçerlidir.

Analiz sonunda çıkan boşluklar tipik olarak dört gruptan birine düşer ve her grubun çaresi farklıdır:

Boşluk türü	Anlamı	Çözüm
Testsiz gereksinim	Gereksinim yazılmış, test unutulmuş	Test durumu eklenir
Eksik senaryo	Test var ama gereksinimin bazı koşulları denenmemiş	Test durumu genişletilir
Test edilemeyen gereksinim	Gereksinim ölçülebilir/doğrulanabilir yazılmamış	Gereksinim düzeltilir — test değil
Analizle karşılanan gereksinim	Test yerine analiz uygun (örn. "her koşulda" özellikleri)	Analiz kanıtı izlenebilirliğe bağlanır ve gerekçelendirilir

Deneyimden bir uyarı: gereksinim kapsam analizini projenin sonuna bırakmayın. Test geliştirme ilerledikçe iz verisini güncel tutan projelerde bu analiz birkaç günlük bir teyittir; sona bırakan projelerde ise yüzlerce boşluğun aynı anda ortaya döküldüğü, takvimi sarsan bir krizdir.

Yapısal kapsam analizi

Yapısal kapsam analizi (structural coverage analysis), testlerin kodun ne kadarını çalıştırdığını gösterir. Ancak bu ölçüm tek başına yeterli değildir; amaç "yüksek sayı" değil, anlamlı kanıttır. Kapsam verisi, gereksinim tabanlı testlerle birlikte yorumlanmalıdır.

Hangi kapsam ölçütünün arandığı yazılım seviyesine bağlıdır ve ölçütler birbirinin üzerine inşa edilir:

Ölçüt	Neyi ister	Arandığı seviye
Satır kapsama (statement coverage)	Her çalıştırılabilir deyim (statement) en az bir kez çalışsın	C, B, A
Karar kapsama (decision coverage)	Her giriş ve çıkış noktası en az bir kez işletilsin, her karar olası bütün sonuçlarını alsın	B, A

Ölçüt	Neyi ister	Arandığı seviye
Değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC)	Bileşik karardaki her koşulun sonucu tek başına etkilediği gösterilsin	A
Veri ve kontrol bağlaşımları analizi	Bileşenler arası veri ve kontrol ilişkileri testlerde işletilmiş olsun	C, B, A

Seviye D'de yapısal kapsam hedefi yoktur. Standardın sözlüğünde karar (decision) yalnızca dallanma noktası değildir: koşullardan ve Boole işleçlerinden kurulu her Boole ifadesi karardır. MC/DC'nin mantığı bir örnekle netleşir:

```
if ((basinc_dusuk && motor_calisiyor) || bakim_kipi) {
    uyarı_ver();
}
```

Karar kapsama için bu `if`'in bir kez doğru, bir kez yanlış olması yeter — üç koşuldan biri hiç etkisini göstermeden bu sağlanabilir. MC/DC ise her koşul için, kararın sonucunu yalnızca o koşulun değiştirdiğini gösteren bir test çifti, yani bir bağımsızlık çifti (independence pair) ister. Bu karar için dört test yeter:

Test	basinc_dusuk	motor_calisiyor	bakim_kipi	Karar
1	doğru	doğru	yanlış	doğru
2	yanlış	doğru	yanlış	yanlış
3	doğru	yanlış	yanlış	yanlış
4	doğru	yanlış	doğru	doğru

1–2 çifti `basinc_dusuk`, 1–3 çifti `motor_calisiyor`, 3–4 çifti `bakim_kipi` koşulunun bağımsız etkisini gösterir. Böylece örneğin `bakim_kipi` teriminin gereksiz (veya yanlış) olup olmadığı test kümesinde görünür olur. N koşullu bir karar için MC/DC tipik olarak N+1 testle sağlanabilir; üstünlüğü, 2 üzeri N kombinasyonun tamamını denemeden koşul düzeyinde duyarlılık sağlamasıdır.

Bu çiftlerde öteki koşullar sabit tutulmuştur; bu biçime benzersiz neden MC/DC (unique-cause MC/DC) denir. Aynı koşul kararda birden çok kez geçiyorsa (bağlı koşul, coupled condition) ötekileri sabit tutmak mümkün olmaz; o zaman yalnızca sonucu etkileyebilecek koşulların sabit kalmasını arayan maskeleye MC/DC (masking MC/DC) kullanılır. DO-178C'nin MC/DC tanımı iki biçimi de kabul eder. C'deki kısa devre değerlendirmesi (short-circuit evaluation) hangi koşulların gerçekten değerlendirildiğini değiştirdiğinden test seti buna göre yorumlanır. Kendi kararlarınızın bağımsızlık çiftlerini [MC/DC Test Seti Üretici](#) ile deneyebilirsiniz.

Önemli bir ayrıntı: kapsam verisi **gereksinim tabanlı testlerin koşusundan** toplanır. Kapsamı yükseltmek için gereksinimsiz "kapsam testi" yazmak yöntemi tersine çevirir; düşük kapsam, ya testlerin ya gereksinimlerin ya da kodun eksik/fazla olduğunun işaretidir ve önce bu kök neden bulunur.

Kapsam, kaynak kod ya da nesne kodu düzeyinde ölçülebilir. Çoğu araç bunun için koda sayaçlar ekler, yani kodu donatır (instrumentation). Donatılmış kod uçacak kod değildir; boyutu ve zamanlaması farklıdır. Bu yüzden yaygın yaklaşım, kapsamı donatılmış derleme sürümünden toplamak, kredi amaçlı test sonuçlarını donatılmamış kodla elde etmek ve iki koşulun aynı sonucu verdiğini göstermektir. Kapsam aracı bir doğrulama faaliyetini otomatikleştirdiği ve bir boşluğu gözden kaçırabileceği için, çıktısı ayrıca doğrulanmıyorsa kalifiye edilir (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)).

Seviye A'da ek olarak, kapsam hangi kod biçimi üzerinde ölçülmüş olursa olsun, kaynak kod ile nesne kodu arasındaki mesafeye bakılır. Derleyici ya da bağlayıcı, kaynak koda doğrudan izlenemeyen nesne kodu üretiyorsa — örneğin derleyicinin eklediği dizi sınırı denetimleri ya da örtük ilkendirme kodu — bu kod belirlenir ve doğruluğu ek doğrulamayla gösterilir. Beklenti böyle bir kodun yokluğunu kanıtlamak değil, var olanı bulup doğrulamaktır; derleyici seçenekleri ve eniyileme düzeyi bu iş yükünü doğrudan belirlediği için planlama kararıdır.

Veri bağlaşımı ve kontrol bağlaşımı analizi

Yapısal kapsamın bileşen içi ölçümünü, bileşenler **arası** bir analiz tamamlar: veri bağlaşımı (data coupling) ve kontrol bağlaşımı (control coupling) analizi. Veri bağlaşımı, bir bileşenin başka bileşenlerle paylaştığı, yani tek başına denetlemediği veriye bağımlı olmasıdır; kontrol bağlaşımı ise bir bileşenin diğerinin çalışmasını etkilemesidir (çağrı, kip seçimi vb.). Analizin amacı, mimaride tanımlanan tüm bağlaşımların gereksinim tabanlı testler sırasında gerçekten **işletildiğini** göstermektir: her arayüz en az bir testte veri taşımış mı, her çağrı ilişkisi tetiklenmiş mi? Bu analiz, birim testleri kusursuz olsa bile entegrasyonda saklanan arayüz hatalarını hedefler ve tipik olarak entegrasyon testi kapsam verisiyle beslenir. Seviye A, B ve C'de beklenir.

Somut bir örnek: sensör işleme bileşeninin yazdığı `irtifa_gecerli` bayrağını kontrol yasası bileşeni okuyorsa, veri bağlaşımının kanıtı bu bayrağın hem doğru hem yanlış değeriyle okuyan bileşene ulaştığı ve davranışını değiştirdiği entegrasyon testleridir; kontrol bağlaşımının kanıtı ise iki bileşenin tasarımıda öngörülen sırayla çağrıldığıının aynı testlerde gözlenmesidir. Analizin dayanağı mimaride tanımlı arayüzler olduğundan, örtük global değişkenlerle kurulmuş bağlaşımlar burada bulgu olarak geri döner (bkz. [7. Yazılım Tasarımı](#)).

Kapsam boşluklarının ele alınması

Kapsam analizi bittiğinde çalışmamış kod parçaları listelenir ve her birinin nedeni bulunur. Kapsanmayan kodun dört olası nedeni vardır; yapılacak iş nedene göre değişir:

- **Test eksikliği:** Kod bir gereksinime izlenebiliyor ama gereksinim tabanlı test durumları ya da prosedürleri o yolu tetiklememiş. Çözüm yeni ya da genişletilmiş test durumudur.
- **Gereksinim eksikliği:** Kod gerekli bir davranışı gerçekleştiriyor ama bu davranış hiçbir gereksinimde yazmıyor. Önce gereksinim yazılır ve izlenebilirliği kurulur, sonra testi eklenir.
- **Gereksiz kod (extraneous code):** Hiçbir sistem ya da yazılım gereksinimine izlenemeyen kod ya da veri; unutulmuş hata ayıklama çıktısı gibi. Ölü kod (dead code) bunun özel hâlidir: bir geliştirme hatası sonucu çalıştırılabilir nesne kodunda bulunan ve hedef ortamın hiçbir işletim konfigürasyonunda çalıştırılmayan kod. Gereksiz kodun kaldırılması beklenir; kaldırma da bir değişiklik olduğundan etkisi analiz edilir ve gereken doğrulama yinelenir.
- **Devre dışı bırakılmış kod (deactivated code):** Gereksinime izlenebilen ve tasarım gereği ya hiç çalıştırılması amaçlanmayan ya da yalnızca belirli konfigürasyonlarda çalışan kod (örneğin başka bir uçak tipine ait seçenek). Kaldırılmaz; ancak devre dışı kalma mekanizmasının (pin programlama, konfigürasyon verisi) istem dışı etkinleşmeye izin vermediği ayrıca doğrulanır. Belirli konfigürasyonlarda çalışan kod, o konfigürasyon kurularak test edilir. Koşullu derlemeyle (conditional compilation) imajın dışında bırakılan kod çalıştırılabilir nesne kodunda bulunmadığı için sözlükteki tanımın dışında kalır; planlarda hangi başlık altında anılırsa anılsın her derleme varyantı ayrıca doğrulanır.

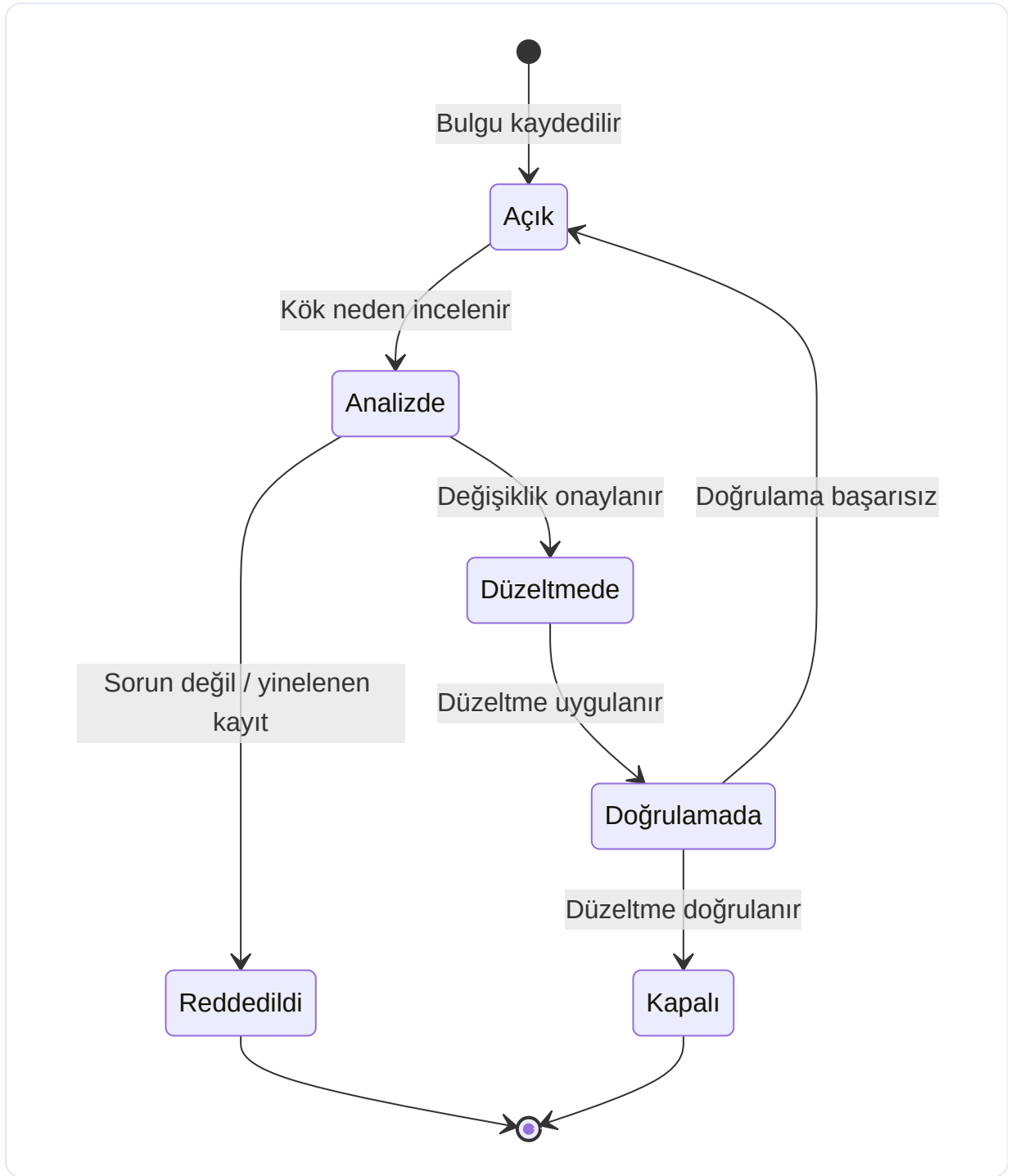
Savunmacı programlama yapıları — normalde ulaşılmaması gereken bir `default` dalı gibi — bu ayrımda özel bir yer tutar: tasarım gereği vardır ve ölü kod sayılmaz, ama testle tetiklenmesi güç olabilir. Böyle yapılar için gereksinim ya da tasarım dayanağı gösterilir ve testle ulaşılamayan kısım analizle gerekçelendirilir.

Bu sınıflandırmanın kayıtlı ve gerekçeli olması gerekir; "kapsam %100 değil ama önemsiz" cümlesi denetimde kabul görmez, her boşluğun tek tek hesabı verilir. Sınıfların ayrıntısı ve karar akışı [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#) bölümündedir.

Problem raporlama

Doğrulama bulgu üretir; bulgular ancak kayıt altına alınırsa değer taşır. Problem raporu (problem report) üç tür bulgunun resmî kayıdır: konfigürasyon kontrolüne alınmış bir iş ürünündeki kusur (kod hatası, gereksinim belirsizliği, test hatası, doküman tutarsızlığı), plan ve standartlardan süreç sapması ve yazılımın anormal davranışı. Temel çizgi (baseline) öncesindeki gözden geçirme bulguları ise gözden geçirme kaydında izlenip kapatılır; sınırın tam yeri yazılım konfigürasyon yönetimi planında (Software Configuration Management Plan, SCMP) tanımlanır (bkz. [10. Yazılım Konfigürasyon Yönetimi](#)). Hangi kayıta izlenirse izlensin, sözlü aktarılan ya da e-postada kalan bir bulgu sertifikasyon açısından hiç bulunmamış gibidir.

İyi bir problem raporu en azından şunları içerir: sorunun gözlemlendiği derleme sürümü ve ortam (konfigürasyon sürümüyle), yeniden üretme adımları, gözlenen ve beklenen davranış, etkilenen iş ürünleri ve sorumlu; kapanışta yapılan düzeltici faaliyet de rapora işlenir. Tanım, sorunun olası emniyet ve işlev etkisinin değerlendirilmesine yetecek ayrıntıda olmalıdır. Rapor açıldıktan sonra tanımlı bir yaşam döngüsünden geçer:



Proje içinde sınıflandırma iki eksenle yapılır ve karar mekanizmalarını besler:

- **Etki (önem):** Sorun emniyeti veya bir gereksinim uyumunu mu etkiliyor, yoksa yalnızca dokümantasyon ya da kullanım kolaylığı sorunu mu? Emniyet etkisi olan raporlar sistem emniyet değerlendirme sürecine geri beslenir.
- **Kök nedenin bulunduğu ürün:** Hata kodda mı, gereksinimde mi, testte mi, yoksa araçta mı? Bu eksen, düzeltmenin hangi süreçten yeniden geçeceğini belirler — gereksinim hatası yalnızca kod yamasıyla kapatılamaz.

Her problem raporunun kapanışı da doğrulamadır: düzeltme yapıldıktan sonra sorunu bulan test yeniden koşulur ve değişiklik etki analizine (change impact analysis) göre etkilenen diğer testler tekrarlanır (regresyon, regression).

Açık problem raporlarıyla sertifikasyona gitmek

Gerçek projeler sertifikasyon anına sıfır açık raporla varmaz; bu normaldir. Kritik olan, açık kalan her raporun **değerlendirilmiş** olmasıdır. Otoritelerin bu konudaki güncel beklentisi FAA AC 20-189 ve EASA AMC 20-189'da toplanmıştır: açık raporlar en az aşağıdaki dört sınıfla sınıflandırılır ve her rapora, tablodaki öncelik sırasıyla tek bir sınıf atanır.

Sınıf	Kapsadığı açık rapor
Önemli (significant)	Katastrofik, tehlikeli ya da majör bir arıza durumuna (failure condition) yol açabilecek ya da işletme kurallarına uyumu etkileyebilecek problem
İşlevsel (functional)	Ürün, sistem ya da ekipman düzeyinde bir işlevi etkileyen problem
Süreç (process)	Emniyet ya da işlev etkisi doğuramayan süreç uygunsuzluğu
Yaşam döngüsü verisi (life-cycle data)	Süreç eksikliğine bağlı olmayan veri kusuru; örneğin bir doküman hatası

Sertifikasyon öncesinde açık raporlar tek tek gözden geçirilir ve her biri için şu sorular yanıtlanır: hangi sınıfta, hangi işlevi nasıl kısıtlıyor, işletme sınırlaması gerektiriyor mu, hangi sürümde düzeltilmesi planlanıyor? Yeterli gerekçesi olmayan "önemli" sınıfındaki rapor onaydan önce çözülür. Açık bırakılacaksa emniyet etkisinin neden kabul edilebilir olduğu, sistem düzeyindeki hafifletmeler (mitigation) ve işletme kısıtlarıyla birlikte açıkça yazılır; kısıtla hafifletilen rapor kapanmış değil, gerekçesiyle açık taşınan rapordur.

Değerlendirmenin özeti, işlevsel kısıtlar ve gerekçeyle birlikte yazılım başarı özetinde (Software Accomplishment Summary, SAS) yer alır ve sertifikasyon otoritesine sunulur (bkz. [12. Sertifikasyon İrtibatı](#)). Deneyimin öğrettiği kural şudur: denetçiyi rahatsız eden açık rapor sayısı değil, sahipsiz ve değerlendirilmemiş rapordur.

Bu bölümden akılda kalması gerekenler

- Doğrulama, kanıt üretme faaliyetidir; kanıt kayıtlarda — doğrulama durumları, prosedürleri ve sonuçlarında — yaşar.
- Bağımsızlık kişi düzeyindedir ve yalnızca işaretli hedeflerde, seviyeye göre aranır: Seviye A ve B'de doğrulama hedeflerini kapsar, Seviye C ve D'de yalnızca kalite güvencesi hedeflerinde kalır. Arandığı yerde kimin neyi yaptığı kayıttan okunabilmelidir.
- Test tek başına yeterli değildir; gözden geçirme ve analiz (WCET, yığın, bellek haritası, bağlantı/yükleme) de gerekir — en kötü durum sınırları yalnız testle gösterilemez, analiz gerekir ve ölçümle desteklenir.

- Testler gereksinim tabanlıdır; normal aralık ve gürbüzlük test durumları sistematik tekniklerle türetilir ve kredi, kontrol altındaki sürümle hedefte ya da denkliği gerekçelendirilmiş ortamda yapılan koşudan alınır.
- Gereksinim kapsamı eşlemeyle bitmez; testlerin gereksinimin tamamını sınıadığı da gösterilir.
- Yapısal kapsam ölçütü seviyeye göre artar (satır → karar → MC/DC); kapsam verisi gereksinim tabanlı testlerden toplanır ve her boşluk dört nedenden birine (test eksikliği, gereksinim eksikliği, gereksiz kod, devre dışı bırakılmış kod) bağlanır. Seviye A'da kaynak koda izlenemeyen nesne kodu ayrıca doğrulanır.
- Doğrulama bulguları problem raporuyla yaşar; sertifikasyona açık raporla gidilebilir, ama sınıflandırılmamış ve SAS'ta gerekçelendirilmemiş raporla gidilemez.

10. Yazılım Konfigürasyon Yönetimi

Yazılım konfigürasyon yönetimi (software configuration management), hangi iş ürününün hangi sürümde olduğunu, neden değiştiğini ve teslim edilen yazılımın tam olarak nelerden oluştuğunu her an gösterebilmeyi sağlar. Bu bölüm konfigürasyon yönetiminin hedeflerini ve faaliyetlerini, temel çizgileri, kontrol kategorilerini, ürettiği iş ürünlerini ve değişiklik etki analizini işlenmiş bir örnekle birlikte anlatır.

Konfigürasyon yönetimi neden gerekli?

Aviyonik projelerde yazılım yaşam döngüsü verisi (software life cycle data) — kitapta kısaca iş ürünü — birbirine bağlıdır. Bir gereksinim değiştiğinde bunun tasarıma, koda, testlere, gözden geçirme (review) kayıtlarına ve bazen sertifikasyon otoritesine (certification authority) sunulan veriye yansması gerekir. Doğrulama kanıtı da belirli bir içeriğe aittir: bir test sonucu, koşulduğu derlemenin ve sınıadığı gereksinim sürümünün kimliği bilinmiyorsa hiçbir şey kanıtlamaz. Konfigürasyon yönetiminin koruduğu şey bu yüzden dosya sürümleri değil, sertifikasyon kanıtının geçerliliğidir.

DO-178C'de konfigürasyon yönetimi; doğrulama, kalite güvencesi (quality assurance) ve sertifikasyon irtibatıyla birlikte bütünleyici süreçlerden (integral processes) biridir. Geliştirmenin ardından gelen bir adım değildir; planlamadan itibaren diğer süreçlere eşlik eder ve yazılım onaylandığında da bitmez, sistem ya da ekipman serviste kaldığı sürece sürer (bkz. [4. DO-178C ve Destekleyici Dokümanlara Genel Bakış](#)).

Hedefler ve seviyeye göre beklenti

DO-178C'nin konfigürasyon yönetimi hedefleri (objective), standardın Ek A'sındaki Tablo A-8'de altı satırda toplanır: konfigürasyon öğelerinin tanımlanması; temel çizgilerin ve izlenebilirliğin kurulması; problem raporlama, değişiklik kontrolü, değişiklik gözden geçirmesi ve durum muhasebesinin işletilmesi; arşivleme, geri getirme ve sürüm tesliminin sağlanması; yazılım yükleme kontrolü; yaşam döngüsü ortamının kontrolü. Standardın gövde metni aynı kapsamı dokuz hedef olarak sayar; tablo bunlardan dördünü (problem raporlama, değişiklik kontrolü, değişiklik gözden geçirmesi, durum muhasebesi) tek satırda birleştirir.

Altı hedefin tamamı Seviye A'dan Seviye D'ye kadar geçerlidir ve hiçbirinde bağımsızlık aranmaz. Yazılım seviyesi (software level) düştükçe hafifleyen şey hedeflerin kendisi değil, hangi verinin hangi sıklıkta kontrol edildiğidir; bunu aşağıda anlatılan kontrol kategorileri belirler. "Seviye D yazılımda konfigürasyon yönetimi gerekmez" düşüncesi bu yüzden yanlıştır.

Hedeflerin projede nasıl karşılanacağı yazılım konfigürasyon yönetimi planında (Software Configuration Management Plan, SCMP) yazılır: adlandırma ve sürümleme kuralları, temel çizgiler, problem raporu akışı, kurulun yetkisi, arşiv düzeni ve tedarikçi verisinin kontrolü oradadır (bkz. [5. Yazılım Planlama](#)). Faaliyetlerin fiilen yapıldığını konfigürasyon yönetimi kayıtları gösterir; sürecin plana uygun yürüdüğünü ise kalite güvencesi denetler.

Konfigürasyon yönetimi faaliyetleri

Konfigürasyon yönetimi tek bir işlem değil, projenin başından sonuna kadar süren bir faaliyetler bütünüdür. Pratikte en çok karşılaşılan faaliyetler şunlardır:

- **Konfigürasyon tanımlama (configuration identification):** Kontrol altına alınacak her iş ürününe — gereksinim dokümanı, tasarım verisi, kaynak kod dosyası, test prosedürü, derleme betiği — benzersiz bir kimlik ve sürüm numarası verilir. Bu kimliği taşıyan her birime konfigürasyon ögesi (configuration item) denir. Çalıştırılabilir nesne kodundan ayrı yüklenen parametre verisi ögesi (parameter data item, PDI) dosyaları da kendi kimliği ve sürümü olan konfigürasyon öğeleridir (bkz. [22. Konfigürasyon Verisi](#)). Bir öğe adlandırılıp tanımlanmadan onun "hangi sürümü" diye konuşmak mümkün değildir.
- **Temel çizgi oluşturma:** Belirli olgunluk noktalarında (örneğin gereksinim gözden geçirmesi tamamlandığında, ilk resmî test sürümü hazırlandığında) onaylı iş ürünleri kümesi dondurulur. Temel çizgiden sonraki her değişiklik, değişiklik kontrolünden geçmek zorundadır.
- **Problem raporlama (problem reporting):** Temel çizgiye alınmış bir iş ürünündeki kusur, plan ve standartlardan süreç sapması ve yazılımın anormal davranışı bir problem raporu (problem report) ile kayıt altına alınır. Rapor; belirtiyi, etkilenen öğeyi ve sürümü, kararı ve kapanış kanıtını içerir.
- **Değişiklik kontrolü ve değişiklik gözden geçirmesi (change control, change review):** Değişiklik kontrolü, kontrol altındaki öğeleri ve temel çizgileri izinsiz değişikliğe karşı korur; her değişikliğin öğenin kimliğine yansımaları, kaydedilmesini, onaylanmasını ve izlenmesini sağlar. Değişiklik kaynağına kadar izlenir ve yaşam döngüsü süreçleri, değişikliğin çıktılarını etkilediği noktadan başlayarak yinelenir; etkilenen verinin tamamı güncellenir. Değişiklik gözden geçirmesi bu kararın dayanağını üretir ve sonucunu izler: problemin ya da önerilen değişikliğin sistem gereksinimlerine ve yaşam döngüsü verisine etkisi değerlendirilir, sistem süreçlerine (sistem emniyet değerlendirmesi dahil) geri bildirim verilir, değişiklik onaylanır ya da reddedilir, onaylananın uygulandığı görülür ve karar etkilenen süreçlere iletilir. Çoğu projede bu değerlendirmeyi bir değişiklik kontrol kurulu (change control board) yapar; kurul standardın değil uygulamanın terimidir. İyi işleyen bir akış beş soruyu cevapsız bırakmaz: değişiklik neden gerekli, hangi iş ürünleri etkileniyor, kim onaylıyor, hangi doğrulama yeniden yapılacak ve değişiklik hangi sürüme girecek?
- **Konfigürasyon durum muhasebesi (configuration status accounting):** Hangi öğenin hangi sürümde olduğu, hangi problem raporlarının açık ya da kapalı olduğu ve hangi değişikliklerin hangi temel çizgiye girdiği her an raporlanabilir durumda tutulur.

- **Arşivleme, geri getirme ve sürüm teslimi (archive, retrieval, release):** Onaylı sürümler bozulmaya ve yetkisiz değişikliğe karşı korunan bir ortamda saklanır; yıllar sonra bile aynı içerik geri getirilebilmelidir. Saklama süresini projenin tercihi değil uçuşa elverişlilik (airworthiness) gerekleri belirler; pratikte veri, ürün serviste kaldığı sürece erişilebilir olmalıdır. Bu ufukta arşiv ortamı eskimeden yenilenir, kopyalar fiziksel olarak ayrı yerlerde tutulur ve geri getirme ara ara denenerek doğrulanır. Sürüm teslimi, bir ögenin kullanıma yetkili biçimde açılmasıdır: yazılım üretiminde yalnızca yayımlanmış ögeler kullanılır, yayımlama yetkisinin kimde olduğu bellidir ve en azından hedefe yüklenen çalıştırılabilir nesne kodu (executable object code) ile varsa PDI dosyaları bu yoldan geçer.
- **Yükleme kontrolü (load control):** Yazılımın ana kopyadan sisteme ya da ekipmana aktarılmasını kapsar. Yüklem fabrikada da sahada da yapılsa iki şey beklenir: yüklenmesi onaylanacak konfigürasyonları tanımlayan parça numaralama ve yüklem medyası kimliklendirme kuralları ile yazılımın donanımla uyumlu olduğunu gösteren kayıtlar. Uygulamada buna sağlama toplamı (checksum) ya da döngüsel artıklık denetimi (cyclic redundancy check, CRC) gibi bir bütünlük denetimi eşlik eder. Yüklem sahada yapılıyorsa konu genişler (bkz. [18. Sahada Yüklenebilir Yazılım](#)).
- **Yaşam döngüsü ortamının kontrolü:** Derleyici, bağlayıcı, test araçları ve işletim ortamının sürümleri de kayıt altındadır; aksi hâlde çalıştırılabilir nesne kodunun arşivlenen kaynaktan tutarlı biçimde yeniden üretilebilmesi garanti edilemez. Yazılımı derleyen ve yükleyen araçların çalıştırılabilir sürümleri en az aşağıda anlatılan ikinci kontrol kategorisinin gerektirdiği kadar kontrol edilir. Kalifiye edilmiş araçların çalıştırılabilir sürümleri ve kalifikasyon verisi de kontrolün içindedir; onlarda sıklığı araç kalifikasyonu belgesi DO-330 belirler (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)).

Temel çizgiler

Temel çizgi (baseline), gözden geçirilip onaylanmış ve o andan sonra yalnızca değişiklik kontrolüyle değiştirilebilen bir konfigürasyon ögeleri kümesidir. İşlevi, sonraki işe sabit bir zemin vermektir: "tasarım hangi gereksinim sürümüne göre yapıldı", "bu test hangi derlemeye karşı koşuldu" soruları ancak bir temel çizgi kimliğiyle cevaplanır.

DO-178C iki şeyi şart koşar: sertifikasyon kredisi alınacak konfigürasyon ögeleri temel çizgiye alınır ve yazılım ürünü için, yazılım konfigürasyon indeksinde tanımlanan bir ürün temel çizgisi kurulur. Ara temel çizgilerin sayısını ve adını ise belirlemez; bunlar SCMP'de, yaşam döngüsünün geçiş kriterleriyle birlikte tanımlanır. Yaygın bir kurgu şöyledir:

Temel çizgi	Tipik içerik	Neyin zemini olur?
Planlama	Planlar ve standartlar	Geliştirme faaliyetlerinin başlaması
Gereksinim	Yüksek seviyeli gereksinimler ve iz verisi (trace data)	Tasarım, test durumlarının geliştirilmesi
Tasarım	Yazılım mimarisi ve düşük seviyeli gereksinimler	Kodlama

Temel çizgi	Tipik içerik	Neyin zemini olur?
Kod ve derleme	Kaynak kod, derleme betikleri, çalıştırılabilir nesne kodu	Kredi alınacak test koşulları
Doğrulama	Test durumları ve test prosedürleri	Test koşulları ve kapsam analizi
Ürün	Yazılım konfigürasyon indeksinde tanımlanan sürümün tamamı	Otoriteye sunum, üretim ve yükleme

Hangi kurgu seçilirse seçilsin üç kural değişmez:

- **Temel çizgi değiştirilmez.** Değişiklik, ögenin yeni bir sürümünü ve gerektiğinde yeni bir temel çizgiyi doğurur; eskisi olduğu gibi arşivde kalır. "Temel çizgiyi güncelledik" cümlesi, iz bırakmadan yapıldıysa bir denetim bulgusudur.
- **Yeni temel çizgi türetildiği temel çizgiye izlenebilir.** İki temel çizgi arasındaki farkın hangi problem raporları ve değişiklik talepleriyle oluştuğu kayıttan okunabilmelidir. Buradaki izlenebilirlik (traceability), gereksinim–kod–test izlenebilirliği değil, sürümler arasındaki soy bağıdır.
- **Ara temel çizgiler küçük ve sık alınır.** Her temel çizgi bir doğrulama faaliyetine zemin olacak kadar olgun, ama "her şey bitsin" diye bekletilmeyecek kadar erken kurulur.

Kontrolün ne zaman başladığı da buradan çıkar: konfigürasyon tanımlama ve sürüm kontrolü projenin ilk gününden işler; resmî değişiklik kontrolü ve problem raporlama ise ilgili iş ürününün ilk temel çizgisiyle başlar. Temel çizgi öncesindeki gözden geçirme bulguları gözden geçirme kaydında izlenip kapatılır; sınırın tam yeri SCMP'de yazılıdır.

Kontrol kategorileri: CC1 ve CC2

DO-178C yukarıdaki faaliyetlerin tümünü her veri ögesine aynı sıklıkta uygulamaz. İki kontrol kategorisi (control category) tanımlanır; ikincisinin faaliyetleri birincinin alt kümesidir. Birinci kategori (CC1) tam kontroldür: temel çizgi, problem raporlama, değişikliğin kaydı ve onayı, değişiklik gözden geçirmesi ve durum muhasebesi dahil tüm faaliyetler uygulanır. İkinci kategori (CC2) daha hafif bir rejimdir: ögenin tanımlanması, türetildiği ögeye izlenebilmesi, değişirken bütünlüğünün ve kimliğinin korunması, yetkisiz değişikliğe karşı korunması, geri getirilebilmesi ve saklanması yeterlidir. Standardın Tablo 7-1'i bu ayrımı faaliyet faaliyet verir ve her kategori için asgariyi tanımlar; aşağıda aynı bilgi konuya göre gruplanmıştır:

Konu	CC1 ve CC2'de ortak	Yalnızca CC1'de
Kimlik ve soy bağı	Konfigürasyon tanımlama; ögenin türetildiği ögeye ve ait olduğu çıktıya ya da sürece izlenebilmesi	Temel çizgi kurma, temel çizginin korunması ve temel çizgiler arası izlenebilirlik
Sorun ve değişiklik	Değişikliğe karşı bütünlüğün korunması; değişen ögenin kimliğinin de değişmesi	Problem raporlama; değişikliğin kaydı, onayı ve izlenmesi; değişiklik gözden geçirmesi

Konu	CC1 ve CC2'de ortak	Yalnızca CC1'de
Durum bilgisi	—	Konfigürasyon durum muhasebesi
Arşiv ve teslim	Geri getirme; yetkisiz değişikliğe karşı koruma; veri saklama	Arşiv ortamının seçimi ve yenilenmesi, kopyaların ayrı yerde tutulması ve kopyalamanın doğrulanması; sürüm teslimi

CC2 "kontROLSÜZ" demek değildir: bir test sonuç dosyasının hangi koşuya ait olduğu bilinmeli, dosya yetkisiz biçimde değiştirilememeli ve yıllar sonra bulunabilmelidir. Aranmayan şey, o dosyadaki bir düzeltme için problem raporu açıp kurul kararı beklemektir.

Hangi verinin hangi kategoride yönetileceği yazılım seviyesine göre değişir. Yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC), yazılım konfigürasyon indeksi (Software Configuration Index, SCI) ve yazılım başarı özeti (Software Accomplishment Summary, SAS) gibi otoriteye sunulan veriler uygulandıkları her seviyede CC1'dir; geliştirme, doğrulama, konfigürasyon yönetimi ve kalite güvencesi planları (sırasıyla SDP, SVP, SCMP, SQAP) ise seviyeye göre kategori değiştirir. Standardın Ek A tablolarındaki dağılım şöyledir:

Kontrol kategorisi	Veri
Uygulandığı her seviyede CC1	PSAC, yazılım gereksinim verisi, kaynak kod, çalıştırılabilir nesne kodu, PDI dosyası, SCI, SAS
Seviye A, B ve C'de CC1; Seviye D'de CC2	Tasarım tanımı (design description); yazılım yaşam döngüsü ortam konfigürasyon indeksi (Software Life Cycle Environment Configuration Index, SECI)
Seviye A ve B'de CC1; Seviye C ve D'de CC2	SDP, SVP, SCMP, SQAP; doğrulama durumları ve prosedürleri
Seviye A ve B'de CC1; Seviye C'de CC2	Gereksinim, tasarım ve kodlama standartları
Hangi sürecin çıktısı olduğuna göre	İz verisi: geliştirme süreçlerinin çıktısı olarak uygulandığı her seviyede CC1; test tarafında Seviye A ve B'de CC1, Seviye C ve D'de CC2
Her seviyede CC2	Doğrulama sonuçları, problem raporları, konfigürasyon yönetimi kayıtları, kalite güvencesi kayıtları

Kaynak kod ile üç standart Seviye D'nin tablolarında çıktı olarak yer almaz; "uygulandığı her seviye" ifadesi bu yüzden kullanılmıştır. Tablodaki kategoriler asgaridir: proje bir veriyi standardın istediğinden daha sıkı kontrol etmeyi seçip bunu SCMP'ye yazabilir, daha gevşek edemez. İlk bakışta şaşırtan bir ayrıntı: problem raporlama CC1 veriye uygulanan bir faaliyettir, problem raporunun kendisi ise CC2 veridir — bir raporu düzeltmek için ikinci bir rapor açılmaz.

Konfigürasyon yönetimi iş ürünleri

Konfigürasyon yönetimi faaliyetlerinin çıktısı, sertifikasyon dosyasının önemli bir parçasını oluşturan somut iş ürünleridir. Denetimlerde en sık masaya gelen ürünler şunlardır:

- **Yazılım konfigürasyon indeksi (SCI):** Teslim edilen yazılım sürümünün "malzeme listesi"dir. Çalıştırılabilir nesne kodunu ve varsa PDI dosyalarını, bunları üreten kaynak kod sürümlerini, ilgili yaşam döngüsü verisini, arşiv ve teslim ortamını, derleme (build) ve yükleme talimatlarını ve çalıştırılabilir nesne kodunun bütünlük denetim değerlerini tanımlar. Bir SCI okunduğunda o sürümün tam olarak nelerden oluştuğu ve nasıl yeniden üretileceği anlaşılabilir. SCI, PSAC ve SAS ile birlikte otoriteye asgari olarak sunulan üç veriden biridir; diğer veriler talep edildiğinde erişime açılır.
- **Yazılım yaşam döngüsü ortam konfigürasyon indeksi (SECI):** Yazılımı üretmek ve doğrulamak için kullanılan ortamı tanımlar: derleyici ve bağlayıcı sürümleri, derleme seçenekleri, test araçları, kalifiye edilmiş araçlar ve donanım ortamı. SECI olmadan "aynı kodu beş yıl sonra yeniden derleyin" talebi karşılanamaz. SECI ayrı bir doküman olabileceği gibi SCI'nin bir bölümü olarak da verilebilir.
- **Problem raporları:** Her raporun tekil bir numarası, açık bir problem tanımı, etkilenen öge ve sürüm bilgisi, sınıflandırması, çözüm kararı ve kapanış kanıtı bulunur. DO-178C'nin rapordan beklediği çekirdek daha dardır: problemin görüldüğü ve düzeltilecek öge ya da süreç, emniyet ve işlev etkisini değerlendirmeye yetecek ayrıntıda bir tanım ve yapılan düzeltici faaliyet; sınıflandırma ve kapanış kanıtı gibi alanları proje ekler. Raporun yaşam döngüsü ve sınıflandırma eksenleri [9. Yazılım Doğrulama](#) bölümünde anlatılır; bu bölüm ayrı bir şema kurmaz. Sürüm tesliminde açık kalan raporlar FAA AC 20-189 ve EASA AMC 20-189'un beklediği sınıflarla (önemli, işlevsel, süreç, yaşam döngüsü verisi) sınıflandırılır; özetleri, işlevsel kısıtlar ve açık bırakma gerekçesi SAS'ta sunulur (bkz. [12. Sertifikasyon İrtibatı](#)). Birçok kuruluş aynı listeyi SCI'ye ya da sürüm notuna da koyar; bu durumda listeler elle değil, aynı tarihte tek kaynaktan üretilmelidir.
- **Konfigürasyon yönetimi kayıtları:** Temel çizgi kayıtları, değişiklik kontrol kurulu karar tutanakları, sürüm teslim kayıtları, arşiv doğrulama kayıtları gibi faaliyetlerin gerçekten yapıldığını gösteren kanıtlardır.

Problem raporu ile kalite güvencesinin açtığı uygunsuzluk kaydı arasındaki sınır da planlarda yazılı olmalıdır. Plan ve standartlardan süreç sapması problem raporlamanın kapsamındadır; standart, süreç sorunlarıyla ürün sorunlarının ayrı sistemlerde kaydedilmesine de izin verir. Projeler bunu iki biçimde düzenler. Bazıları süreç uygunsuzluklarını da aynı problem raporlama sisteminde, türü ayrıca işaretlenmiş kayıtlar olarak tutar. Bazıları uygunsuzlukları ayrı bir kalite güvencesi kaydı olarak izler ve kendi takibiyle kapatır; bu durumda da uygunsuzluğun giderilmesi temel çizgideki bir veriyi değiştirmeyi gerektiriyorsa o değişiklik problem raporu ve değişiklik kontrolü üzerinden yürür. Hangisi seçilirse seçilsin SCMP ile SQAP aynı akışı anlatmalıdır (bkz. [11. Yazılım Kalite Güvencesi](#)).

Aşağıdaki tablo, bu ürünlerin cevapladığı temel soruları ve otoriteye nasıl ulaştıklarını özetler:

İş ürünü	Cevapladığı soru	Otoriteye sunum
SCI	Bu sürümde tam olarak ne var, nasıl yeniden üretilir?	Sunulur
SECI	Bu sürüm hangi araçlarla ve hangi ortamda üretildi?	SCI'nin parçasıysa onunla sunulur; ayrıysa talep üzerine gösterilir
Problem raporları	Bilinen sorunlar neler, hangileri açık?	Talep üzerine gösterilir; açık raporların özeti SAS'tadır
Konfigürasyon yönetimi kayıtları	Süreç gerçekten işletildi mi?	Talep üzerine, çoğunlukla denetimde gösterilir

Deneyimin gösterdiği önemli bir nokta: SCI ve SECI, teslim gününe bırakılırsa eksik ve hatalı çıkar. En sağlıklı yaklaşım, bu indeksleri mümkün olduğunca derleme ve sürüm altyapısından otomatik üretmek, elle tutulan kısmı en aza indirmektir.

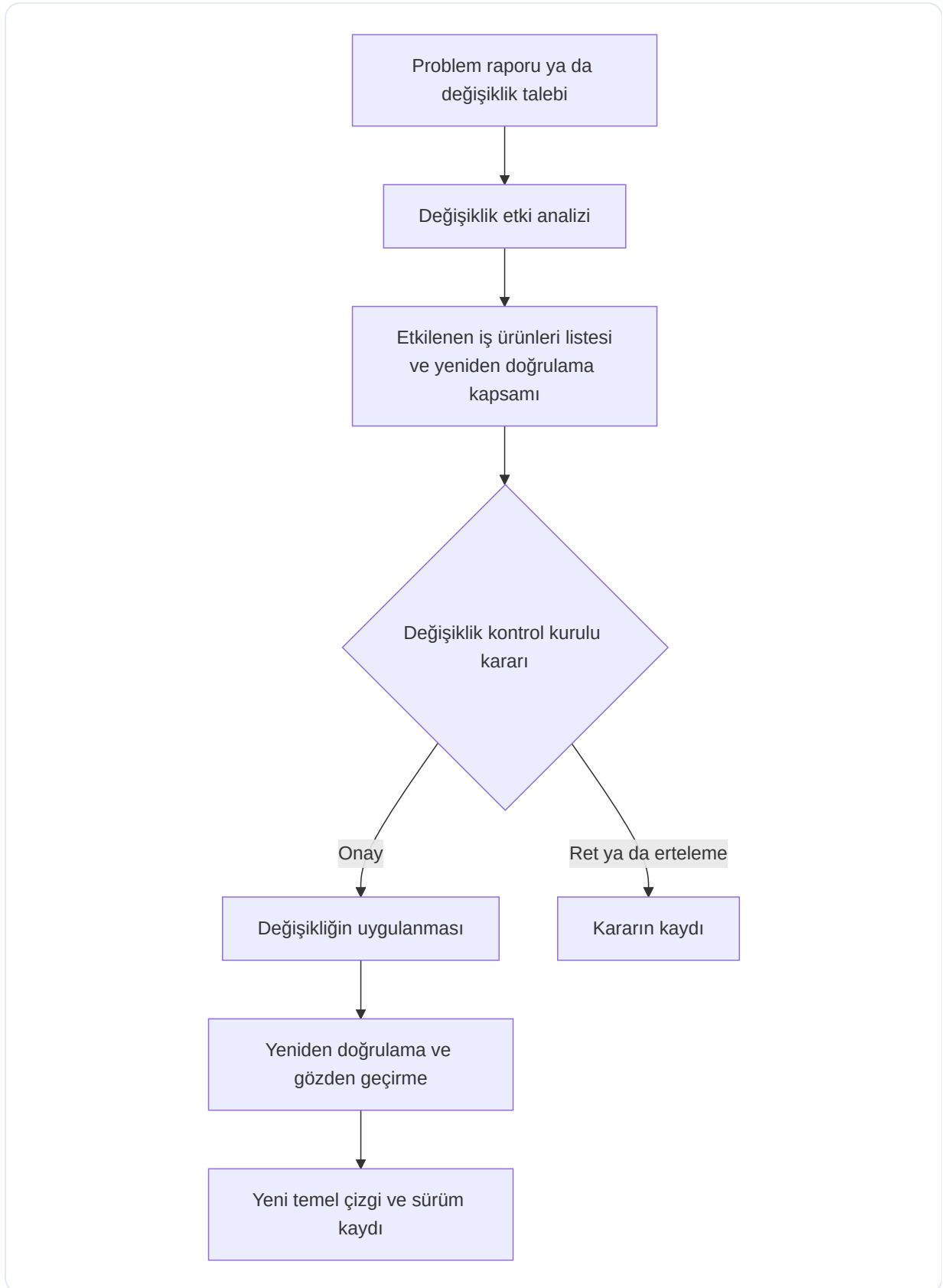
Değişiklik etki analizi

Değişiklik etki analizi (change impact analysis), bir değişikliğin dokunduğu her şeyi değişiklik uygulanmadan **önce** sistematik biçimde ortaya çıkarma disiplini. Sık yapılan hata, analizi "hangi dosyalar değişecek" sorusuna indirgemektir; oysa asıl soru "hangi kanıtlar geçerliliğini yitirecek" sorusudur.

DO-178C'de bu değerlendirme değişiklik gözden geçirmesinin işidir. İyi bir etki analizi en azından şu eksenleri tarar:

- **Sistem ve emniyet eksen:** Problem ya da değişiklik sistem gereksinimlerini etkiliyor mu? Etkiliyorsa sistem süreçlerine ve sistem emniyet değerlendirmesine bildirilmiş, gelen yanıt değerlendirilmiş mi?
- **Gereksinim eksen:** Değişiklik hangi yüksek ve düşük seviyeli gereksinimleri etkiliyor? İz verisi bu taramanın ana aracıdır.
- **Tasarım ve mimari eksen:** Yazılım mimarisi, arayüzler, zamanlama bütçeleri veya bellek kullanımı etkileniyor mu? Yazılım bölümlenmesi varsa bölümler arası izolasyon varsayımları bozuluyor mu?
- **Kod eksen:** Değişen fonksiyonlar, bunları çağıran ve bunlardan etkilenen kod bölgeleri, paylaşılan veri yapıları ve derleme seçenekleri.
- **Doğrulama eksen:** Hangi testler yeniden koşulmalı? Yalnızca değişen gereksinimin testleri mi, yoksa regresyon kapsamı daha mı geniş? Yapısal kapsam analizi sonuçları hâlâ geçerli mi?
- **Sertifikasyon eksen:** SCI, SECI, hedeflerle kanıtları eşleyen uyum matrisi (compliance matrix) ve otoriteye sunulan veriler güncellenecek mi? Değişiklik, daha önce verilen bir sapma (deviation) gerekçesini etkiliyor mu?

Tipik akış şu şekildedir:



Analizin çıktısı, değişiklik kontrol kurulunun karar verebileceği somut bir listedir: etkilenen iş ürünleri, güncellenecek dokümanlar, yeniden koşulacak testler ve tahmini iş yükü. Sertifikalı bir yazılımda sonradan yapılan değişikliklerde bu analiz ayrıca otorite ile paylaşılan resmî bir veri

hâline gelir ve yeniden doğrulama kapsamı ona dayanarak savunulur. Tip tasarımı değişikliğinin küçük (minor) ya da büyük (major) sayılması ise bir DO-178C kararı değildir; ürün düzeyinde, sertifikasyon kuralları çerçevesinde verilir ve yazılım değişiklik etki analizi bu karara girdi sağlar. Önceden geliştirilmiş yazılımın değiştirilerek yeniden kullanılması [24. Yazılım Yeniden Kullanımı](#) bölümünde ele alınır.

Pratik bir uyarı: etki analizi iz verisinin kalitesi kadar iyidir. Gereksinim, kod ve test arasındaki izler eksik veya güncel değilse analiz kâğıt üzerinde tamam görünse bile gerçek etkiyi kaçırır. Gereksinim tarafındaki işleyiş [6. Yazılım Gereksinimleri](#) bölümünde anlatılır.

Baştan sona bir örnek: tek bir gereksinim değişikliği

Kurgusal bir hava verisi bilgisayarını ele alalım. Yazılımın 2.1.0 sürümü temel çizgiye alınmış ve testleri koşulmuştur. Sistem ekibi, pitot ısıtıcısı arızasının mürettebata bildirilme süresini 2 saniyeden 1 saniyeye indiren bir sistem gereksinimi değişikliği gönderir. Zincir şöyle işler:

- 1. Kayıt.** Değişiklik talebi açılır; etkilenen temel çizgi (2.1.0) ve değişikliğin kaynağı (sistem gereksinimi değişikliği) yazılır.
- 2. Etki analizi.** İz verisi sorgulanır: yüksek seviyeli gereksinim HLR-042, ona bağlı iki düşük seviyeli gereksinim, bir kaynak dosya, üç test durumu ve bir test prosedürü listelenir. Mimari ve arayüzler etkilenmez. Zamanlama bütçesi için gerekçe yazılır: izleme görevinin periyodu değişmemekte, yalnızca sayaç eşiği küçülmektedir. Geçersiz kalacak kanıt da listelenir: HLR-042'nin test sonuçları, değişen kaynak dosyanın gözden geçirme kaydı ve yapısal kapsam verisi.
- 3. Karar.** Değişiklik kontrol kurulu talebi onaylar, hedef sürümü (2.2.0) ve yeniden doğrulama kapsamını karara bağlar: değişen gereksinimin testleri ile aynı görevdeki diğer izleme işlevlerinin regresyon testleri.
- 4. Uygulama ve yeniden doğrulama.** Her öge değiştirilir, gözden geçirilir ve yeni sürümüyle kaydedilir; testler 2.2.0 derlemesine karşı koşular.
- 5. Kapanış.** Etki analizindeki her kalemin kapandığı, yani onaylanan değişikliğin eksiksiz uygulandığı teyit edilir; yeni temel çizgi kurulur, talep kapanış kanıtlarıyla birlikte kapatılır.

Zincirin sonunda durum muhasebesi şu tabloyu verebilmelidir:

Konfigürasyon ögesi	Önceki sürüm	Yeni sürüm	Açıklama
HLR-042 (yüksek seviyeli gereksinim)	B	C	Bildirim süresi 2 s yerine 1 s
LLR-118 (düşük seviyeli gereksinim)	D	E	Sayaç eşiği yeniden hesaplandı

Konfigürasyon ögesi	Önceki sürüm	Yeni sürüm	Açıklama
LLR-119 (düşük seviyeli gereksinim)	B	B	Gözden geçirildi; değişiklik gerekmedi
pitot_izleme.c	1.7	1.8	Adlandırılmış sabit güncellendi
Test durumları TC-042-01...03	C	D	Sınır değerler ve beklenen sonuçlar
Test prosedürü TP-07	2.3	2.4	Yeni test durumlarına uyarlandı
İz verisi	2.1.0	2.2.0	Bağlar yeni sürümleri gösteriyor
Çalıştırılabilir nesne kodu	2.1.0	2.2.0	Yeniden derlendi; yeni bütünlük denetim değeri
SCI	2.1.0	2.2.0	Yukarıdaki sürümleri ve talebi listeliyor
SECI	5	5	Ortam değişmedi; yeni SCI aynı SECI'ye atıf yapıyor

Tabloda iki satır özellikle öğreticidir. LLR-119 değişmemiştir ama etki analizinde yer aldığı için gözden geçirildiği kayıtlıdır; "bakıldı ve değişmedi" ile "hiç bakılmadı" arasındaki fark denetimde bu kayıtlarla gösterilir. SECI de değişmemiştir; değişmediğinin bilinmesi, ortamın kontrol altında olmasının sonucudur. Bir denetçi 2.2.0 sürümünün çalıştırılabilir nesne kodundan başlayıp SCI üzerinden kaynak dosyaya, oradan gereksinime, test sonucuna ve değişiklik talebine kopmadan ulaşabiliyorsa konfigürasyon yönetimi işini yapmıştır.

Sık düşülen tuzaklar

Konfigürasyon yönetimi hataları genellikle kötü niyetten değil, "sonra düzeltiriz" yaklaşımından doğar. Sahada en sık karşılaşılan tuzaklar şunlardır:

- **Konfigürasyon kontrolünü geç kurmak.** Ekip, "önce prototipi bitirelim, kontrolü resmî faza girince başlatırız" der. Sonuç: hangi gereksinim sürümüne göre kod yazıldığı belirsizleşir ve geriye dönük temel çizgi kurmak, baştan kurmaktan çok daha pahalıya gelir. Sürüm kontrolü ilk günden işlemeli, resmî değişiklik kontrolü ise gözden geçirilmiş ilk iş ürününün temel çizgisiyle devreye girmelidir.
- **Kontrolsüz geliştirme ve doğrulama ortamları.** Derleyici sürümü geliştiricinin makinesine göre değişiyorsa aynı kaynak koddan farklı nesne kodlar üretiliyor demektir. Test bilgisayarındaki araç güncellemesi kayıt altına alınmamışsa geçmiş test sonuçları savunulamaz hâle gelir. Ortam da bir konfigürasyon ögesidir; SECI bu yüzden vardır.
- **Eksik veya gayriresmî problem raporu.** "Küçük hataları e-postayla hallettik" cümlesi denetimde ciddi bir bulguya dönüşür. Problem raporu açılmayan hata, etki analizi yapılmamış, kararı kaydedilmemiş hata demektir. Küçük görünen bir sorunun başka bir bölgedeki etkisi ancak kayıt ve analizle görülür.

- **Değişikliği koda uygulayıp kanıtı güncellememek.** Kod değişir, testler koşulur ama tasarım verisi ve iz verisi eski hâlinde kalır. Doküman ile kod arasında açılan makas, ilerleyen aşamalarda büyük bir uyumsuzluk yığınınına dönüşür.
- **Temel çizgiyi "her şey bitince" almak.** Tek ve dev bir temel çizgi, ara aşamalardaki doğrulama sonuçlarını hangi içeriğe bağlayacağınızı belirsizleştirir. Ara temel çizgiler (gereksinim, tasarım, test hazırlık) küçük ama düzenli adımlarla alınmalıdır.
- **Sürüm aracına aşırı güvenmek.** Modern sürüm kontrol araçları çok şeyi otomatik yapar; ancak araç, değişiklik kontrol kurulu kararını, etki analizini veya CC1/CC2 ayrımını kendiliğinden üretmez. Araç bir altyapıdır, süreç değildir.
- **Açık problem raporlarını teslimde görünmez kılmak.** Sürüm tesliminde açık raporları SAS'ta listeleyp değerlendirmeden "nasılsa kapatacağız" demek, hem otorite nezdinde güveni zedeler hem de bilinen bir sorunun uçuşa etkisinin değerlendirilmeden kalmasına yol açar.

Bu tuzakların ortak ilacı aynıdır: konfigürasyon yönetimini bürokratik bir yük olarak değil, doğrulama kanıtının geçerliliğini koruyan mühendislik altyapısı olarak görmek ve projenin ilk gününden itibaren işletmek. Denetime hazırlanırken bu bölümdeki beklentilerin kontrol listesi karşılıkları [SW SOI-2](#) ve [SW SOI-4](#) sayfalarındadır.

Bu bölümden akılda kalması gerekenler

- Konfigürasyon yönetimi dosya sürümlerini değil, doğrulama kanıtının geçerliliğini korur; hedef tablosundaki altı hedefi Seviye A'dan D'ye kadar geçerlidir ve bağımsızlık aranmaz.
- Kredi alınacak öğeler temel çizgiye alınır ve ürün temel çizgisi SCI'de tanımlanır. Temel çizgi değiştirilmez, yenisi türetilir; her temel çizgi öncekine izlenebilir.
- Konfigürasyon tanımlama ve sürüm kontrolü ilk günden işler; resmî değişiklik kontrolü ve problem raporlama ilk temel çizgiyle başlar.
- CC1 tam kontroldür; CC2'de temel çizgi, problem raporlama ve değişikliğin resmî onayı aranmaz ama tanımlama, bütünlük, koruma, geri getirme ve saklama yine beklenir. Hangi verinin hangi kategoride olduğunu seviyeye göre standardın Ek A tabloları belirler.
- SCI sürümün içeriğini, SECI sürümü üreten ortamı tanımlar; ikisi olmadan sürüm yeniden üretilemez. SCI, PSAC ve SAS ile birlikte otoriteye sunulur.
- Açık problem raporlarının değerlendirilmesi SAS'ta yer alır; başka belgelerdeki listeler aynı kaynaktan üretilir.
- Değişiklik etki analizi "hangi dosyalar değişecek" değil, "hangi kanıtlar geçersiz kalacak" sorusunu cevaplar ve iz verisinin kalitesine dayanır.

11. Yazılım Kalite Güvencesi

Kalite güvencesi (quality assurance, QA), DO-178C'de yazılım yaşam döngüsü süreçlerinin ve çıktılarının onaylı planlara ve standartlara uygun olduğuna dair bağımsız güvence sağlayan süreçtir. Bu bölüm kalite güvencesinin hedeflerini, bağımsızlığını, denetim faaliyetlerini, uygunsuzluk yönetimini ve teslim öncesindeki yazılım uygunluk gözden geçirmesini anlatır.

Kalite güvencesi yazılım üretmez, yazılımı doğrulamaz da. Ürettiği şey güvendir: planda yazanın gerçekten yapıldığına, yapılanın kaydının durduğuna ve sapmaların görülüp kapatıldığına dair güven. Sertifikasyon kanıtı dayandığı için bu küçük bir iş değildir. Bugün atlanan bir gözden geçirme (review) kaydı ya da temel çizgiye (baseline) alınmadan koşulan bir test, aylar sonra kapatılması pahalı bir kanıt boşluğuna dönüşür. Kalite güvencesi; doğrulama, konfigürasyon yönetimi (configuration management) ve sertifikasyon irtibatıyla birlikte bütünleyici süreçlerden (integral processes) biridir: proje sonunda atılan bir adım değil, planlamadan teslim kadar süren bir gözetimdir.

Kalite güvencesi neyi güvence altına alır?

Kalite güvencesini doğrulamayla karıştırmak kolaydır, çünkü ikisi de "kontrol eden" taraftadır. Ayrım, sordukları soruda ortaya çıkar:

Süreç	Cevapladığı soru	Nasıl cevaplar
Doğrulama (verification)	Ürün gereksinimlerini karşılıyor mu, hata içeriyor mu?	Gereksinim, tasarım, kod ve testlerin teknik içeriğini gözden geçirme, analiz ve testle değerlendirir
Kalite güvencesi	Süreçler ve çıktıları onaylı plan ve standartlara uygun mu; sapmalar kaydedilip kapatılıyor mu?	Faaliyetleri ve kayıtlarını örnekleme yoluyla denetler, eksikleri izler
Sertifikasyon irtibatı ve sertifikasyon otoritesi (certification authority)	Sunulan uyum kanıtı yeterli ve inandırıcı mı?	Planları, özet verileri ve denetim bulgularını değerlendirir

Bir kod gözden geçirmesi üzerinden düşünelim. Kodun düşük seviyeli gereksinimi doğru gerçekleyp gerçeklemediğine karar vermek doğrulamanın işidir. Kalite güvencesi aynı kararı ikinci kez vermez; gözden geçirmenin planda tarif edildiği gibi yapıp yapılmadığına bakar: doğru sürüm üzerinde mi yapıldı, kontrol listesi kullanıldı mı, bağımsızlık gereken yerde sağlandı mı, bulgular kapanmadan kayıt kapatıldı mı?

Bu, kalite güvencesinin yalnızca sürece baktığı anlamına gelmez. Değerlendirme süreçlerin çıktıları da kapsar: bir tasarım verisinin tasarım standardına uyup uymadığı, bir test sonucunun kayıtlı ortamda üretilip üretilmediği, teslim edilecek sürümün verisinin eksiksiz olup olmadığı

kalite güvencesinin sorularıdır. Fark, ürünün teknik doğruluğunu yeniden kanıtlamaya çalışmaması; onaylı plan ve standartlara uygunluğu ve kanıtın bütünlüğünü sorgulamasıdır.

Kalite güvencesinin hedefleri ve seviyeye göre beklenti

DO-178C'nin hedef tablolarından biri kalite güvencesi sürecine ayrılmıştır (Tablo A-9) ve beş hedef (objective) içerir. Bu hedeflerin ortak kalıbı önemlidir: kalite güvencesinden işi bizzat yapması değil, işin yapıldığına dair güvence elde etmesi beklenir.

Kalite güvencesi hedefi	A	B	C	D
Planlar ve standartlar geliştirilmiş; DO-178C ile uyumu ve kendi aralarındaki tutarlılığı gözden geçirilmiştir	Evet	Evet	Evet	—
Yazılım yaşam döngüsü süreçleri onaylı planlara uygun yürümektedir	Evet	Evet	Evet	Evet
Yazılım yaşam döngüsü süreçleri onaylı standartlara uygun yürümektedir	Evet	Evet	Evet	—
Süreçler arası geçiş kriterleri (transition criteria) sağlanmıştır	Evet	Evet	Evet	—
Yazılım uygunluk gözden geçirmesi (software conformity review) yapılmıştır	Evet	Evet	Evet	Evet

Standardın gövde metni kalite güvencesi için dört hedef sayar; tablo, plan ve standartlara uygunluğu iki ayrı satıra böldüğü için beş hedef içerir. Ayrımın nedeni seviyedir: planlara uygunluk Seviye D'de de aranır, standartlara uygunluk aranmaz.

Tablodan üç sonuç çıkar. Birincisi, uygulandığı her yerde bu hedefler bağımsızlıkla karşılanır; Seviye C'deki beş, Seviye D'deki iki bağımsızlıklı hedefin tamamı kalite güvencesi hedefleridir. İkincisi, yazılım seviyesi düştükçe kalite güvencesi ortadan kalkmaz, daralır: Seviye D'de planlara uygunluk ve uygunluk gözden geçirmesi kalır. Otoritenin teyit ettiği Seviye E'de DO-178C rehberliği uygulanmadığı için kalite güvencesi hedefi de yoktur. Üçüncüsü, ilk hedef kalite güvencesini planlama aşamasına bağlar: planların ve standartların gözden geçirildiğine dair güvence, ilk satır kod yazılmadan önce elde edilir. Bu hedef DO-178B'nin hedef tablosunda yer almıyordu; tabloya DO-178C ile girdi.

Bağımsızlık ve yetki

DO-178C'de bağımsızlık (independence) iki ayrı anlamda kullanılır ve ikisi sık karıştırılır. Doğrulamada bağımsızlık, bir iş ürününü doğrulayanın onu geliştiren kişi olmamasıdır; ayrıntısı [9. Yazılım Doğrulama](#) bölümündedir. Kalite güvencesinde bağımsızlık buna bir öge daha ekler: düzeltici faaliyetin (corrective action) yapılmasını sağlama yetkisi. Bulgu yazabilen ama o bulguyu kapattıramayan bir kalite güvencesi, tanım gereği bağımsız sayılmaz.

	Doğrulama bağımsızlığı	Kalite güvencesi bağımsızlığı
Anlamı	Ürünü doğrulayan, onu geliştiren kişi değildir; insan faaliyetine denk güvence veren bir araç da kullanılabilir	Geliştirme faaliyetlerinden ayrı durmanın yanında düzeltici faaliyeti sağlayacak yetkiye sahip olmak
Nerede aranır	Hedef tablolarında bağımsızlıkla işaretli doğrulama hedeflerinde; fiilen Seviye A ve B'de	Seviyeye uygulanan bütün kalite güvencesi hedeflerinde; Seviye A'dan D'ye
Nasıl sağlanır	Aynı ekipten başka bir mühendis yeterlidir; ayrı organizasyon birimi şart değildir	Yaygın yol, proje yönetiminden ayrı bir raporlama hattıdır
Tipik zaafı	Yazarın kendi işini gözden geçirmesi	Takvimden sorumlu yöneticiye bağlı olmak; bulguyu kapattırarak yetkinin bulunmaması

Ayrı raporlama hattı, yetkiyi sağlamanın alışılmış yoludur; standardın dayattığı bir organizasyon şeması değildir. Aranılan sonuçtur: kalite güvencesi, takvim baskısı altında "bu seferlik göz yumalım" talebine hayır diyebilmeli ve çözülmeyen bir bulguyu proje yöneticisinin üstüne taşıyabilmelidir. Küçük kuruluşlarda kalite güvencesi tam zamanlı ayrı bir ekip olmayabilir; o zaman da kişinin kendi ürettiği işi denetlememesi ve yetkisinin yazılı olması gerekir. Yetkinin kimde olduğu, kime raporlandığı ve anlaşmazlığın nasıl çözüleceği yazılım kalite güvencesi planında tanımlanır.

Kalite güvencesi planı ve kayıtları

Yazılım kalite güvencesi planı (Software Quality Assurance Plan, SQAP), bu sürecin sözleşmesidir. Beş plan içindeki yeri [5. Yazılım Planlama](#) bölümünde anlatılır; burada içeriğine kalite güvencesinin gözünden bakıyoruz. İyi bir SQAP şu soruları cevaplar:

- **Kim, hangi yetkiyle?** Kalite güvencesinin sorumluluğu, bağımsızlığı, raporlama hattı ve onay yetkisi.
- **Ne yapılacak, nasıl?** Her yaşam döngüsü süreci için kullanılacak yöntemler: denetim, gözden geçirmelere katılım, tanıklık, izleme ve raporlama.
- **Ne zaman?** Faaliyetlerin yaşam döngüsündeki yeri: hangi temel çizgiden önce, hangi geçişte, hangi sıklıkta.
- **Sapma bulunursa ne olur?** Uygunsuzluğun kaydı, sınıflandırılması, izlenmesi, kapanış ölçütü ve üst yönetime taşıma yolu; problem raporlama (problem reporting) akışıyla ilişkisi.
- **Teslimden önce ne yapılır?** Yazılım uygunluk gözden geçirmesinin yöntemi.
- **Tedarikçiler nasıl gözetilir?** Alt yüklenicilerin süreç ve çıktılarının uygunluğunun nasıl güvence altına alınacağı.
- **Hangi kayıtlar tutulur?** Üretilecek kalite güvencesi kayıtlarının tanımı.

Otoritenin planlama denetiminde, yani ilk katılım aşaması (Stage of Involvement, SOI) denetiminde SQAP'ta baktığı noktalar [SW SOI-1](#) kontrol listesinde. Tipik zayıflık ölçülemeyen taahhüttür: "süreçler düzenli olarak denetlenir" cümlesinin ne zaman yerine getirildiği

anlaşılamaz.

Kalite güvencesinin kendi çıktısı **kalite güvencesi kayıtlarıdır**. Standart bunlara bir biçim dayatmaz, ama iki şeyin kayıtlarda bulunmasını açıkça bekler: denetim sonuçları ve sertifikasyona sunulan her sürüm için uygunluk gözden geçirmesinin tamamlandığının kanıtı. Onaylanmış süreç sapmalarının kayıtları, gözden geçirme raporları ve toplantı tutanakları da çoğunlukla buradadır. Bu kayıtlar da yazılım yaşam döngüsü verisidir (software life cycle data) ve ikinci kontrol kategorisiyle (control category 2, CC2) yönetilir; kontrol kategorileri [10. Yazılım Konfigürasyon Yönetimi](#) bölümünde açıklanır. İşe yarar bir denetim kaydı en azından şunları söyler: neyin, hangi temel çizgi üzerinde, hangi ölçüte göre denetlendiği; hangi kayıtların örnek olarak seçildiği; ne bulunduğu ve hangi uygunsuzlukların açıldığı. "Denetim yapıldı, sorun yok" yazan bir kayıt, yapılan işi başkasının yeniden izlemesine imkân vermez.

Denetim ve gözlem

Kalite güvencesi masa başında belge saymakla yürümez; işin yapıldığı yere gider. Temel aracı denetimdir (audit), ama tek aracı değildir:

Faaliyet	Neye bakar	Örnek soru
Süreç denetimi	Bir sürecin planda tarif edildiği gibi işletildiğine	"Plan her değişiklik için etki analizi istiyor; son beş değişikliğin analizini görebilir miyim?"
Veri denetimi	Bir çıktının standarda uyduğuna ve kontrol kategorisine göre yönetildiğine	"Bu tasarım verisi tasarım standardının zorunlu tuttuğu kısımları içeriyor mu, hangi temel çizgide?"
Gözden geçirmelere katılım	Gözden geçirmenin tanımlı yöntemle ve doğru sürüm üzerinde yapıldığına	"Kontrol listesi kullanıldı mı; bulgular kapanmadan kayıt kapatılmış mı?"
Tanıklık (witnessing)	Kredi amaçlı test koşusunun, derlemenin ya da yüklemenin prosedüre göre ve kayıtlı ortamda yapıldığına	"Test edilen imajın (image) kimliği temel çizgideki imajla aynı mı?"
Tedarikçi denetimi	Tedarikçinin süreç ve çıktılarının onaylı plan ve standartlara uyduğuna	"Tedarikçideki gözden geçirme kayıtları bizim planımızın istediği içeriği taşıyor mu?"

Kalite güvencesinin güvence altına alacağı, projeden projeye değişmeyen konuları DO-178C de sayar. Bir kısmı planlarla ilgilidir: planlar ekibin elindedir ve geliştirme ortamı planlarda tanımlandığı gibi kurulmuştur. Bir kısmı sapmalarla ilgilidir: plan ve standartlardan sapmalar fark edilir, kaydedilir, değerlendirilir ve çözülene kadar izlenir; onaylanan sapmalar da kayıtlıdır. Geri kalanı süreçler arasındaki bağlardır: problem raporlama ve düzeltici faaliyet akışı yazılım konfigürasyon yönetimi planına (Software Configuration Management Plan, SCMP) göre işler, sistem süreçlerinden ve sistem emniyet değerlendirmesinden gelen girdiler ele alınmıştır ve veri kontrol kategorisine uygun yönetilir.

Örnekleme ve ölçekleme

Kalite güvencesi her kaydı okuyamaz, okuması da beklenmez; örnekleme (sampling) yapar. Örneklemenin nasıl seçildiği denetimin değerini belirler. Rastgele beş kayıt yerine riske göre seçim daha çok şey gösterir: ekibe yeni katılan birinin işi, yeni devreye alınan bir araç ya da süreç, son anda değişen öğeler, daha önce uygunsuzluk çıkmış alanlar ve emniyet açısından en kritik işlevler. Seçilen örneklem denetim kaydına yazılır; örneklem bir sorun gösterdiğinde kapsam genişletilir, çünkü üç kayıttan birinde görülen eksik çoğu zaman tek bir kişinin dalgınlığı değil, sürecin bir açığıdır.

Denetim sıklığı ve örneklem büyüklüğü SQAP'ta tanımlanır ve projeye göre ölçeklenir: yazılım seviyesi, ekibin büyüklüğü ve deneyimi, tedarikçi sayısı ve önceki denetimlerin sonuçları belirleyicidir. Seviye D'de iki hedefle sınırlı bir kalite güvencesi ile Seviye A'da yaşam döngüsünün her sürecine yayılan bir kalite güvencesi aynı yoğunlukta olmaz.

Geçiş kriterlerinin denetimi

Planlar, bir süreçten diğerine hangi koşulda geçileceğini geçiş kriterleriyle tanımlar. Kalite güvencesinin ayrı bir hedefi, bu kriterlerin yalnızca yazılmış değil, proje boyunca sağlanmış olduğuna dair güvence elde etmektir. Uygulamada bu, kriter listesini kayıtlarla karşılaştırmak demektir: "gereksinimler gözden geçirilmiş, açık büyük bulgu yok, temel çizgiye alınmış" diyen bir kriter için gözden geçirme kaydına, bulgu listesine ve temel çizgi kaydına bakılır.

Kriter sağlanmadan sonraki faaliyete başlanmışsa bu bir uygunsuzluktur. Aynı kriter tekrar tekrar aşıyorsa sorun ekipte değil kriterde olabilir; o zaman doğru çözüm kriteri sessizce yok saymak değil, planı değişiklik kontrolünden geçirerek güncellemektir. Artımlı çalışan projelerde denetim de artım ya da işlev bazında yapılır. Her geçişin mi yoksa geçişlerden alınan bir örneklemin mi denetleneceğini SQAP belirler. Süreçler arası geçişler için örnek kriterler ve kanıtları [Ek A: Örnek Geçiş Kriterleri](#) sayfasındadır.

Tedarikçi gözetimi

Yazılım yaşam döngüsü faaliyetlerinin bir kısmı dışarıda yürüyorsa kalite güvencesinin alanı da oraya uzanır. DO-178C, tedarikçilerin süreç ve çıktılarının onaylı plan ve standartlara uyduğuna dair güvence elde etmeyi kalite güvencesinin faaliyetleri arasında sayar; sertifikasyon sorumluluğu işi devredenden ayrılmaz. Tedarikçinin kendi kalite güvencesi bulursa bile ana yüklenici onun kayıtlarını örnekleyerek denetler ve bu gözetimin kayıtlarını tutar. Denetim hakkının sözleşmeye nasıl yazılacağı ve gözetimin nasıl kurulacağı [26. Yazılım Yaşam Döngüsü Faaliyetlerinde Dış Kaynak Kullanımı](#) bölümünde ele alınır.

Yaşam döngüsü boyunca kalite güvencesi

Faaliyetler yaşam döngüsüne şöyle yayılır:

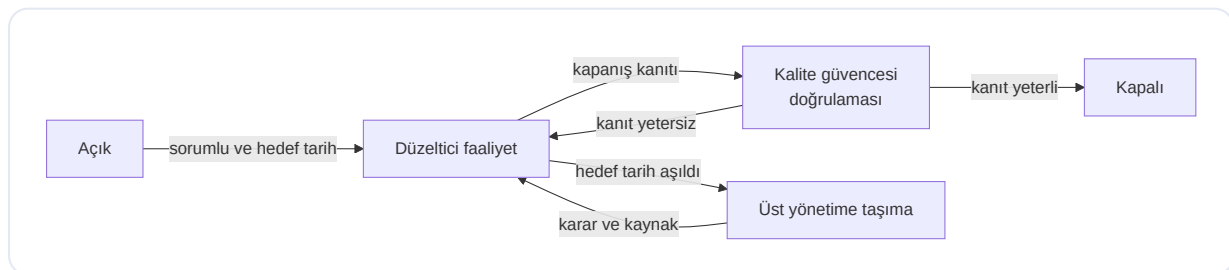
Aşama	Kalite güvencesinin yaptığı	Bıraktığı kayıt
Planlama	SQAP'ın hazırlanması; planların ve standartların uyum ve tutarlılık açısından gözden geçirildiğine dair güvence	Plan gözden geçirmesi kayıtları, denetim raporu
Gereksinim, tasarım, kodlama, entegrasyon	Süreç ve veri denetimleri, gözden geçirmelere katılım, geçiş kriterlerinin denetimi	Denetim raporları, uygunsuzluk kayıtları
Doğrulama	Kredi amaçlı test koşullarına tanıklık; test ortamının ve problem raporu akışının denetimi	Tanıklık kayıtları, denetim raporları
Teslim öncesi	Yazılım uygunluk gözden geçirmesi; açık uygunsuzlukların kapanışı	Uygunluk gözden geçirmesi kaydı
Bütün aşamalar	Uygunsuzlukların izlenmesi, tedarikçi gözetimi, durumun yönetime raporlanması	Uygunsuzluk durum raporları

Uygunsuzluk yönetimi

Uygunsuzluk (nonconformance), onaylı bir plana ya da standarda aykırı yürütülen bir faaliyet ya da bir süreç çıktısında bu belgelere göre görülen eksikliklerdir. DO-178C aynı durumu plan ve standartlardan sapma (deviation) olarak anar; kalite güvencesinden beklenen, sapmaların fark edilip kayda geçtiğine, etkisinin değerlendirildiğine ve çözümlene kadar izlendiğine dair güvence elde etmesidir. Uygunsuzluğun görülmesi başarısızlık değil, işleyen bir sürecin işaretidir; asıl sorun görülmeyen ya da görülüp kayda geçmeyen sapmadır.

İyi bir uygunsuzluk kaydı dört şeyi açık yazar: neyin gözlendiği ve hangi plan ya da standart maddesine aykırı olduğu; etkisinin ne olduğu ve nereye kadar uzandığı; sorumlusu ve hedef tarihi; kapanış için hangi kanıtın isteneceği. Etki sorusu en çok atlanandır. Bir süreç sapması çoğu zaman bir kanıtı da geçersiz kılar: temel çizgiye alınmamış kod üzerinde koşulan test yalnızca bir süreç hatası değildir, o testin sonucundan kredi alınamayacağı anlamına da gelir.

Düzeltilici faaliyet de iki katmanlıdır. İlk katman görülen örneği düzeltir; ikinci katman kök nedeni (root cause) giderir ki aynı sapma başka bir kayıta yeniden ortaya çıkmasın. Kalite güvencesi, kaydı ancak kapanış kanıtını gördükten sonra kapatır. Hedef tarihi aşan ya da üzerinde anlaşılamayan kayıt üst yönetime taşınır; bağımsızlık tanımındaki yetki tam burada işe yarar.



Uygunsuzluk ile problem raporu

DO-178C'de problem raporu yalnızca yazılım hatasının kaydı değildir. Problem raporlama üç tür sorunu kapsar: yazılımın anormal davranışı, yaşam döngüsü verisindeki kusur ve plan ya da standartlara süreç uyumsuzluğu. Kalite güvencesinin bulunduğu uygunsuzluk üçüncü türe girer; kayıt hangi sistemde tutulursa tutulsun, problem raporlamanın süreç tarafıdır.

	Ürün ya da veri kusuru	Süreç uygunsuzluğu
Örnek	Test başarısız olur; tasarım ile gereksinim çelişir	Test, temel çizgiye alınmamış kod üzerinde koşulmuştur; gözden geçirme kontrol listesi kullanılmamıştır
Çoğunlukla kim bulur	Doğrulama, entegrasyon, saha	Kalite güvencesi denetimi; ekipten herkes de bildirebilir
Düzeltilme neyi değiştirir	Verinin kendisini, değişiklik kontrolünden geçerek	Çalışma biçimini; gerekirse etkilenen faaliyet tekrarlanır
Kapanışı kim doğrular	Doğrulama: düzeltme yeniden doğrulanır	Kalite güvencesi: kapanış kanıtı ve tekrarın önleniği görülür

Projeler bunu iki biçimde düzenler. Bazıları tek bir problem raporlama sistemi kullanır ve kaydın türünü bir alanla ayırır. Bazıları kalite güvencesi uygunsuzluklarını ayrı bir kayıt listesinde, kalite güvencesi kaydı olarak tutar; sapma kontrol altındaki bir veride değişiklik gerektirdiğinde kayda bağlı bir problem raporu açılır. Hangisi seçilirse seçilsin sınırı SCMP ile SQAP birlikte ve aynı biçimde tarif etmelidir; iki planın iki ayrı akış anlatması, denetimde bulguya dönüşmeye adaydır. Problem raporlamanın konfigürasyon yönetimindeki yeri 10. bölümde, problem raporlarının sınıflandırılması ve teslimde açık kalan raporların ele alınışı 9. bölümde anlatılır.

Örnek bir uygunsuzluk kaydı

Aşağıdaki kayıt kurgusaldır; bir denetimin tek bir bulgusunun açılıştan kapanışa nasıl işlendiğini gösterir.

Alan	İçerik
Kaynak	Kod gözden geçirme sürecinin denetimi; son temel çizgiye giren dosyalardan 12 gözden geçirme kaydı örneklenmiştir
Gözlem	3 kayıta gözden geçirilen dosya sürümü, temel çizgiye alınan sürümden eskidir; dosyalar gözden geçirmeden sonra değişmiş, yeniden gözden geçirilmemiştir
Dayanak	Yazılım doğrulama planı: gözden geçirme kaydı gözden geçirilen sürümü tanımlar; onaydan sonra yapılan değişiklik yeniden gözden geçirilir
Etki	Üç dosyada kod gözden geçirmesi kanıtı temel çizgideki kodu kapsamıyor; aynı dönemde birleştirilen diğer dosyalar da şüphelidir
Sınıf	Büyük: kredi alınan doğrulama kanıtını etkiliyor

Alan	İçerik
Sorumlu ve hedef	Yazılım geliştirme lideri; bir sonraki temel çizgiden önce
Düzeltilici faaliyet	Üç dosya güncel sürüm üzerinde yeniden gözden geçirilir; aynı dönemde birleştirilen bütün dosyalar için kayıttaki sürüm ile temel çizgideki sürüm karşılaştırılır; kök neden olarak depo, onaydan sonraki değişiklikte onayı düşürecek biçimde ayarlanır
Kapanış kanıtı	Yeniden gözden geçirme kayıtları, sürüm karşılaştırma listesi, depo ayarının kaydı; bir sonraki denetimde yeni örnekleme tekrar görülmemesi

Kayıtta dikkat edilecek üç nokta var: gözlem ölçülebilir biçimde yazılmıştır ("bazı kayıtlar eksik" değil, "12 kayıttan 3'ü"); dayanak bir plan maddesidir, denetçinin kanaati değil; düzeltilici faaliyet yalnız üç dosyayı düzeltmekle kalmaz, sapmanın yayılımını ve kök nedenini de ele alır. Sınıflandırma ölçeği SQAP'ta tanımlanır; buradaki "büyük" yalnızca bir örnektir.

Yazılım uygunluk gözden geçirmesi

Yazılım uygunluk gözden geçirmesi, sertifikasyon başvurusunun parçası olarak sunulacak bir yazılım sürümü için teslimden önce yapılan kapanış denetimidir. Üç şey hakkında güvence arar: yaşam döngüsü süreçleri o sürüm için tamamlanmıştır; yaşam döngüsü verisi eksiksizdir; çalıştırılabilir nesne kodu (executable object code) ve varsa parametre verisi ögesi (parameter data item, PDI) dosyaları kontrol altındadır ve yeniden üretilebilir. Kalite güvencesinin ayrı bir hedefidir ve Seviye A'dan D'ye bütün seviyelerde beklenir.

Uygunluk gözden geçirmesi yeni bir doğrulama turu değildir; var olan kayıtların teslim edilecek sürümle örtüştüğünü sınar. Aşağıdaki tablo, standardın bu gözden geçirmeden beklediği tespitleri yukarıdaki üç güvenceye göre gruplar:

Güvence	Sorulan soru	Tipik kanıt
Süreçler tamam	Planlarda kredi alınacağı söylenen her faaliyet yapılmış ve kaydı saklanmış mı?	Plandaki faaliyet listesinin gözden geçirme, analiz ve test kayıtlarıyla karşılaştırması
Süreçler tamam	Her problem raporu değerlendirilmiş, durumu kayıtlı mı; önceki bir uygunluk gözden geçirmesinden ertelenenler yeniden ele alınmış mı?	Problem raporu listesi, açık raporların gerekçeleri
Veri eksiksiz	Veri plan ve standartlara göre üretilmiş, SCMP'ye göre kontrol altına alınmış mı?	Yazılım konfigürasyon indeksindeki (Software Configuration Index, SCI) sürümlerin arşivdekilerle karşılaştırması
Veri eksiksiz	Veri, kaynaklandığı sistem gereksinimlerine, emniyetle ilgili gereksinimlere ve yazılım gereksinimlerine izlenebiliyor mu?	İz verisi (trace data); örnekleme iki yönde izleme

Güvence	Sorulan soru	Tipik kanıt
Veri eksiksiz	Yazılım gereksinimlerinden sapmalar kayıtlı ve onaylı mı?	Sapma kayıtları ve onayları
Kod kontrollü ve yeniden üretilebilir	Çalıştırılabilir nesne kodu ve varsa PDI dosyaları arşivlenmiş kaynak koddan yeniden üretilebiliyor mu?	Kayıtlı ortamda yeniden derleme ve karşılaştırma kaydı
Kod kontrollü ve yeniden üretilebilir	Onaylı yazılım, yayımlanmış yükleme talimatıyla yüklenebiliyor mu?	Yükleme denemesinin kaydı, kimlik denetimi

Önceden geliştirilmiş yazılım (previously developed software) için kredi alınıyorsa bir soru daha eklenir: güncel ürün temel çizgisi, önceki temel çizgiye ve ona yapılan onaylı değişikliklere izlenebiliyor mu? Plan ve standartlardan sapmalar ise bu listenin değil, yaşam döngüsü boyunca yapılan denetimlerin konusudur. Sertifikasyondan sonraki değişikliklerde standart, gözden geçirmenin değişikliğin önemiyle gerekçelendirilen bir alt kümesinin yapılmasına izin verir.

Zamanlama bu gözden geçirmenin en hassas yanıdır. Gözden geçirme, sertifikasyona esas son temel çizgi üzerinde yapılır; ondan sonra veri ya da kod değişirse en azından etkilenen kelimeler için tekrarlanır. Takvim sıkıştığında gözden geçirmeyi öne çekip ardından "küçük bir düzeltme" yapmak, son denetimin tipik bulgularından birini üretir: kayıt, sunulan sürümü anlatmıyordur. Sürprizleri azaltmanın yolu, son temel çizgiden önce bir prova yapmaktır: yeniden derlemenin aynı çıktıyı vermemesi ya da yükleme talimatındaki eksik adım gibi sorunlar düzeltmeye zaman varken ortaya çıkar.

Çıktı, bulgularıyla birlikte bir kalite güvencesi kayıdır; bulgular uygunsuzluk olarak izlenir ve teslimden önce kapatılır ya da gerekçesiyle kayda geçer. Uygunluk gözden geçirmesi başvuru sahibinin (applicant) kendi iç denetimidir; otoritenin son denetiminin yerine geçmez, ona girdi olur. Otoritenin bu kayıta ve çevresindeki veride nelere baktığı [SW SOI-4](#) kontrol listesindedir.

Etkili ve etkisiz kalite güvencesi

Aynı standardı, aynı planları uygulayan iki projede kalite güvencesinin katkısı çok farklı olabilir. Fark çoğu zaman ekibin büyüklüğünde değil; teknik yetkinliğinde, bağımsızlığında ve projeye ne zaman dahil olduğundadır.

Etkili kalite güvencesinin özellikleri

Etkili bir kalite güvencesi ekibini sahada üç özelliğiyle tanırırsınız:

- **Teknik yetkinlik.** Denetlediği işi anlayacak kadar mühendislik bilgisine sahiptir. Bir gözden geçirme kaydına baktığında yalnızca imzaların tam olup olmadığını değil, gözden geçirmenin gerçekten yapıldı yapılmadığını da sorgulayabilir. Yazılım ekibi, teknik konuşabildiği bir kalite

güvencesi mühendisini ciddiye alır.

- **Bağımsızlık.** Kalite güvencesi, denetlediği ekibin takviminden sorumlu yöneticiye bağlı değildir ve bulgusunu kapattırarak yetkiyi taşır. Bu yetki kâğıt üzerinde kalmamalıdır: açık bir uygunsuzlukla temel çizgi alınamadığını bir kez gören ekip, mekanizmanın gerçek olduğunu anlar.
- **Erken katılım.** Planlama aşamasından itibaren projededir. Planları, standartları ve geçiş kriterlerini daha yazılırken gözden geçirir; ilk denetimini proje sonunda değil, ilk yaşam döngüsü verisi üretilirken yapar.

Erken katılımın etkisi en çok maliyet üzerinde görülür: planlama aşamasında yakalanan bir süreç sapması birkaç saatlik düzeltmeyle kapanırken, aynı sapmanın sertifikasyon denetiminde ortaya çıkması aylarca sürecek bir yeniden çalışma (rework) doğurabilir. Çünkü geç yakalanan sapma yalnızca düzeltilmez; o sapmanın gölgesinde üretilmiş bütün kanıtın yeniden üretilmesi gerekir.

Etkisiz kalite güvencesinin belirtileri

Etkisiz kalite güvencesi de kendini belli eden desenler üretir:

- **Salt evrak kontrolü.** Denetim, "belge var mı, imza var mı" sorusuna indirgenir. İçerik hiç sorgulanmadığı için süreç kâğıt üstünde kusursuz, sahada bozuk olabilir.
- **Geç katılım.** Kalite güvencesi projeye teslimat yaklaşırken dahil olur; artık sapmaları önleme değil, yalnızca belgeleme şansı vardır.
- **Yaptırım gücü eksikliği.** Uygunsuzluk kayıtları açılır ama kimse kapatmak zorunda hissetmez; kayıtlar aylarca açık kalır ve mekanizma inandırıcılığını yitirir.
- **Damgacılık (rubber-stamping).** Ekip, kalite güvencesini bir onay damgası gibi görür; kalite güvencesi de bu role razı olursa bağımsız gözetim fiilen ortadan kalkar.

İki profili yan yana koymak farkı netleştirir:

Boyut	Etkili kalite güvencesi	Etkisiz kalite güvencesi
Odak	Sürecin içeriği ve kanıt kalitesi	Belge ve imza varlığı
Katılım zamanı	Planlama aşamasından itibaren	Teslimata yakın
Teknik derinlik	İşi anlar, teknik soru sorar	Kontrol listesini işaretler
Uygunsuzluk takibi	Kapanış kanıtı ister, gerektiğinde üst yönetime taşır	Kayıt açar, takip etmez
Ekipteki algı	Erken uyarı mekanizması	Aşılması gereken engel
Denetimdeki sonuç	Bulgular önceden kapanmıştır	Bulgular denetimde ortaya çıkar

Pratikte kalite güvencesinin etkisini ölçmenin en kestirme yollarından biri, sertifikasyon otoritesinin SOI denetimlerinde çıkan bulgu sayısına bakmaktır. Etkili bir kalite güvencesi, otoritenin bulacağı sorunları aylar önce kendi uygunsuzluk kayıtlarıyla yakalamış ve kapatmış olur. Bu denetimlerin işleyişi [12. Sertifikasyon İrtibatı](#) bölümünde anlatılır.

Bu bölümden akılda kalması gerekenler

- Kalite güvencesi, süreçlerin ve çıktılarının onaylı plan ve standartlara uygun olduğuna dair güvence sağlar; ürünün teknik doğruluğunu yeniden kanıtlamak doğrulamanın işidir.
- DO-178C'nin hedef tablosunda kalite güvencesi için beş hedef vardır; hepsi bağımsızlıkla karşılanır. Seviye A, B ve C'de beşi de, Seviye D'de planlara uygunluk ve uygunluk gözden geçirmesi geçerlidir.
- Kalite güvencesinde bağımsızlık, geliştirmeden ayrı durmanın yanında düzeltici faaliyeti sağlama yetkisini de içerir; ayrı raporlama hattı bunun alışılmış yoludur.
- Denetim örneklemeyle yapılır: süreç ve veri denetimi, gözden geçirmelere katılım, tanıklık, geçiş kriterlerinin denetimi ve tedarikçi gözetimi SQAP'ta planlanır, sonuçları kalite güvencesi kayıtlarına geçer.
- Uygunsuzluk, plan ya da standarda aykırılıktır; problem raporlamanın süreç tarafıdır. Kaydı, etkisinin değerlendirilmesi, kök nedenin giderilmesi ve kanıtlarla kapatılması bu disiplinin merkezindedir.
- Yazılım uygunluk gözden geçirmesi teslimden önceki son kapıdır: süreçlerin tamamlandığını, verinin eksiksiz olduğunu ve nesne kodun yeniden üretilip yüklenebildiğini son temel çizgi üzerinde gösterir.
- Etkili kalite güvencesi teknik yetkinlik, bağımsızlık ve erken katılımı tanıtır; salt evrak kontrolü, geç katılım ve yaptırım gücü eksikliği etkisizliğin belirtileridir.

12. Sertifikasyon İrtibatı

Sertifikasyon irtibatı (certification liaison), onayı isteyen kuruluş ile sertifikasyon otoritesi (certification authority) arasında uyumun nasıl gösterileceği üzerine kurulan ve proje boyunca süren mutabakat sürecidir. Süreç yazılım sertifikasyon planıyla (Plan for Software Aspects of Certification, PSAC) başlar, yazılım başarı özetiyle (Software Accomplishment Summary, SAS) kapanır. Bu bölüm sürecin hedeflerini, taraflarını ve kanallarını, SAS'ın içeriğini, katılım aşaması (Stage of Involvement, SOI) denetimlerini ve sertifikasyon uçuş testlerinden önce beklenen yazılım olgunluğunu anlatır.

Sertifikasyon yalnızca belge teslimi değildir; ortak anlayış kurma sürecidir. Teknik ekip ne kadar iyi çalışırsa çalışsın, uyumun nasıl gösterileceği otoriteyle erken konuşulmamışsa ya da kanıt dağınık sunuluyorsa süreç yavaşlar; en kötü durumda tamamlanmış iş yeniden yapılır. İrtibat bu yüzden proje sonunda yapılan bir teslim değil; doğrulama, konfigürasyon yönetimi (configuration management) ve kalite güvencesi (quality assurance) gibi yaşam döngüsü boyunca süren bütüncü süreçlerden (integral processes) biridir.

İrtibat süreci ne ister?

DO-178C sertifikasyon irtibatı için üç hedef (objective) tanımlar. Üçü de Seviye A'dan Seviye D'ye kadar her yazılım seviyesinde (software level) geçerlidir: seviye düştükçe üretilecek kanıt azalır, otoriteyle anlaşma yükümlülüğü azalmaz.

Hedef	Pratikte anlamı	Dayandığı veri
Başvuru sahibi (applicant) ile otorite arasında iletişim ve ortak anlayış kurulur	Otorite projeyi, yazılımın sistemdeki yerini ve takvimi bilir; sorular ortaya çıktığı anda konuşulur	PSAC
Uyum yöntemi (means of compliance) önerilir ve PSAC üzerinde mutabakat sağlanır	Uyumun hangi belgeye göre, hangi yaşam döngüsü, araç ve yöntemle gösterileceği iş başlamadan kararlaştırılır	PSAC
Uyum kanıtı sunulur	Planlananın yapıldığı ve hedeflerin karşılandığı gösterilir	SAS ve yazılım konfigürasyon indeksi (Software Configuration Index, SCI)

İkinci hedefin arkasında standardın hukuki konumu vardır: DO-178C bir yasa değil, otoritelerin kabul ettiği bir uyum yöntemidir ([4. DO-178C ve Destekleyici Dokümanlara Genel Bakış](#)). Bu yüzden "uyumu neye göre göstereceğiz?" sorusunun cevabı varsayılmaz, PSAC'ta yazılır: hangi belge, hangi teknoloji ekleri (supplement), varsa hangi alternatif yöntem (alternative method).

PSAC üzerindeki mutabakat ürünün onayı değildir; otorite, önerilen yol izlenirse uyumun gösterilebileceğini kabul etmiş olur. Bunun iki sonucu vardır. Birincisi, PSAC erken gönderilmelidir: planlara göre aylarca veri üretildikten sonra gelen bir itiraz, o verinin bir kısmını

geçersiz kılar. İkincisi, mutabakat PSAC'ın anlattığı projeyi kapsar: yazılım seviyesi, kapsam, araç ya da yöntem değişirse PSAC güncellenir ve otoriteye yeniden bildirilir. PSAC'ın içeriği [5. Yazılım Planlama](#) bölümünde anlatılır.

Otoriteye sunulan veri ile erişime açık tutulan veri aynı şey değildir. Asgari olarak üç veri sunulur: PSAC, SCI ve SAS; üçü de her yazılım seviyesinde tam kontrol isteyen birinci kontrol kategorisindedir (CC1). Geri kalan yazılım yaşam döngüsü verisi (software life cycle data) — diğer planlar, standartlar, gereksinim ve tasarım verisi, doğrulama sonuçları, problem raporları (problem report), konfigürasyon yönetimi ve kalite güvencesi kayıtları — gönderilmez; otorite istediğinde, çoğunlukla denetimde, gösterilir. Hangi verinin sunulacağı, hangisinin talep üzerine gösterileceği PSAC'ta yazılır. Bu ayırım yükü hafifletmez: sunulmayan veri de denetim günü, kontrol altındaki sürümüyle dakikalar içinde açılabilir.

Kalifikasyon gerektiren bir araç kullanılıyorsa asgari liste uzar: DO-330, yazılım yaşam döngüsü ortam konfigürasyon indeksinin (Software Life Cycle Environment Configuration Index, SECI) de sunulmasını bekler. Araç kalifikasyon seviyesi (tool qualification level, TQL) TQL-1 ile TQL-4 arasındaysa buna araç kalifikasyon planı ile aracın konfigürasyon indeksi ve başarı özeti eklenir.

Verinin bir bölümü ayrıca ürünün tip tasarımıyla (type design) ilişkilidir. Otoriteyle aksi kararlaştırılmadıkça bu küme altı veriden oluşur: gereksinim verisi, tasarım tanımı, kaynak kod, çalıştırılabilir nesne kodu (executable object code) ile varsa parametre verisi ögesi (parameter data item, PDI) dosyaları, SCI ve SAS. Bu verinin geri getirilebilmesi ve onayı, tip tasarım verisine ilişkin uçuşa elverişlilik kurallarına tabidir.

İyi yürüyen bir irtibatın ortak özellikleri vardır ve hepsi otoritenin ek soru sorma ihtiyacını azaltır:

- **Erken ve sürprizsizdir.** Kalifikasyon gerektiren bir araç ([13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)), yeniden kullanılan bir bileşen ya da plandan sapma, otoritenin denetimde kendi bulduğu bir şey olmamalıdır.
- **Kısa ama eksiksizdir.** Otorite uzmanının zamanı sınırlıdır; özet belge kararı ve gerekçesini verir, ayrıntı için kontrol altındaki veriye kimliğiyle atıf yapar.
- **Kanıtı dayanır.** "Yaptık" cümlesinin yanında kaydın kimliği ve sürümü bulunur.
- **Savunmacı değil, açıktır.** Bilinen zayıflık, gerekçesi ve kapanış planıyla birlikte söylenir. Soruyu tahmin edip cevabını önceden yazmak, soruyu geçiştirmekten ucuzdur.
- **Tutarlıdır.** Sürüm numarası, açık rapor listesi ve terimler her belgede aynı görünür; belgeler arası küçük bir tutarsızlık, otoriteye örneklemeyi genişlettirir.

Kiminle ve hangi kanaldan?

"Otorite" tek bir kişi değildir ve her projede aynı derinlikte yer almaz. İrtibatın tarafları şöyle özetlenebilir:

Taraf	Rolü
Başvuru sahibi	Onay için başvuran kuruluş: uçak, motor ya da ekipman üreticisi. Uyumu göstermek ve irtibatı yürütmek onun sorumluluğudur
Sertifikasyon otoritesi	Uyum gösterimini kabul eden kurum ve onun yazılım uzmanı: FAA, EASA ya da ulusal sivil havacılık otoritesi (Türkiye'de Sivil Havacılık Genel Müdürlüğü)
Yetkili temsilci ya da onaylı organizasyon	Otoritenin, uyumu kendi adına ya da kendi gözetimi altında doğrulama yetkisi tanıdığı kişi ya da birim. FAA düzeninde yetkilendirilmiş mühendislik temsilcisi (Designated Engineering Representative, DER) ve organizasyon yetkilendirmesi (Organization Designation Authorization, ODA); EASA düzeninde tasarım organizasyonu onayına (Design Organisation Approval, DOA) sahip kuruluşun uyum doğrulama mühendisleri (Compliance Verification Engineer, CVE)
Tedarikçi	Yazılımı ya da bir bölümünü geliştiren alt yüklenici. Kanıtı üretir; irtibat başvuru sahibi üzerinden yürür

Otoritenin projeye ne kadar yakından bakacağı sabit değildir. Katılım düzeyi (level of involvement) risk temelli belirlenir: yazılım seviyesi, kullanılan teknolojinin ya da yöntemin yeniliği, tasarımın karmaşıklığı, başvuru sahibinin deneyimi ve geçmiş performansı. Seviye A yazılımı ilk kez geliştiren bir kuruluş her denetimde otorite uzmanını karşısında bulur; olgun bir kuruluşun türev projesinde otorite işin büyük kısmını yetkili temsilciye bırakıp yalnızca PSAC ile SAS'ı kendisi inceleyebilir. Hangi durumun geçerli olduğu tahmin edilmez; proje başında otoriteye sorulur ve otoritenin katılacağı noktalar PSAC'a yazılır.

İrtibat birkaç kanaldan yürür ve her kanalın bir zamanı vardır:

Kanal	Ne zaman	Ne için
Tanışma ve planlama toplantıları	Proje başında, PSAC gönderilmeden önce	Sistem, yazılım seviyesi, yenilikler ve takvim üzerine ilk ortak anlayış
PSAC	Geliştirme verisinin üretimi hızlanmadan	Uyum yöntemi, yaşam döngüsü, ek hususlar ve otoritenin katılım noktaları üzerine mutabakat
Konuya özel yazılı mutabakat belgesi	Genel rehberin doğrudan cevaplamadığı bir konu çıktığında	Otoritenin beklentisi ile başvuru sahibinin önerisinin aynı belgede kayda geçmesi: FAA'da "issue paper", EASA'da sertifikasyon gözden geçirme maddesi (Certification Review Item, CRI)
SOI denetimleri	Yaşam döngüsünün tipik olarak dört durağında	Sürecin planlara göre işlediğinin örneklemeye görülmesi
SCI ve SAS	Sertifikasyona esas sürüm belirlendiğinde	Uyum kanıtının sunulması

Alternatif yöntemler de aynı kanalı izler. Bir hedef standardın tarif ettiğinden farklı bir yolla karşılanacaksa — örneğin servis geçmişinden kredi alınacaksa — bu, PSAC'ta gerekçesiyle önerilir ve mutabakat işe başlamadan alınır. Sonradan savunulan alternatif yöntem, tartışmayı projenin en az esnek olduğu ana taşıır.

Tedarikçi kullanıldığında irtibat zinciri uzar ama sorumluluk yer değiştirmez. Otoritenin muhatabı başvuru sahibidir; tedarikçinin planları, sapmaları ve açık problem raporları başvuru sahibinin PSAC ve SAS'ına yansır, denetimler gerektiğinde tedarikçinin tesisinde yapılır. Gözetimin nasıl kurulacağı **26. Yazılım Yaşam Döngüsü Faaliyetlerinde Dış Kaynak Kullanımı** bölümündedir. Ekipman tek başına onaylanıyorsa — teknik standart emri (Technical Standard Order, TSO) ya da Avrupa'daki karşılığı ETSO kapsamında — başvuru sahibi ekipman üreticisidir ve yazılım kanıtını o sunar. Ekipmanın belirli bir uçağa takılması ayrı bir onay konusudur; uçak üreticisi SAS'taki kısıtlara ve açık problem raporlarına o aşamada ihtiyaç duyar.

Yazılım başarı özeti

Yazılım başarı özeti, yazılım yaşam döngüsünün sonunda otoriteye sunulan kapanış belgesidir. En kısa tanımıyla SAS, projenin başında verilen sözün — yani PSAC'ın — ne ölçüde ve nasıl yerine getirildiğini anlatır. PSAC "şunu şöyle yapacağız" derken, SAS "şunu şöyle yaptık; sapmalar, gerekçeleri ve kalan açık noktalar da bunlardır" der.

Bu ikili yapı, sertifikasyon irtibatının belki de en güçlü aracıdır: otorite iki belgeyi yan yana koyduğunda projenin bütün hikâyesini görebilmelidir. Bu yüzden SAS, PSAC ile aynı başlık düzenini izleyecek biçimde yazılırsa hem yazması hem okuması kolaylaşır.

İyi bir SAS'ta tipik olarak şu içerik bulunur:

İçerik	Yanıtladığı soru
Sistem ve yazılıma genel bakış	Bu yazılım ne yapar, hangi sistemin parçasıdır?
Sertifikasyon hususları	Hangi yazılım seviyesi atandı, sistem emniyet değerlendirme süreciyle uyumlu mu; uyum hangi belgelere göre gösterildi?
Ek hususlar (additional considerations)	Hangi özel durumlar (araç kalifikasyonu, önceden geliştirilmiş yazılım (previously developed software, PDS), alternatif yöntemler) kullanıldı?
Yaşam döngüsü özeti	Süreçler planlandığı gibi mi yürüdü?
Planlardan sapmalar	Nerede, neden ve hangi onayla plandan ayrıldı?
Yaşam döngüsü verisi	Hangi veri üretildi; geçerli SCI ve SECI hangi kimlik ve sürümde; otorite bu veriye nasıl erişir?
Tedarikçi gözetimi	Tedarikçinin süreç ve çıktıların planlara ve standartlara uyduğu nasıl güvence altına alındı? (tedarikçi kullanıldıysa)
Yazılım tanımlaması	Sunulan yazılımın tam konfigürasyonu (sürüm, parça numarası) nedir?
Yazılım karakteristikleri	Çalıştırılabilir nesne kodunun boyutu, zamanlama ve bellek payları ne kadar; her biri nasıl ölçüldü ya da hesaplandı?
Değişiklik geçmişi	Önceki sertifikasyondan bu yana yazılımda ve süreçlerde ne değişti; hangi değişiklik emniyeti etkileyen bir arıza yüzünden yapıldı? (değişiklik projelerinde)

İçerik	Yanıtladığı soru
Yazılım durumu	Sertifikasyon anında açık kalan problem raporları hangileri, hangi sınıfta; işleve, operasyona ve emniyete etkileri, açık bırakma gerekçesi ve hafifletici önlemleri nedir?
Uyum beyanı (compliance statement)	DO-178C'ye uyulduğuna dair başvuru sahibinin açık ifadesi ve uyumun hangi yöntemlerle gösterildiğinin özeti; otoritenin ek kararları ve başka başlıkta geçmeyen sapmalar da burada ele alınır

Tablo, DO-178C'nin SAS için saydığı on iki içerik başlığını iki farkla izler. Standart sistem ve yazılım genel bakışını ayrı başlıklar olarak sayar. Sapmalar için ise ayrı bir başlık öngörmez: her başlık PSAC'ta önerilenden farkı kendi konusunda anlatır, başka yerde geçmeyen sapmalar uyum beyanında ele alınır. Sapmaları ayrıca tek bir listede toplamak, okuyanın işini kolaylaştıran yaygın bir uygulamadır.

Yazılım karakteristikleri, otoritenin SAS'ta özellikle aradığı bilgidir: en kötü durum yürütme süresi (worst-case execution time, WCET) dahil zamanlama payları, bellek ve yığın payları ve kaynak sınırlamaları, ölçüm ya da analiz yöntemiyle birlikte verilir. Bu sayılar [9. Yazılım Doğrulama](#) bölümünde anlatılan analizlerin sertifikasyona esas sürüm için güncellenmiş sonuçlarıdır; aylar önceki bir derlemenin payını SAS'a taşımak sık görülen bir hatadır.

SAS'ın yayımlanması tek başına bir yazı işi değildir. Kalite güvencesi, sertifikasyona sunulan sürüm için yazılım uygunluk gözden geçirmesini (software conformity review) tamamlar ([11. Yazılım Kalite Güvencesi](#)); SAS'taki uyum beyanı bu gözden geçirmenin sonucuna dayanır.

Deneyimden birkaç öneri:

- **SAS'ı proje sonunda sıfırdan yazmayın.** Her önemli kilometre taşında (özellikle SOI denetimlerinden sonra) taslağı güncellerseniz, kapanışta yalnızca son durumu işlemek kalır. Proje sonunda hafızadan yazılan SAS, tutarsızlıkların en sık görüldüğü belgedir.
- **Sapmaları saklamayın.** Otorite, plandan hiç sapmamış bir proje beklemes; sapmayı fark edip gerekçelendiren ve kalite güvencesi kaydına bağlayan bir ekip bekler. Denetimde ortaya çıkan gizli sapma, belgede açıkça yazılmış on sapmadan daha çok güven kaybettirir.
- **Açık problem raporlarını tek tek değerlendirin.** Her açık rapor, planlarda tanımlı şemaya göre tek bir sınıfa atanır; FAA AC 20-189 ve EASA AMC 20-189 bu şemanın en az önemli (significant), işlevsel (functional), süreç (process) ve yaşam döngüsü verisi (life-cycle data) sınıflarını içermesini bekler. Sınıfın yanında etkilenen işlev, getirdiği kısıt ve neden bu hâliyle kabul edilebilir olduğu yazılır. Etki değerlendirmesi yazılım ekibinin tek başına vereceği bir karar değildir; sistem ve emniyet ekibiyle birlikte yapılır. Sınıfların tanımı ve değerlendirme soruları [9. Yazılım Doğrulama](#) bölümündedir. "Önemsiz olduğu için açık bırakıldı" gibi genel bir cümle, otoritenin ek soru sormasını garanti eder.
- **Konfigürasyon tanımlamasını derleme (build) kanıtına bağlayın.** SAS'ta beyan edilen sürüm, SCI'de ve gerçek derleme çıktısında birebir aynı olmalıdır; buradaki bir yazım hatası bile son denetimde zaman kaybettirir. SCI'nin içeriği [10. Yazılım Konfigürasyon Yönetimi](#) bölümünde anlatılır.

SAS, sertifikasyon irtibatının final sınavı gibidir: buraya kadar anlatılan açıklık, tutarlılık ve kanıta dayalı anlatım ilkelerinin hepsi bu tek belgede sınanır.

Katılım aşaması denetimleri

Otorite, projeyi tipik olarak dört katılım aşaması denetimiyle izler: planlama (SOI-1), geliştirme (SOI-2), doğrulama (SOI-3) ve son sertifikasyon (SOI-4).

Bu denetimler DO-178C'de tanımlı değildir; standart yalnızca otoritenin yaşam döngüsü süreçlerini ve çıktılarını uygun gördüğü ölçüde gözden geçirebileceğini, başvuru sahibinin de bu gözden geçirmeleri düzenleyip veriyi erişime açacağını söyler. Dört aşamalı düzen otorite uygulamasından gelir: FAA'nın yazılım onay yönergesi Order 8110.49 (güncel sürümü 8110.49A) ile yazılım gözden geçirmelerine ilişkin iş yardımcısı (job aid) bu düzeni tarif eder. EASA tarafında aynı düzeni anlatan CM-SWCEH-002 sertifikasyon notu yürürlükten kalkmıştır; yine de Avrupa'daki projelerde aynı düzen yaygın olarak izlenir. "Dört" bu yüzden bir kural değil, tipik durumdur: katılım düzeyine göre aşamalar birleştirilebilir, bir bölümü belge üzerinden masa başında yapılabilir ya da yetkili temsilciye bırakılabilir; büyük projelerde bir aşama birden çok oturuma da bölünebilir.

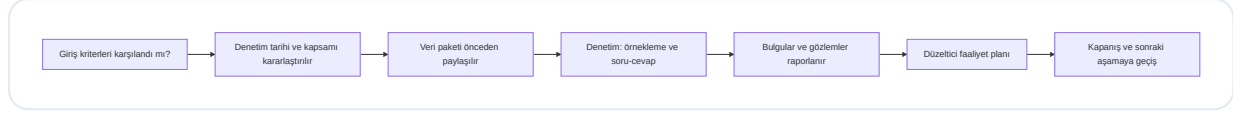
SOI denetimlerinin mantığı basittir: otorite, kanıtın tamamını proje sonunda tek seferde görmek yerine, yaşam döngüsünün doğal duraklarında örnekleme yaparak inceler. Böylece sistematik bir sorun varsa erken yakalanır; proje sonunda "her şeyi yeniden yap" riskine girilmez. Her aşamanın kabaca giriş kriteri ve odağı şöyle özetlenebilir:

Aşama	Tipik giriş kriteri	Otoritenin odağı
SOI-1 (Planlama)	Yazılım seviyesinin gerektirdiği planlar ve standartlar (Seviye A, B ve C'de beş plan ve üç standart) yayımlanmış ve gözden geçirilmiş, ekip planlara göre çalışmaya başlamış	Planların hedefleri karşılaması, birbirleriyle tutarlılığı, araç kalifikasyonu kararları
SOI-2 (Geliştirme)	Gereksinim, tasarım ve kodun temsil edici bir bölümü üretilmiş ve gözden geçirilmiş	Süreçlerin plana uygun işlediği, izlenebilirlik, geliştirme verisinin kalitesi
SOI-3 (Doğrulama)	Doğrulama verisinin temsil edici bir bölümü tamamlanmış ve gözden geçirilmiş, kapsam analizi sonuç üretmeye başlamış	Test ve gözden geçirme kanıtının yeterliliği, yapısal kapsam boşluklarının çözümü, problem raporu yönetimi
SOI-4 (Son sertifikasyon)	Tüm faaliyetler tamamlanmış, önceki bulgular kapanmış, SAS ve SCI yayımlanmış	Açık kalemlerin kapanışı, son konfigürasyon, uyum beyanı

"Temsil edici bölüm" için otoriteler farklı eşikler anmıştır: FAA'nın Order 8110.49'u SOI-2 ve SOI-3 için tipik olarak en az yarısını anar, EASA'nın yürürlükten kalkan notu en az dörtte üçünü arıyordu. Projede geçerli eşik PSAC'ta otoriteyle kararlaştırılır. Öteki kriterler de otoriteden otoriteye ve

projeden projeye küçük farklılıklar gösterir; tam listeler için Kaynaklar kısmındaki [SW SOI-1](#), [SW SOI-2](#), [SW SOI-3](#) ve [SW SOI-4](#) kontrol listelerine bakılabilir.

Denetim akışı genellikle şu düzende ilerler:



Denetimin çıktısı çoğunlukla üç türde kaydedilir: bir hedefe uyumun gösterilemediğini söyleyen bulgu (finding), uyumsuzluk olmayan ama iyileştirme öneren gözlem (observation) ve sorumlusu ile tarihi belli aksiyon (action). Yanıt planında öncelik bulgulardadır: onaydan önce kapanmaları gerekir. Gözlem bir uyum sorunu değildir; yine de değerlendirilip yanıtlanması, aynı konunun sonraki denetimde bulguya dönüşmesini önler.

Hazırlık için sahada işe yarayan birkaç öneri:

- **Giriş kriterini kendiniz doğrulamadan denetim istemeyin.** Erken çağrılan bir SOI-2, "veri henüz olgun değil" bulgusuyla kapanır ve hem takvimi hem güveni zedeler. Denetim öncesi bir iç ön denetim (kalite güvencesinin yürüttüğü bir prova; bkz. [11. Yazılım Kalite Güvencesi](#)) en etkili yatırımdır. Yaşam döngüsü geçiş kriterleriniz ([Ek A: Örnek Geçiş Kriterleri](#)) düzenli işletiliyorsa bu doğrulamanın kanıtı zaten elinizdedir.
- **İzlenebilirliği canlı gösterebilecek durumda olun.** Otorite genellikle rastgele bir gereksinim seçer ve tasarıma, koda, teste kadar izini sürmek ister. Bu zincir ancak araç üzerinde dakikalar içinde gösterilebiliyorsa ikna edicidir.
- **Bulguları kişiselleştirmeyin, sistematik kökü arayın.** Tek bir gözden geçirme kaydındaki eksik imza önemsiz görünebilir; otoritenin asıl sorusu "bu tekil bir hata mı, süreç boşluğu mu"dur. Düzeltici faaliyet yanıtı da bu soruya göre yazılmalıdır.
- **Önceki aşamanın bulgularını kapatmadan — ya da en azından kapanış planında otoriteyle mutabık kalmadan — sonrakine girmeyin.** Sahipsiz bir SOI-2 bulgusuyla geline SOI-3, daha ilk saatte güven sorununa dönüşür. SOI-4'te ise önceki bulguların hepsinin kapanmış olması beklenir.
- **Denetim kayıtlarını konfigürasyon yönetimi altına alın.** Sorulan sorular, verilen yanıtlar ve taahhütler; sonraki aşamalarda "bunu konuşmuştuk" diyebilmenin tek dayanağıdır.

Sertifikasyon uçuş testlerinden önce yazılım olgunluğu

Sertifikasyon amaçlı uçuş testleri, yazılım açısından özel bir eşiktir: uçakta artık "deneme" yazılımı değil, sertifikasyona esas kanıtın toplanacağı yazılım koşturmaktadır. Eşiğin resmî bir karşılığı da vardır; örneğin FAA düzeninde sertifikasyon uçuş testleri tip muayene yetkilendirmesi (Type Inspection Authorization, TIA) verildikten sonra başlar. Öte yandan bu uçuşlar çoğu zaman

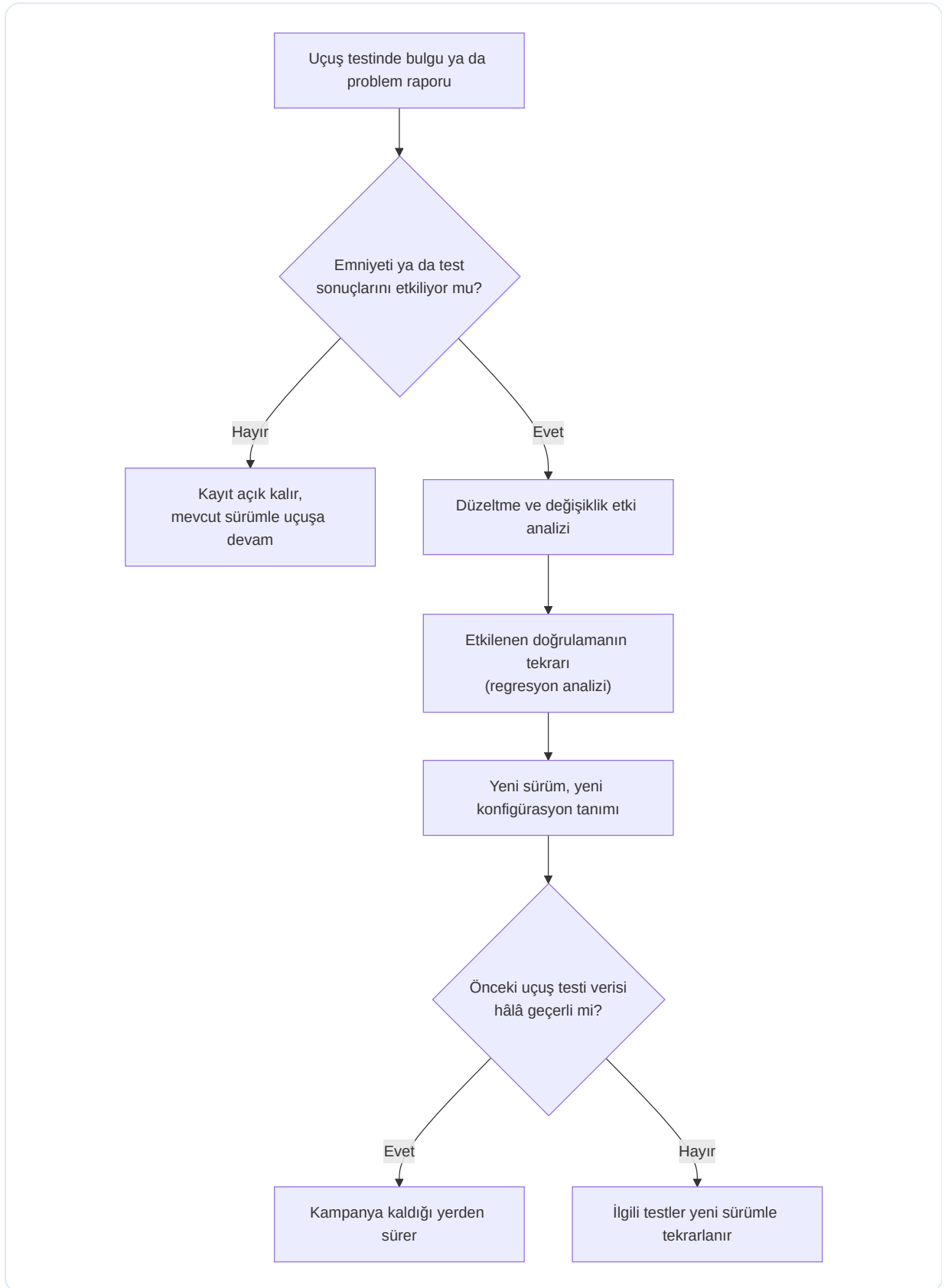
yazılımın son onayından önce yapılır: doğrulama sürmektedir, SOI-4 henüz yapılmamıştır. Uçaktaki sürüm bu yüzden henüz onaylı ürün değildir ve otoriteler, resmî uçuş testine girecek yazılımın belirli bir olgunluk düzeyine ulaşmış olmasını bekler.

Bazı üreticilerde bu ayırım etiket renkleriyle anılır: geliştirme ve test amaçlı, doğrulaması henüz tamamlanmamış donanım ve yazılım **kırmızı etiket** (red label), doğrulaması tamamlanmış ve konfigürasyonu onaylı ürün **siyah etiket** (black label) diye adlandırılır. Bu adlar bir standart ya da otorite terimi değildir; kuruluşunuzda başka adlar kullanılıyor olabilir, önemli olan ayırımın kendisidir.

Sertifikasyon uçuşuna aday bir yazılım sürümünden pratikte beklenenler şunlardır:

- **Gereksinimler ve kod dondurulmuş olmalıdır.** Uçuş testi sırasında işlevsellik hâlâ değişiyorsa, toplanan kanıtın hangi konfigürasyona ait olduğu tartışmalı hâlâ gelir.
- **Doğrulama büyük ölçüde tamamlanmış olmalıdır.** Kalan doğrulama işi (örneğin son kapsam analizi boşlukları) tanımlı, sınırlı ve emniyeti etkilemediği değerlendirilmiş olmalıdır.
- **Açık problem raporları değerlendirilmiş olmalıdır.** Uçuş emniyetini ya da test edilecek işlevi etkileyen açık kayıtlar uçuşa çıkmaz; kalanların "bu hâliyle uçulabilir" kararı kayıt altında olmalıdır.
- **Sürüm, konfigürasyon yönetimi altında ve yeniden üretilebilir olmalıdır.** Uçaktaki çalıştırılabilir nesne kodunun hangi kaynak koddan, hangi araç zinciriyle üretildiği kesin olarak izlenebilmelidir.
- **Yükleme ve uygunluk kontrolü yapılmış olmalıdır.** Uçaktaki birimde gerçekten beyan edilen sürümün yüklü olduğu (parça numarası, sağlama toplamı) uçuş öncesi doğrulanır; buna uygunluk denetimi (conformity inspection) eşlik eder. Bu denetim uçtaki fiziksel konfigürasyonun beyan edilenle aynı olduğuna bakar; kalite güvencesinin yazılım uygunluk gözden geçirmesiyle karıştırılmamalıdır. Yüklemenin nasıl doğrulanacağı [18. Sahada Yüklenebilir Yazılım](#) bölümünde anlatılır.

Uçuş testi kampanyası aylar sürebildiği için yazılımın hiç değişmemesi gerçekçi değildir. Kritik olan, değişikliğin kontrollü olması, yani her değişikliğin değişiklik etki analizinden (change impact analysis) geçmesidir:



Buradaki en pahalı hata, "küçük" bir yazılım değişikliğinin hangi uçuş testi noktalarını geçersiz kıldığının analiz edilmemesidir. Analiz yapılmadan sürüm atlırsa, otorite haklı olarak önceki uçuşlarda toplanan verinin geçerliliğini sorgular ve testlerin tekrarını isteyebilir. Analizin yazılım

tarafında neleri kapsadığı **10. Yazılım Konfigürasyon Yönetimi** bölümünde anlatılır; uçuş testi döneminde buna "hangi test noktaları yeniden uçulmalı" sorusu eklenir.

Son bir deneyim notu: uçuş test ekibi ile yazılım ekibi arasındaki iletişim de bir sertifikasyon irtibatı konusudur. Uçakta hangi sürümün olduğu, bilinen kısıtlamaların neler olduğu ve hangi işlevlerin henüz doğrulanmadığı, her uçuş öncesi test ekibine yazılı olarak bildirilmelidir. Bu bilgi akışı koptuğunda hem emniyet riski doğar hem de toplanan kanıt kullanılamaz hâle gelir.

Bu bölümden akılda kalması gerekenler

- Sertifikasyon irtibatı PSAC ile başlar, SAS ile kapanır. Üç hedefi — ortak anlayış, uyum yöntemi üzerinde mutabakat, uyum kanıtının sunulması — Seviye A'dan Seviye D'ye her yazılım seviyesinde geçerlidir.
- Otoriteye asgari olarak PSAC, SCI ve SAS sunulur, kalifiye araç varsa DO-330 bu listeyi genişletir; geri kalan veri talep edildiğinde, kontrol altındaki sürümüyle gösterilebilir olmalıdır.
- Otoritenin muhatabı başvuru sahibidir ve katılım düzeyini otorite belirler. Genel rehberin cevaplamadığı konular ve alternatif yöntemler iş başlamadan yazılı mutabakata bağlanır.
- SAS, PSAC'ın aynasıdır: sapmalar, zamanlama ve bellek payları, sınıflandırılmış açık problem raporları ve uyum beyanı, SCI ile birebir tutarlı biçimde yazılır.
- SOI denetimleri DO-178C'den değil otorite uygulamasından gelir ve dört aşama tipik durumdur. Giriş kriterleri karşılanmadan denetim istenmez; önceki bulgular kapatılmış ya da kapanış planında mutabık kalınmış olmalıdır. İç ön denetim en ucuz sigortadır.
- Sertifikasyon uçuş testine giren yazılım dondurulmuş, büyük ölçüde doğrulanmış ve konfigürasyonu kesin olarak tanımlanmış olmalıdır; değişiklik kaçınılmazsa değişiklik etki analiziyle yönetilir.

KISIM IV

Araç Kalifikasyonu ve DO-178C Ekleri

13. DO-330 ve Yazılım Aracı Kalifikasyonu

Araç kalifikasyonu (tool qualification), çıktısı ayrıca doğrulanmadan kullanılan bir yazılım aracına duyulan güvenin kanıtla temellendirilmesidir. Bu bölüm, kalifikasyonun hangi araçlar için gerektiğini, üç ölçüt ile yazılım seviyesinden araç kalifikasyon seviyesinin nasıl bulunduğunu ve DO-330'un bu seviyeye göre aracın geliştiricisinden ve kullanıcılarından hangi kanıtı beklediğini anlatır.

Araç neden kalifiye edilir?

Derleyiciden test betiğine kadar her araç, yaşam döngüsündeki bir işi insandan devralır. İş hızlanır; ama aracın hatası da o işin çıktısına sessizce taşınır. Araç kaynaklı risk iki temel biçimde ortaya çıkar:

- **Hata eklemek:** araç, ürüne giren bir çıktıyı yanlış üretir. Bir kod üretici (code generator) modeli hatalı koda çevirebilir; bir dönüştürme aracı tablodaki bir değeri kesip yuvarlayabilir.
- **Hatayı gözden kaçırmak:** araç, var olan bir hatayı raporlamaz. Bir test aracı başarısız adımı başarılı gösterebilir; bir kapsam aracı hiç çalışmamış bir dalı çalışmış sayabilir; bir statik analiz (static analysis) aracı bir ihlali atlayabilir.

İkinci grupta sık yapılan bir karıştırma vardır: kalifikasyonu ilgilendiren, aracın yanlış alarm vermesi (false positive) değil, gerçek bir hatayı raporlamamasıdır (false negative). Yanlış alarm iş yükü doğurur ve mühendis sonucu incelediğinde düşer; raporlanmayan hata ise kimsenin bakmadığı yerde kalır.

Bu risklerin varlığı tek başına kalifikasyon gerektirmez. Belirleyici olan, aracın hatasını yakalayacak başka bir faaliyetin bulunup bulunmadığıdır: araç bir faaliyeti kaldırıyor, azaltıyor ya da otomatikleştiriyorsa ve çıktısına başka kimse bakmıyorsa, "araç var" demek yetmez; aracın o kullanım amacı için güvenilir olduğu gösterilmelidir. Böylece otomasyon sertifikasyon kanıtını zayıflatmak yerine güçlendirir.

DO-330 nedir, DO-178C ile ilişkisi

DO-330, "Software Tool Qualification Considerations" adıyla 2011'de DO-178C ile birlikte yayımlanmıştır; EUROCAE karşılığı ED-215'tir. DO-331, DO-332 ve DO-333'ün aksine bir teknoloji eki (supplement) değildir: DO-178C'nin hedeflerine (objective) ekleme yapmaz, kendi hedef tabloları olan bağımsız bir araç kalifikasyonu belgesidir. Bu yapı onu alandan bağımsız kılar; örneğin yer tabanlı haberleşme, seyrüsefer, gözetim ve hava trafik yönetimi (CNS/ATM) yazılımını konu alan DO-278A da araç kalifikasyonu için ona başvurur ve aynı mantık havacılık verisi işleyen araçlara da taşınır (bkz. [23. Havacılık Verileri](#)).

İki belge arasındaki iş bölümü nettir:

- **DO-178C**, "bu araç kalifiye edilmeli mi, edilecekse hangi seviyede?" sorusunu yanıtlar. Kalifikasyonun ne zaman gerektiği, aracın rolünü sınıflandıran üç ölçüt ve seviye tablosu, DO-178C'nin ek hususlar (additional considerations) kısmındadır (bölüm 12.2).
- **DO-330**, "bu seviyede kalifikasyon nasıl yapılır?" sorusunu yanıtlar: aracın yaşam döngüsü süreçlerini, seviyeye göre karşılanacak hedefleri ve üretilecek veriyi tanımlar.

FAA AC 20-115D ve EASA AMC 20-115D, DO-330'u DO-178C ve teknoloji ekleriyle birlikte kabul edilebilir uyum yöntemi olarak tanıır. Belge ailesinin tamamı [4. DO-178C ve Destekleyici Dokümanlara Genel Bakış](#) bölümünde tablo hâlinde verilmiştir.

Kalifikasyon ihtiyacının ve seviyesinin belirlenmesi

Her araç kalifikasyon gerektirmez. Karar, iki temel soruya verilen yanıtla başlar:

1. **Araç hata ekleyebilir mi ya da mevcut bir hatayı gözden kaçırabilir mi?** Örneğin bir kod üretici, ürettiği kaynak koda hata ekleyebilir; bir test aracı ise başarısız bir testi başarılı gösterebilir.
2. **Aracın çıktısı doğrulama süreciyle ayrıca doğrulanıyor mu?** Aracın olası hatasını yakalayacak bir gözden geçirme, analiz ya da test varsa (örneğin üretilen kodun gözden geçirilmesi ve test edilmesi), araç hatası ürünü etkilemeden yakalanır ve kalifikasyona gerek kalmaz.

Yalnızca "araç hata yapabilir **ve** çıktısı doğrulanmıyor" durumunda kalifikasyon gündeme gelir; bu koşul geliştirme araçları kadar doğrulama araçları için de geçerlidir. Değerlendirme, aracın *fiilî kullanım biçimine* göre yapılır: aynı araç bir projede kalifikasyon gerektirirken, çıktısı elle doğrulanan başka bir projede gerektirmeyebilir. Derleyici bunun bilinen örneğidir: ürettiği nesne kodu gereksinim tabanlı testlerle hedef ortamda doğrulandığı için genellikle kalifiye edilmez. Karar planlama aşamasında verilir ve yazılım sertifikasyon planına (Plan for Software Aspects of Certification, PSAC) yazılır; bu yanı [5. Yazılım Planlama](#) bölümünde işlenmiştir.

Kalifikasyonun **derinliği**, aracın rolünü sınıflandıran üç ölçüte (criteria) göre belirlenir:

- **Ölçüt 1:** Aracın çıktısı uçuş yazılımının bir parçası olur; dolayısıyla araç ürüne hata ekleyebilir. Tipik örnek, ürettiği kod ayrıca gözden geçirilmeyen kod üreticidir.
- **Ölçüt 2:** Araç bir doğrulama faaliyetini otomatikleştirir, bir hatayı gözden kaçırabilir **ve** çıktısı, kendi otomatikleştirdiği faaliyetin dışındaki doğrulama ya da geliştirme faaliyetlerini azaltmanın ya da kaldırmanın gerekçesi yapılır. Tipik örnek, sonucuna dayanılarak bazı testlerden vazgeçilen statik analiz ya da ispat aracıdır.

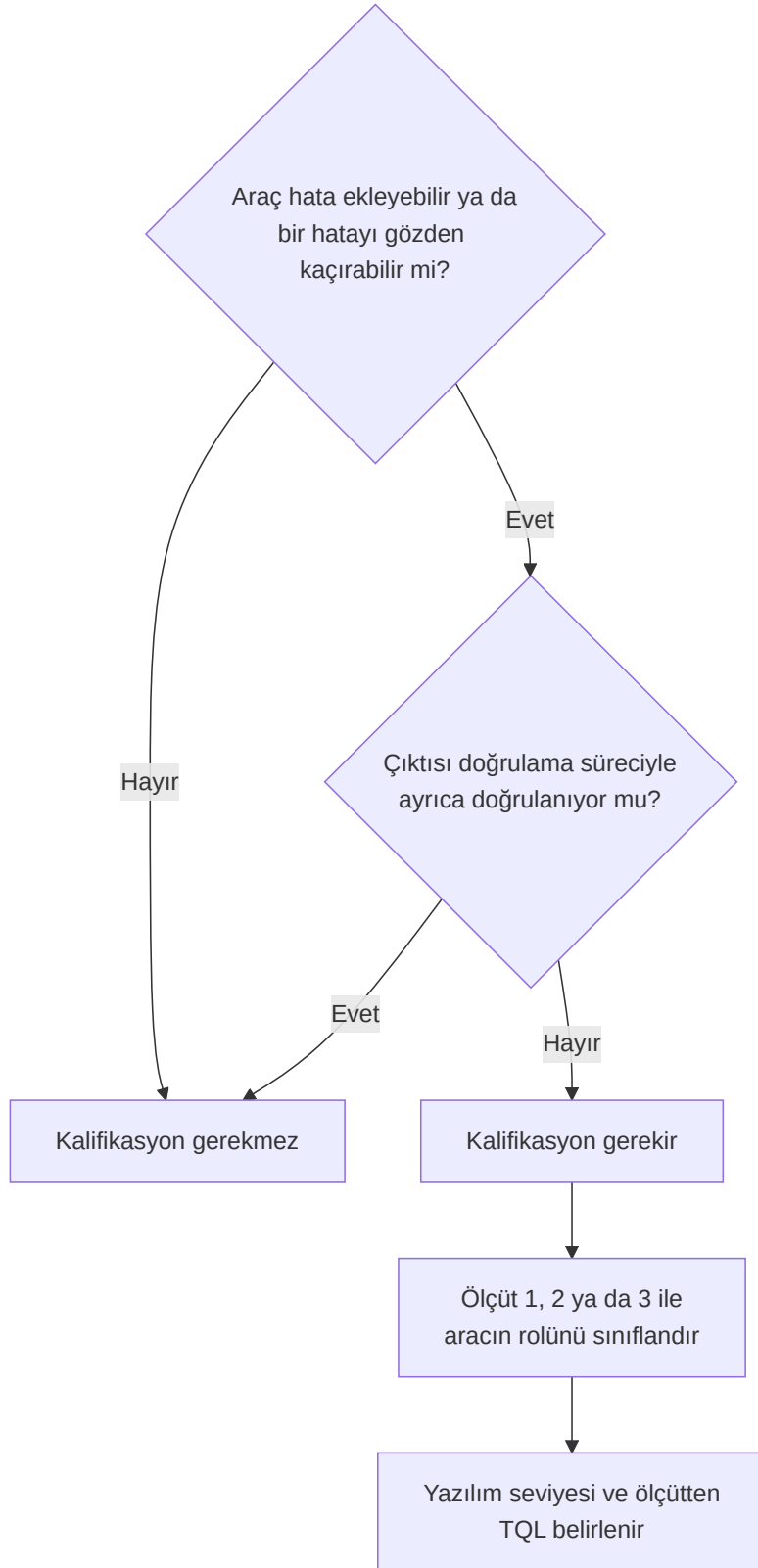
- **Ölçüt 3:** Araç, kullanım amacı içinde bir hatayı gözden kaçırabilir; ancak çıktısı başka bir faaliyetin azaltılmasına gerekçe yapılmaz. Tipik örnekler: yapısal kapsam analizi (structural coverage analysis) aracı, kodlama standardı denetleyicisi, test sonuçlarını beklenen değerlerle karşılaştıran araç.

Ölçütler sırayla, 1'den 3'e doğru değerlendirilir: araç ancak öncekine girmiyorsa sonrakine bakılır. Okurun en çok zorlandığı ayırım Ölçüt 2 ile Ölçüt 3 arasındadır ve aracın türüne değil, sonucundan ne kadar kredi alındığına bağlıdır. Sıfıra bölme gibi çalışma zamanı hatalarının bulunmadığını gösteren bir statik analiz aracını düşünün. Sonuç yalnızca kaynak kodun doğruluğu ve tutarlılığına ilişkin kanıt olarak kullanılıyorsa araç Ölçüt 3'tedir. Aynı sonuç, ilgili gürbüzlük testlerinin bir kısmını yapmamanın gerekçesi de yapılıyorsa araç artık kendi otomatikleştirmede bir faaliyeti azaltmaktadır ve Ölçüt 2'ye geçer. Biçimsel analiz araçlarında bu durum sık görülür (bkz. [16. DO-333 ve Biçimsel Yöntemler](#)).

Ölçüt, yazılım seviyesiyle (software level) birleştirilerek **araç kalifikasyon seviyesi (tool qualification level, TQL)** bulunur. TQL-1 en sıkı, TQL-5 en hafif seviyedir:

Yazılım seviyesi	Ölçüt 1	Ölçüt 2	Ölçüt 3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

Karar akışı özetle şöyledir:



Pratikte en sık yapılan hata, aracı olduğundan ağır sınıflandırmaktır. Örneğin bir test otomasyon aracı, sonuçları bir mühendis tarafından ayrıca gözden geçiriliyorsa hiç kalifikasyon gerektirmeyebilir. Tersi hata da görülür: bir betik "küçük" diye göz ardı edilir; oysa doğrulama

kanıtı üreten ve çıktısı kontrol edilmeyen üç satırlık bir betik bile kalifikasyon değerlendirmesine girmelidir.

DO-330 kalifikasyon süreci

DO-330'un temel fikri, aracın da kendi başına bir yazılım ürünü gibi ele alınmasıdır: aracın bir yaşam döngüsü, planları, gereksinimleri, doğrulama faaliyetleri, konfigürasyon yönetimi (configuration management) ve kalite güvencesi (quality assurance) vardır. Ancak buradaki hedef uçuşa elverişlilik değil, aracın **belirlenen kullanım amacı için** güvenilir olduğunun gösterilmesidir.

Araç operasyonel gereksinimleri

Sürecin merkezinde **araç operasyonel gereksinimleri (tool operational requirements, TOR)** durur. TOR, aracın projede *nasıl kullanılacağını* tanımlar:

- aracın hangi girdilerle, hangi ortamda ve hangi ayarlarla çalıştırılacağı,
- her kullanım senaryosunda aracın üretmesi beklenen çıktı ve davranış,
- kullanım kısıtları (desteklenmeyen dil yapıları, yasaklı seçenekler),
- aracın algılaması gereken anormal çalıştırma kipleri ve tutarsız girdiler ile bunlara vereceği tepki (bu kalem TQL-5'te aranmaz).

TOR bilinçli olarak kullanım odaklıdır: bir aracın yüzlerce özelliği olabilir, ama yalnızca projede fiilen kredi alınan işlevler kalifiye edilir. DO-178C bunun koşulunu da koyar: bu işlevlerin aracın öteki işlevlerinden olumsuz etkilenmediği, yani aralarında koruma (protection) bulunduğu gösterilmelidir. Bu, hem işi sınırlar hem de "araç her şeyi yapar" gibi savunulamaz iddialardan korur.

TOR, **araç gereksinimleri (tool requirements)** ile karıştırılmamalıdır. TOR kullanıcının gözünden yazılır ve "bu projede araçtan ne bekliyoruz?" sorusunu yanıtlar. Araç gereksinimleri ise geliştiricinin gözünden, aracın işlevlerini onu geliştirmeye ve test etmeye yetecek ayrıntıda tanımlar. Birincisi her kalifikasyonda vardır; ikincisi ancak aracın nasıl geliştirildiğine de bakılan seviyelerde istenir.

Yaşam döngüsü faaliyetleri de buna koşut olarak ikiye ayrılır:

- **Araç geliştirme ve doğrulama faaliyetleri:** araç gereksinimlerinin yazılması, tasarım, kodlama, entegrasyon ve aracın bu gereksinimlere karşı gözden geçirme, analiz ve testle doğrulanması. Daha sıkı TQL'lerde (TQL-1 – TQL-3) araç için de gereksinim-tasarım-kod izlenebilirliği beklenir.

- **Operasyonel faaliyetler:** TOR'un yazılması, aracın kullanım ortamına kurulması ve **operasyonel doğrulama ve geçerleme (tool operational verification and validation)**. Burada araç gerçek kullanım ortamında TOR senaryolarıyla çalıştırılır; ayrıca TOR'un, aracın yerine geçtiği faaliyetin ihtiyacını gerçekten karşıladığı gösterilir. Bu adım her TQL'de vardır.

TQL'e göre beklentiler

TQL sıkılaştıkça (TQL-5'ten TQL-1'e doğru) hem faaliyetlerin sayısı hem de bağımsızlık (independence) beklentisi artar. Seviyeler arasındaki farkı anlamının kolay yolu, her seviyenin bir öncekine ne eklediğine bakmaktır:

TQL	Hangi durumda	Kabaca beklenen
TQL-5	Ölçüt 3 (her seviye); Ölçüt 2 (Seviye C, D)	TOR, kurulum, operasyonel doğrulama ve geçerleme; araç için sınırlı kapsamda konfigürasyon yönetimi (tanımlama, arşivleme) ve kalite güvencesi, otoriteyle irtibat. Aracın nasıl geliştirildiğine bakılmaz
TQL-4	Ölçüt 2 (Seviye A, B); Ölçüt 1 (Seviye D)	Ek olarak araç planları, araç gereksinimleri ve araç mimarisi; gereksinimlerin gözden geçirilmesi, aracın bu gereksinimlere karşı test edildiğinin ve gereksinimlerin testlerle kapsandığının kanıtı; temel çizgi, problem raporlama ve değişiklik kontrolü
TQL-3	Ölçüt 1 (Seviye C)	Ek olarak araç geliştirme standartları, düşük seviyeli araç gereksinimleri ve kaynak kodun doğrulanması; aracın kendi kodu için satır kapsama (statement coverage) düzeyinde yapısal kapsam ile veri ve kontrol bağlaşımları (data and control coupling) analizi
TQL-2	Ölçüt 1 (Seviye B)	Ek olarak karar kapsama (decision coverage) ve doğrulama hedeflerinin bir kısmında bağımsızlık
TQL-1	Ölçüt 1 (Seviye A)	Ek olarak değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC), dış bileşenlere (external components) ilişkin analiz ve en geniş bağımsızlık; Seviye A yazılım geliştirmeye yaklaşan bir titizlik

Tablo yön göstermek içindir; hangi hedefin hangi TQL'de ve hangi bağımsızlıkla arandığı DO-330'un hedef tablolarından okunur. Projeler açısından en çok maliyet farkı yaratan basamak TQL-5 ile TQL-4 arasındadır. TQL-5'te araç bir kara kutu gibi ele alınır ve işin neredeyse tamamını kullanıcı kendi ortamında yapabilir. TQL-4'ten itibaren aracın *nasıl geliştirildiğini* gösteren veri gerekir; bu veri araç üreticisinde yoksa ya da paylaşılmıyorsa kalifikasyon çok güçleşir. Bu yüzden ekipler çoğu zaman aracı kullanım biçimini daraltarak Ölçüt 3'te tutmayı, yani araç sonucunu başka bir faaliyeti azaltmanın gerekçesi yapmamayı seçer.

Kalifikasyon verisi

Başlıca veriler ve hangi seviyelerde ayrı birer belge olarak beklendikleri şöyledir:

Veri	İngilizce adı	İçeriği	Hangi TQL'de
Araca özgü bilgi (PSAC, SECI ve SAS içinde)	Tool-specific information	Aracın rolü, ölçüt ve TQL gerekçesi; aracın sürüm tanımı; proje sonunda kalifikasyonun durumu	Tümü
Araç operasyonel gereksinimleri (TOR)	Tool Operational Requirements	Kullanım senaryoları, ortam, kısıtlar	Tümü
Araç kurulum raporu	Tool Installation Report	Kullanım ortamı ve kurulan araç sürümü; SECI içinde de verilebilir	Tümü
Operasyonel doğrulama ve geçerleme sonuçları	Tool Operational Verification and Validation Results	TOR senaryolarının kullanım ortamında koşulmasına ait kayıtlar	Tümü
Araç kalifikasyon planı	Tool Qualification Plan (TQP)	Kalifikasyon faaliyetleri, üretilecek veri, sorumluluklar	TQL-1 – TQL-4
Araç gereksinimleri	Tool Requirements	Aracın işlevlerinin doğrulanabilir tanımı	TQL-1 – TQL-4
Araç tasarım tanımı ve kaynak kod	Tool Design Description, Tool Source Code	Mimari, düşük seviyeli gereksinimler, kod	Mimari TQL-1 – TQL-4; düşük seviyeli gereksinimler, kaynak kod ve bunların doğrulanması TQL-1 – TQL-3
Araç doğrulama sonuçları	Tool Verification Results	Araç gereksinimlerine karşı testler, gözden geçirme ve analiz kayıtları	TQL-1 – TQL-4
Araç konfigürasyon indeksi	Tool Configuration Index (TCI)	Aracın ve ortamının tam sürüm tanımı	TQL-1 – TQL-4
Araç başarı özeti	Tool Accomplishment Summary (TAS)	Planlananla yapılanın karşılaştırması, bilinen problemler, uyum beyanı	TQL-1 – TQL-4

TQL-5'te ayrı bir TQP, TCI ve TAS beklenmez; karşılıkları projenin kendi verisinin araca özgü kısımlarıdır: plan bilgisi PSAC'ta, aracın sürüm tanımı yazılım yaşam döngüsü ortam konfigürasyon indeksinde (Software Life Cycle Environment Configuration Index, SECI), sonuç ise yazılım başarı özetinde (Software Accomplishment Summary, SAS) verilir. Bu, "TQL-5'te yazacak bir şey yok" demek değildir: aracın kimliği ve kullanım amacı, araçtan alınmak istenen kredi, TQL ve gerekçesi, sorumluluklar, kullanım ortamı ve kalifikasyon verisine atıf yine kayıt altında olmalıdır; isteyen proje bunları TQL-5'te de ayrı belgeler olarak düzenleyebilir.

Araç geliştiricisi ve araç kullanıcısı

Sorumluluk paylaşımı da DO-330'un önemli bir katkısıdır. Belge, özellikle ticari hazır (commercial off-the-shelf, COTS) araçlar için **araç geliştiricisi** ile **araç kullanıcısı** rollerini ayırır:

- **Araç geliştiricisi**, araç gereksinimlerini, geliştirme verilerini ve aracın kendi doğrulama kanıtını sağlar. Ticari araçlarda bu rol araç üreticisindedir.
- **Araç kullanıcısı** (yani sertifikasyon başvurusunu yapan proje), TQL'i belirler, TOR'u yazar, aracı kendi ortamında operasyonel olarak doğrular ve kalifikasyon iddiasının bütününden otoriteye karşı sorumlu kalır.

Ticari araç çoğu zaman belirli bir projenin TOR'u ortada yokken geliştirilir. DO-330 bu boşluğu bir **geliştirici TOR'u (developer-TOR)** ile kapatır: geliştirici aracın öngörülen ortamlarını, girdi-çıktılarını ve işlevlerini tanımlayıp doğrulamasını buna karşı yapar; kullanıcı kendi TOR'unu bunun üzerine projeye özgü kullanım ve kısıtlarla kurar.

Kritik nokta şudur: araç üreticisinden ne kadar hazır veri alınır alınsın, kalifikasyon sorumluluğu devredilemez. Kullanıcı, aracın kendi projesindeki kullanım biçimini kapsayan kanıtı her durumda kendisi tamamlamalıdır.

Özel kalifikasyon konuları

Araç belirlenimciliği

Araç belirlenimciliği (tool determinism), aynı girdi ve aynı yapılandırmayla çalıştırılan aracın her seferinde aynı çıktıyı üretmesi beklentisidir. Bu önemlidir, çünkü kalifikasyon kanıtı belirli bir çıktı üzerinden kurulur; araç bir dahaki çalıştırmada farklı çıktı üretiyorsa o kanıt neyi temsil ettiğini yitirir.

DO-178B bu beklentiyi katı biçimde koyuyor, yalnızca aynı ortamda aynı girdiyle hep aynı çıktıyı veren araçların kalifiye edilebileceğini söylüyordu. Modern araçlarda bit düzeyinde aynılık her zaman gerçekçi değildir: bir test aracı rapora zaman damgası gömebilir, bir analiz aracı bulgularını her çalıştırmada farklı sırada dökebilir. DO-330 bu yüzden bir ayrım yapar. Çıktısı uçuş yazılımının parçası olan ve hata ekleyebilen araçta (Ölçüt 1; örneğin kod üretici) beklenti katı kalır: kalifiye edilen işlevlerin aynı ortamda aynı girdiyle aynı çıktıyı ürettiği gösterilir. Çıktısı beklenen sınırlar içinde değişebilen diğer araçlarda ise gösterilmesi gereken iki şeydir: farklılık çıktının kullanım amacını olumsuz etkilememektedir ve çıktının doğruluğu belirlenebilmektedir. FAA'nın DO-178B dönemindeki uygulama talimatı (Order 8110.49) aynı yorumu daha önce yapmıştı: aynı girdi için ortaya çıkabilecek çıktı çeşitlerinin hepsinin doğru olduğu gösterilebiliyorsa araç, kalifikasyon açısından belirlenimci kabul edilir. Pratikteki karşılığı **işlevsel eşdeğerlik** düzeyinde belirlenimciliktir: çıktılar biçimce farklılaşsa bile anlamları aynı olmalı ve bu, analizle gösterilebilmelidir. Değerlendirirken şunlara bakılır:

- Aracın çıktısını etkileyen tüm girdiler (kaynak dosyalar, seçenekler, ortam değişkenleri, araç sürümü) konfigürasyon yönetimi altında mı?
- Aynı girdi kümesiyle tekrar üretilebilirlik test edilmiş mi?
- Kabul edilebilir çıktı farklılıkları tanımlanmış ve gerekçelendirilmiş mi?

Bu esneklik yalnızca çıktısı ürüne girmeyen kalifiye araçlar içindir; çalıştırılabilir nesne kodunun arşivlenen kaynaktan ve kayıtlı ortamdan tutarlı biçimde yeniden üretilebilmesi beklentisini gevşetmez. Derleyiciden her derlemede aynı çıktıyı beklemek, kalifiye edilmese de geçerli bir seçim ölçütüdür (bkz. [8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#) ve [10. Yazılım Konfigürasyon Yönetimi](#)).

Önceden kalifiye edilmiş araçların yeniden kullanımı

Bir araç bir projede kalifiye edildiyse, bu kanıt sonraki projelerde büyük emek tasarrufu sağlayabilir; ancak otomatik olarak geçerli olmaz: DO-178C'ye göre araç, PSAC'ta beyan edildiği belirli sistem için kalifiye edilir. Yeniden kullanım değerlendirmesinde üç soru sorulur:

1. **Araç ve ortamı aynı mı?** Sürüm, yamalar ve yapılandırma birebir aynı değilse ya da aracın çalıştığı ortam (örneğin işletim sistemi) öncekine eşdeğer değilse fark analizi gerekir; küçük görünen bir sürüm atlaması bile davranışı değiştirebilir.
2. **Kullanım biçimi aynı mı?** Yeni projenin TOR'u öncekinin kapsamı içinde kalıyor mu? Yeni bir dil özelliği, yeni bir hedef işlemci ya da yeni bir seçenek kullanılıyorsa, kalifikasyon kanıtının o kısmı yeniden üretilmelidir.
3. **Gereken TQL aynı ya da daha hafif mi?** Önceki proje TQL-5 için kanıt üretmişse, yeni projede TQL-4 gerekiyorsa aradaki fark kapatılmalıdır.

Üç yanıt da olumluysa ve onaylı kalifikasyon verisine erişilebiliyorsa bu veri, yeni projenin planlarında gerekçesiyle birlikte referans gösterilir; önceki kalifikasyondan bu yana açılmış ya da açık kalmış araç problem raporları da yeniden değerlendirilir. Otorite ile katılım aşaması (Stage of Involvement, SOI) denetimlerinde, özellikle planların ele alındığı ilk denetimde bu yaklaşımın erken paylaşılması sürprizleri önler (bkz. [SW SOI-1](#)).

DO-178B döneminde kalifiye edilmiş araçlar ayrı bir geçiş sorusu doğurur. DO-178B araçları yalnızca iki türe ayırıyordu: geliştirme aracı (development tool) ve doğrulama aracı (verification tool). DO-178C bu ikilinin yerine üç ölçütü ve beş TQL'i koydu. DO-330'un sonundaki açıklayıcı soru-yanıtlar ile FAA AC 20-115D ve EASA AMC 20-115D eski türlerle yeni seviyeler arasında bir eşleme verir: geliştirme aracı Ölçüt 1'e karşılık gelir ve kalifiye edildiği yazılım seviyesine göre TQL-1 ile TQL-4 arasında bir seviye alır; doğrulama aracı ise yeni projedeki kullanımına göre Ölçüt 2 ya da Ölçüt 3'e girer ve TQL-4 ya da TQL-5 alır. Eşleme krediyi kendiliğinden vermez; istenen kredi PSAC'ta yazılır ve otoriteyle kararlaştırılır. Eski kalifikasyon yeni projede gereken TQL'i karşılıyorsa DO-178B'ye göre kurulmuş kalifikasyon süreci sürdürülebilir; araç ya da kullanım ortamı değiştiyse önce değişiklik etki analizi (change impact analysis) yapılır. Dikkat isteyen durum, eskiden doğrulama aracı olarak kalifiye edilmiş bir aracın yeni projede Ölçüt 2 kapsamında

Seviye A ya da B için kullanılmasıdır: gereken TQL-4, eski kalifikasyonun üretmediği araç geliştirme verisini ister ve araç DO-330'a göre yeniden kalifiye edilir. "Geliştirme aracı kalifikasyonu" deyimi gündelik dilde hâlâ yaşar; bugünkü karşılığı Ölçüt 1'dir.

Ticari araçların kalifikasyon paketleri

Birçok ticari araç üreticisi, DO-330 kanıtının geliştirici tarafını hazır sunan **kalifikasyon paketi (qualification kit)** satar. Tipik bir paket; araç gereksinimlerini, test durumlarını, test çalıştırma altyapısını ve şablon plan belgelerini içerir. Bu paketler değerlidir, ancak iki sınırı iyi anlamak gerekir:

- Paket, aracı **sizin ortamınızda ve sizin kullanım biçiminizde** doğrulamaz. Testlerin projenin gerçek ortamında (işletim sistemi, sürümler, seçenekler) koşulması ve sonuçların değerlendirilmesi kullanıcının işidir.
- Paket, TQL kararını ve TOR'u veremez; çünkü bunlar aracın projedeki rolüne bağlıdır. "Paketi satın aldık, araç kalifiye" cümlesi otorite nezdinde geçersizdir.

Doğru kullanım şudur: paketi bir hızlandırıcı olarak alın, kendi TOR'unuzu yazın, paketin kapsamadığı kullanım senaryolarını belirleyip ek testlerle kapatın ve tümünü kendi araç kalifikasyon planınız altında birleştirin.

Çalışılmış örnek: bir projenin araç envanteri

Kuralların nasıl işlediğini görmek için varsayımsal bir projenin araç listesine iki soruyu ve ölçütleri uygulayalım. Son iki sütun, aynı kullanımın Seviye A ve Seviye C yazılımında hangi TQL'e götürdüğünü gösterir:

Araç ve kullanım biçimi	Risk	Çıktısı ayrıca doğrulanıyor mu?	Ölçüt	Seviye A	Seviye C
Derleyici ve bağlayıcı	Hata ekleyebilir	Evet: nesne kodu gereksinim tabanlı testlerle hedef ortamda doğrulanır	—	Gerekmez	Gerekmez
Kod üretici; üretilen kod elle yazılmış kod gibi gözden geçirilip test ediliyor	Hata ekleyebilir	Evet	—	Gerekmez	Gerekmez
Aynı kod üretici; üretilen kodun gözden geçirmesi kaldırılmış	Hata ekleyebilir	Hayır	1	TQL-1	TQL-3
Yapısal kapsam aracı; raporu kapsam kanıtı olarak kullanılıyor	Gözden kaçırabilir	Hayır	3	TQL-5	TQL-5

Araç ve kullanım biçimi	Risk	Çıktısı ayrıca doğrulanıyor mu?	Ölçüt	Seviye A	Seviye C
Statik analiz aracı; sonucu hem kod doğruluğu kanıtı hem bazı gürbüzlük testlerini azaltmanın gerekçesi	Gözden kaçırabilir	Hayır	2	TQL-4	TQL-5
Test sonuçlarını beklenen değerlerle karşılaştırıp geçti/kaldı kararı veren betik; sonuçlara ayrıca bakılmıyor	Gözden kaçırabilir	Hayır	3	TQL-5	TQL-5

Tablodan üç ders çıkar. Birincisi, kod üretici iki ayrı satırda yer alır: aracı değil, kullanım biçimini sınıflandırıyoruz. Üretilen kodu elle yazılmış kod gibi ele almak ile aracı kalifiye edip bazı doğrulama adımlarından kredi almak arasındaki seçim [8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#) bölümünde, model tabanlı geliştirmedeki karşılığı [14. DO-331 ve Model Tabanlı Geliştirme ve Doğrulama](#) bölümünde tartışılır. Kalifiye bir kod üretici de kod düzeyindeki yapısal kapsam yükümlülüğünü kendiliğinden kaldırmaz.

İkincisi, statik analiz aracı test azaltma iddiasından vazgeçilirse Ölçüt 3'e ve Seviye A'da bile TQL-5'e iner. Üreticiden geliştirme verisi alınamayan araçlarda bu, çoğu zaman tek uygulanabilir yoldur; bedeli, azaltılmak istenen testlerin yine yapılmasıdır.

Üçüncüsü, en küçük araç da listededir. Yapısal kapsam aracı ile karşılaştırma betiği aynı ölçüte girer; birinin ticari ürün, diğerinin ekip içinde yazılmış birkaç satır olması sonucu değiştirmez (kapsam araçlarının kullanımı [9. Yazılım Doğrulama](#) bölümündedir).

Bu tablo aslında planlama aşamasında tutulması gereken **araç envanterinin** bir örneğidir. Her satırda en az şunlar bulunmalıdır: aracın adı ve sürümü, yaşam döngüsündeki rolü, çıktısını doğrulayan faaliyet (varsa), ölçüt, TQL ve kararın gerekçesi. Envanter PSAC'taki araç bilgisinin kaynağıdır; "kalifikasyon gerekmez" denen satırların gerekçesi de en az diğerleri kadar sorgulanır (bkz. [5. Yazılım Planlama](#)).

Bu bölümden akılda kalması gerekenler

- Araç hatası kanıta yansiyabilir; hata ekleyebilen ya da gözden kaçırabilen ve çıktısı ayrıca doğrulanmayan araç kalifikasyon gerektirir. Karar araca değil, kullanım biçimine göre verilir.
- Ölçütler ve TQL tablosu DO-178C'dedir; DO-330 ise bir teknoloji eki değil, seçilen TQL'de kalifikasyonun nasıl yapılacağını anlatan bağımsız bir belgedir.
- Kalifikasyon derinliğini, aracın rolünü tanımlayan üç ölçüt ile yazılım seviyesinin birleşimi (TQL-1'den TQL-5'e) belirler; Ölçüt 2 ile Ölçüt 3'ü ayıran, araç sonucunun başka bir faaliyeti azaltmanın gerekçesi yapılıp yapılmadığıdır.

- Süreç araç operasyonel gereksinimleri (TOR) etrafında kurulur; araç işlevleri arasındaki koruma gösterilebiliyorsa yalnızca projede fiilen kredi alınan işlevler kalifiye edilir. TQL-5 kullanım odaklı kanıtla yetinir; TQL-4 ve daha sıkı seviyeler aracın nasıl geliştirildiğini gösteren veriyi de ister.
- Araç üreticisinden hazır veri alınsa bile kalifikasyon sorumluluğu başvuru sahibinde kalır; operasyonel doğrulama kullanıcının kendi ortamında yapılır.
- Belirlenimcilik bir tasarruf yolu değil, kalifikasyon kanıtının geçerlilik koşuludur: çıktısı ürüne giren araç aynı girdiyle aynı çıktıyı üretmeli, diğer araçlarda farklılığın kullanım amacını etkilemediği ve çıktının doğruluğu gösterilebilmelidir.
- Yeniden kullanım ve ticari kalifikasyon paketleri emek tasarrufu sağlar; ancak araç, kullanım biçimi ve TQL farkları için projeye özgü analiz ister.

14. DO-331 ve Model Tabanlı Geliştirme ve Doğrulama

Model tabanlı geliştirme (model-based development), yazılım gereksinimlerinin ya da tasarımının metin yerine bir modelle ifade edilmesidir; DO-331 bu durumda modelin hangi yaşam döngüsü verisi sayılacağını, nasıl doğrulanacağını ve simülasyondan hangi koşullarda doğrulama kredisini alınabileceğini tanımlar. Bu bölüm model türlerini, planlama beklentilerini, simülasyon kredisini, model kapsam analizini ve model tabanlı akışta kullanılan araçları ele alır.

DO-331'in tam adı "Model-Based Development and Verification Supplement to DO-178C and DO-278A"dır; 2011'de DO-178C ile birlikte yayımlanmıştır ve EUROCAE karşılığı ED-218'dir. Bir ek (supplement) olduğu için tek başına okunmaz: DO-178C'nin bölüm yapısını izler, model kullanılan yerlerde ana belgenin metnine ekleme ve değişiklik yapar, hedef (objective) tablolarına da modele özgü hedefler ekler. Yazılımın modelle geliştirilmeyen kısımları için DO-178C olduğu gibi geçerlidir. Ekin ana fikri tek cümleyle şudur: model bir rahatlık aracı değil, izlenebilir ve doğrulanması gereken bir yazılım yaşam döngüsü verisidir (software life cycle data).

Model tabanlı geliştirme ve DO-331'in kapsamı

Model, davranışı soyut ama çalıştırılabilir biçimde ifade eder. Durum makineleri, blok şemaları ve veri akış diyagramları, uzun metin paragraflarının bıraktığı belirsizliği azaltır; ekip tasarım kararlarını erken görür, senaryoları simüle eder ve gereksinim uyumsuzluklarını daha kod yazılmadan fark edebilir. Modelden otomatik kod üretimi de elle kodlamadan kaynaklanan hata sınıflarını daraltır. Bu kazanımların bir bedeli vardır: model de metin kadar yanlış olabilir ve çalıştırılabildiği için "doğru görünme" gücü metinden fazladır. DO-331'in bütün düzeni, bu yanıltıcı güveni disipline bağlamak üzerine kuruludur.

Ek, model yazılım gereksinimlerinin ya da tasarımının yerini aldığı anda uygulanır; yani kod o modelden geliştiriliyorsa (elle ya da otomatik) ya da doğrulama o modele dayanıyorsa. Modeli kimin ürettiği bu sonucu değiştirmez: sistem mühendislerinin hazırladığı bir model yazılımın geliştirilmesine temel oluyorsa eke tabidir. Yalnızca kavram çalışması için kullanılan ve yaşam döngüsü verisi olarak sunulmayan bir model kapsam dışında kalır; ancak "açıklama amaçlı" denen bir modelden kod yazılıyorsa bu iddia geçerliliğini yitirir.

DO-331'in DO-178C'ye ekledikleri beş başlıkta toplanabilir; bölümün geri kalanı bu sırayı izler:

- **Model türleri:** modelin hangi yaşam döngüsü verisinin yerini aldığı ve neye karşı doğrulanacağı.
- **Planlama ve modelleme standartları:** modelin planlarda beyanı, modelleme kuralları, model öge kütüphaneleri ve model öğelerinin izlenebilirliği (traceability).

- **Model simülasyonu:** simülasyonun hangi doğrulama hedeflerine, hangi koşullarda kanıt sayılabileceği.
- **Model kapsam analizi:** gereksinim tabanlı doğrulamanın modeli ne ölçüde uyardığı ve uyarılmayan öğelerin nasıl çözümleneceği.
- **Araçlar:** simülatörün, kapsam aracının ve kod üreticinin (code generator) çıktısına ne zaman güvenilebileceği.

Model türleri

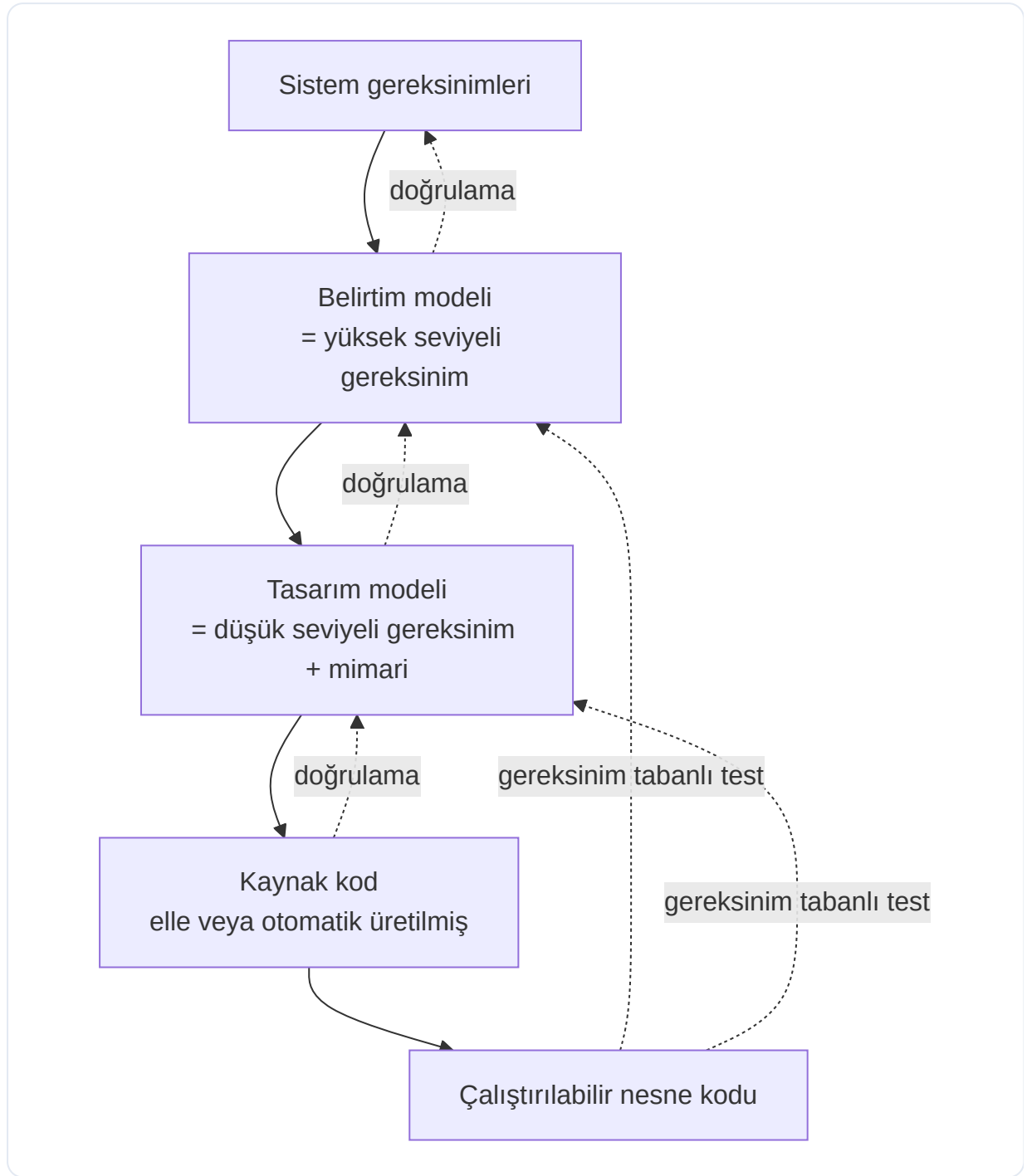
DO-331'in en temel kavramsal katkısı, "model" sözcüğünün tek bir şey ifade etmediğini netleştirmesidir. Bir model, yaşam döngüsünde hangi iş ürününün yerini alıyorsa o iş ürünü gibi ele alınır. Bu bakışla iki ana tür ayrılır:

- **Belirtim modeli (specification model):** Yüksek seviyeli gereksinimin (high-level requirement) yerine geçer. Yazılımın *ne* yapması gerektiğini tanımlar; iç veri yapısı, iç veri akışı ya da kontrol akışı gibi tasarım ayrıntısı içermez. Sistem gereksinimlerinden izlenebilir olmalı ve tıpkı metinsel bir gereksinim gibi gözden geçirilip doğrulanmalıdır.
- **Tasarım modeli (design model):** Düşük seviyeli gereksinimin (low-level requirement) ve/veya yazılım mimarisinin yerine geçer. Yazılımın *nasıl* gerçekleştirileceğini tanımlar; veri akışları, durum makineleri, blok şemaları gibi gerçekleştirmeye yakın detay içerir. Kodun doğrudan kendisinden yazıldığı ya da üretildiği model tasarım modelidir; otomatik kod üretimi ([8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#)) bu modeli girdi alır.

Bu ayrımın pratikte kritik bir sonucu vardır: **bir model kendi kendisinin doğrulama referansı olamaz.** Model hangi seviyeyi temsil ediyorsa, ona uygunluk bir üst seviyedeki gereksinime göre gösterilir. Örneğin tasarım modeli kullanılıyorsa, onun üstünde modelden bağımsız gereksinimler bulunmalı ve model bu gereksinimlere karşı doğrulanmalıdır. Aynı modeli hem gereksinim hem tasarım ilan edip "model modele uygundur" demek, doğrulama zincirini kısa devre yapar; DO-331 bu tuzağı açıkça kapatır: bir model aynı anda hem belirtim hem tasarım modeli sayılamaz.

Model türü	Karşılık geldiği DO-178C iş ürünü	Neye karşı doğrulanır	Tipik içerik
Belirtim modeli	Yüksek seviyeli gereksinimler	Yazılıma tahsis edilmiş sistem gereksinimleri	Girdi/çıktı davranışı, modlar, işlevsel kurallar
Tasarım modeli	Düşük seviyeli gereksinimler + yazılım mimarisi	Modelin geliştirildiği gereksinimler (çoğunlukla yüksek seviyeli gereksinimler)	Durum makineleri, veri/kontrol akışı, algoritmalar

Modelin türü, doğrulama zincirinin şeklini doğrudan belirler. Aşağıdaki diyagram iki modelin birlikte kullanıldığı düzeni gösterir; bu, olası düzenlerden yalnızca biridir.



Çalıştırılabilir nesne kodu (executable object code) yalnızca yüksek seviyeli gereksinimlere karşı test edilmez; Seviye A, B ve C'de düşük seviyeli gereksinimlere, yani bu düzende tasarım modeline dayalı testler de beklenir.

Projede hangi türün kullanıldığı planlama aşamasında açıkça beyan edilmelidir. Deneyimde en sık görülen sorun, ekibin tek bir Simulink/SCADE modelini fiilen hem gereksinim hem tasarım olarak kullanması, ama bunu hiçbir planda adlandırmamasıdır. Sertifikasyon otoritesi ilk katılım aşaması (Stage of Involvement, SOI) denetiminde bu soruyu sorar; cevap belirsizse tüm izlenebilirlik ve doğrulama argümanı yeniden kurgulanmak zorunda kalır. Modelin türünü baştan yazılı hâle getirmek, sonradan zincir onarmaktan çok daha ucuzdur.

Modelin geliştirildiği gereksinimler ve tipik düzenler

"Bir üst seviye" her projede aynı veri değildir. DO-331 bunun için daha genel bir kavram kullanır: **modelin geliştirildiği gereksinimler** (requirements from which the model is developed). Model neyin üzerine kurulduysa ona karşı doğrulanır, simülasyon senaryoları oradan türetilir ve model öğeleri oraya izlenir. Ek, modelin yaşam döngüsüne nereden girdiğine göre değişen örnek düzenler verir; pratikte karşılaşılan başlıcaları şöyle özetlenebilir:

Düzen	Modelin geliştirildiği gereksinimler	Model ve yerini aldığı veri	Dikkat edilecek nokta
Metinsel gereksinim + tasarım modeli	Metinsel yüksek seviyeli gereksinimler	Tasarım modeli: düşük seviyeli gereksinimler ve mimari	En yaygın düzen; gereksinimler modele bakılarak değil, modelden bağımsız yazılmalıdır
Belirtim modeli + tasarım modeli	Belirtim modeli için sistem gereksinimleri; tasarım modeli için belirtim modeli	İki ayrı model, iki ayrı veri	İkinci model birincinin kopyası olmamalı; iki model ayrı ayrı doğrulanır
Belirtim modeli + metinsel tasarım	Sistem gereksinimleri	Belirtim modeli: yüksek seviyeli gereksinimler	Tasarım ve kod için DO-178C olduğu gibi uygulanır
Sistem gereksinimi + tasarım modeli	Yazılıma tahsis edilmiş sistem gereksinimleri	Tasarım modeli; yazılım ya da sistem süreçlerinde üretilmiş olabilir	Arada ayrı bir yazılım gereksinim katmanı yoktur; sistem ve yazılım verisinin sınırı planlarda çizilmelidir

Son düzen, kontrol yasalarını sistem mühendislerinin modellediği projelerde sık görülür ve en çok soru doğuran düzendir. Model sistem ekibinde doğar, yazılım ekibi ondan kod üretir; aynı veri iki tarafa da hizmet ettiği için kimin neyi doğruladığı kolayca belirsizleşir. Bu durumda iki şey netleştirilmelidir. Birincisi, modelin üstündeki gereksinimler yüksek seviyeli gereksinimlerin yükünü taşır: modelden bağımsız yazılmış, gözden geçirilmiş ve test durumu (test case) türetililecek kadar ayrıntılı olmalıdır. "Kontrol yasası modeldeki gibidir" diyen bir gereksinim, modeli yine kendi referansı yapar. İkincisi, modelin sahipliği, konfigürasyon kontrolü ve değişiklik akışı planlarda tanımlanmalıdır: sistem ekibinin modelde yaptığı her değişiklik artık bir yazılım tasarım değişikliğidir. Sistem ve yazılım süreçleri arasındaki veri alışverişi [2. Sistem Bağlamında Yazılım](#), gereksinim katmanlarının ayrımı ise [6. Yazılım Gereksinimleri](#) bölümünde anlatılmıştır.

Planlama ve modelleme standartları

Model tabanlı akışın kararları planlara yazılır ([5. Yazılım Planlama](#)). Yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC) hangi yazılım bileşenlerinde model kullanıldığını, her modelin türünü ve yerini aldığı veriyi, simülasyondan kredi istenip istenmediğini ve araçların kalifikasyon durumunu beyan eder. Yazılım geliştirme planı (Software Development Plan, SDP) modelleme ortamını — araç, sürüm, kod üretim ayarları — tanımlar ve uygulanacak modelleme standartlarına atıf yapar. Yazılım doğrulama planı (Software Verification

Plan, SVP) modelin nasıl doğrulanacağını, simülasyon ortamını ve hedeften farklarını, model kapsam ölçütlerini ve kapsam boşluklarının nasıl çözümleneceğini yazar. Planlama denetiminde ([SW SOI-1](#)) plan ve standartlara yöneltilen her soru, model kullanan projede model için de yanıtlanır.

Yazılım modelleme standartları (software model standards), gereksinim, tasarım ve kodlama standartlarının yanına eklenen dördüncü standart türüdür. Modelleme dilleri ve araçları, emniyet-kritik kullanım için uygun olmayan pek çok özellik barındırır; standart bunların hangilerine izin verildiğini söyler. Tipik içerik:

- izin verilen ve yasaklanan model öğeleri ile araç ayarları (örneğin örtük veri tipi dönüşümü, yürütme sırasını araca bırakan yapılar, değişken adimli çözücü (solver));
- adlandırma, yerleşim ve karmaşıklık sınırları (iç içe düzey sayısı, bir diyagramdaki öğe sayısı, bir durum makinesindeki geçiş sayısı);
- gereksinim ya da tasarım ifade eden öğelerin, açıklama notu gibi bilgi amaçlı öğelerden nasıl ayırt edileceği;
- izlenebilirliğin hangi birimle (blok, alt sistem, durum, geçiş) kurulacağı ve türetilmiş gereksinimlerin nasıl işaretleneceği.

Belirtim modeli ile tasarım modeli farklı soyutlama düzeyinde olduğundan, ikisi de kullanılıyorsa her biri için ayrı kurallar gerekir. Standarda uygunluk bir doğrulama hedefidir; kural denetimi araçla otomatikleştirilebilir, ancak aracın denetlemediği kurallar gözden geçirmeyle kapatılır.

Model öğe kütüphaneleri (model element libraries) ayrı bir dikkat ister. Bir kütüphane bloğu modele konduğunda davranışı modelin, üretilen kodu da uçuş yazılımının parçası olur. Bu yüzden kullanılan her kütüphane öğesi, modelin yazılım seviyesine uygun biçimde geliştirilmiş ve doğrulanmış olmalıdır: ne yaptığını tanımlayan bir gereksinimi, doğrulama kanıtı ve konfigürasyon kontrolü vardır. Bu güvenceye sahip olmayan ya da projede kullanılmayan öğeler ya kütüphaneden çıkarılır ya da modelleme standardıyla yasaklanır. Araçla birlikte gelen hazır blokları "aracın parçası" sayıp doğrulama dışında bırakmak, derleyici kütüphanelerini hesaba katmamakla aynı hatadır.

İzlenebilirlik model öğesi düzeyinde kurulur: her öğe ya da anlamlı öğe grubu, modelin geliştirildiği gereksinimlerden birine izlenir. Hiçbir gereksinime izlenemeyen bir model öğesi için iki olasılık vardır. Öğe gerekliyse — örneğin bir sayacın taşmasını önleyen sınırlama — bir türetilmiş gereksinimdir (derived requirement): gerekçesiyle kaydedilir ve sistem süreçlerine, sistem emniyet değerlendirmesi dahil, geri bildirilir. Gerekli değilse istenmeyen işlevdir ve modelden kaldırılır. Tasarım modeli için beklenen tasarım niteliklerinin kendisi [7. Yazılım Tasarımı](#) bölümündekilerle aynıdır; model yalnızca gösterimi değiştirir.

Model simülasyonu ve kredisi

Model simülasyonu (model simulation), modelin bir simülasyon ortamında çalıştırılarak davranışının gözlemlenmesidir. DO-331 bunu yalnızca bir geliştirme kolaylığı olarak değil, belirli koşullar altında **doğrulama kredisi (verification credit)** alınabilecek, tanımlı ve disiplinli bir doğrulama yöntemi olarak tanıır. "Kredi" burada şu anlama gelir: normalde gözden geçirme, analiz ya da test ile kapatılacak bir doğrulama hedefinin, simülasyon sonuçlarına dayanarak kapatılması.

Kredi alınabilmesi için simülasyon, serbest bir "modeli çalıştırıp bakma" etkinliği olmaktan çıkarılmalıdır. Pratikte bu şu disiplini gerektirir:

- Simülasyon senaryoları, modelin geliştirildiği gereksinimlerden türetilir; yani gereksinim tabanlı test (requirements-based testing) mantığıyla yazılır.
- Her senaryonun beklenen sonucu ve geçti/kaldı ölçütü önceden tanımlanır; beklenen sonuç modelin çıktısından değil, gereksinimden üretilir.
- Senaryolar, prosedürler ve sonuçlar konfigürasyon yönetimi altına alınır ve gereksinimlere izlenebilirlik kurulur.
- Simülasyon senaryolarının kendisi de gözden geçirilir; yanlış senaryo, yanlış güven üretir.

Son madde bir tavsiye olmanın ötesindedir: kredi alınan simülasyonda senaryolar, prosedürler ve sonuçlar doğrulama verisidir ve DO-331 bunların doğruluğunu ayrı hedefler olarak arar; tıpkı test durumlarının ve test sonuçlarının doğruluğunun aranması gibi.

Simülasyonun neye kredi sağlayıp neye sağlayamayacağı, en çok yanlış anlaşılan konudur. Kaba bir özet:

Doğrulama hedefi	Simülasyon kredisi	Not
Modelin, geliştirildiği gereksinimlere uygunluğu	Evet, uygun koşullarda	Senaryolar o gereksinimlerden türetilmişse
Modelin doğruluğu, tutarlılığı ve algoritmalarının doğruluğu	Kısmen	Gözden geçirme ve analizle birlikte
Modelin modelleme standartlarına uygunluğu ve izlenebilirliği	Hayır	Gözden geçirme ya da analizle gösterilir
Çalıştırılabilir nesne kodunun yüksek seviyeli gereksinimlere uygunluğu	Kısmen, sıkı koşullarda	Yalnızca tasarım modeli simülasyonu; testle ve fark analiziyle birlikte
Çalıştırılabilir nesne kodunun düşük seviyeli gereksinimlere (tasarım modeline) uygunluğu	Hayır	Senaryolar modelin üstündeki gereksinimlerden türetilir; modele dayalı testin yerini tutmaz
Kod üzerindeki yapısal kapsam analizi (structural coverage analysis)	Model kapsamıyla karşılanmaz	Yapısal kapsam kod üzerinde ölçülür; model üzerinde ölçülen kapsam onun yerine geçmez

Doğrulama hedefi	Simülasyon kredisi	Not
Hedef donanım ile uyumluluk (zamanlama, bellek, G/Ç) ve donanım/yazılım entegrasyonu	Hayır	Yalnızca hedef üzerinde gösterilebilir

Çalıştırılabilir nesne kodunun doğrulanması esas olarak hedef ortamda testle yapılır ([9. Yazılım Doğrulama](#)). DO-331, tasarım modeli simülasyonunun bu test hedeflerinin bir kısmına, testle ve özel analizlerle birlikte kullanıldığında *kısmi* katkı verebileceğini kabul eder. Böyle bir kredi isteği en azından şunları göstermelidir: simüle edilen model, kodun üretildiği modelin aynısıdır; senaryolar modelin geliştirildiği gereksinimlerden türetilmiştir; simülasyon ortamı ile hedef arasındaki farklar analiz edilmiştir; hangi test hedefinin hangi ölçüde simülasyonla desteklendiği planlarda yazılmış ve otoriteyle kararlaştırılmıştır. Kredi iki yerde kesin olarak durur. Simülasyon senaryoları modelin üstündeki gereksinimlerden türetilir ve çalıştırılan şey kod değil modeldir; bu yüzden düşük seviyeli gereksinimlere, yani tasarım modeline dayalı testlerin ve bu testlerin kapsamının yerini simülasyon tutamaz. Hedef bilgisayarla uyumluluğu ve donanım/yazılım entegrasyonunu gösteren testler de her durumda hedefte yapılır.

Hedefe ilişkin sınırın nedeni **temsil yeteneğidir**: simülasyon ortamı, hedef ortamın yalnızca bir yaklaşıklamasıdır. Masaüstü işlemcinin kayan nokta davranışı, çözücünün zaman adımı, sıralama ve zamanlama varsayımları, donanım giriş/çıkışlarının idealize edilmesi gibi farklar, modelde doğru görünen davranışın hedefte farklılaşmasına yol açabilir. Bu yüzden krediye başvururken simülasyon ortamı ile hedef ortam arasındaki farklar tanımlanmalı ve bu farkların doğrulanan özellik üzerinde etkisi olmadığı gerekçelendirilmelidir. Gerekçelendirilemeyen her fark, o hedefin hedef ortam testiyle kapatılması gerektiği anlamına gelir.

Deneyimden bir uyarı: simülasyon geri beslemesi hızlı olduğu için ekipler doğal olarak ona güvenmeye başlar ve hedef testlerini "formalite" gibi görme eğilimi oluşur. Zamanlama, kesme etkileşimi ve donanım arayüzü kaynaklı hatalar ise çoğunlukla ancak hedefte ortaya çıkar. Simülasyon, hedef testinin *ikamesi* değil, ona gelmeden önce hata sayısını düşüren bir *ön filtresi* olarak konumlandırılmalıdır.

Model kapsam analizi

Model kapsam analizi (model coverage analysis), gereksinim tabanlı doğrulamanın — simülasyon senaryolarının ya da testlerin — tasarım modelini ne ölçüde uyardığını ölçer. Sorduğu soru şudur: modelin geliştirildiği gereksinimlerden türetilen senaryolar koşulduğunda, modelin ifade ettiği davranışlardan hangileri hiç çalışmadı? Uyarılmayan bir geçiş ya da dal, o davranış kimsenin istemediğinin ya da kimsenin denemediğinin işaretidir. DO-331 bu analizi tasarım modelleri için tanımlar: kodun geliştirildiği veri tasarım modelidir ve istenmeyen işlev koda oradan taşınır.

Model kapsamının tek bir hazır tanımı yoktur; ölçütü proje belirler ve SVP'de yazar. Tipik ölçütler durum makinelerinde her durumun ve her geçişin, mantık ifadelerinde her kararın ve koşulun, sayısal girdilerde eşdeğerlik sınıflarının ve sınır değerlerin, anahtarlama ve doygunluk gibi öğelerde her çalışma kolunun uyarılmasıdır. Ölçüt araçtan araca değiştiği için "model kapsamı tam" cümlesi, hangi ölçüte göre tam olduğu söylenmeden bir şey ifade etmez.

Kapsanmayan her model ögesi tek tek çözümlenir. Olası nedenler ve karşılıkları:

- **Senaryo eksikliği:** Öge bir gereksinime izleniyor ama hiçbir senaryo onu tetiklememiş. Yeni ya da genişletilmiş senaryo yazılır.
- **Gereksinim eksikliği:** Öge gerekli bir davranışı ifade ediyor ama modelin geliştirildiği gereksinimlerde bu davranış yazmıyor. Önce gereksinim tamamlanır, sonra senaryosu eklenir.
- **Türetilmiş gereksinim:** Öge, tasarımın gerektirdiği ve üst seviyeye izlenemeyen bir davranıştır. Türetilmiş gereksinim olarak kaydedilir, sistem süreçlerine bildirilir ve kendi senaryosuyla doğrulanır.
- **Devre dışı bırakılmış işlev:** Öge tasarım gereği bu konfigürasyonda çalışmaz. Gerekçesi kaydedilir ve istem dışı etkinleşemeyeceği gösterilir.
- **İstenmeyen işlev:** Ögenin hiçbir dayanağı yoktur. Modelden kaldırılır.

Bu sınıflandırma, kod üzerindeki yapısal kapsam boşluklarının çözümlenmesiyle aynı mantığı izler ([9. Yazılım Doğrulama](#) ve [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#)); fark, sorunun kod yazılmadan önce yakalanmasıdır. Ancak model kapsam analizi kod üzerindeki yapısal kapsam analizinin yerini almaz. Üretilen ya da elle yazılan kod, modelde karşılığı olmayan yapılar içerebilir: kod üreticinin eklediği koruma dalları, kütüphane işlevleri, elle yazılmış arayüz kodu. Otomatik kod üretimi olsa bile satır kapsama (statement coverage), karar kapsama (decision coverage) ya da değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC) yükümlülüğü, yazılım seviyesinin gerektirdiği ölçüde, kod düzeyinde devam eder. Model kapsamı tam olan bir projede kod kapsamındaki boşluklar azalır, ama ölçüm yine kod üzerinde yapılır.

Model tabanlı akışta araçlar

Model tabanlı akış araçsız yürümez; her aracın çıktısına ne kadar güvenildiği de bir sertifikasyon sorusudur. Kural diğer araçlarla aynıdır: araç bir süreci kaldırıyor, azaltıyor ya da otomatikleştiriyorsa ve çıktısı ayrıca doğrulanmıyorsa araç kalifikasyonu (tool qualification) gerekir. Ölçütler ve araç kalifikasyon seviyeleri [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#) bölümünde anlatılmıştır; model tabanlı akışın tipik araçları şöyle değerlendirilir:

Araç	Yapabileceği hata	Kalifikasyon ne zaman gündeme gelir	Tipik sınıflandırma
Kod üretici	Modele aykırı ya da modelde olmayan kod üretir	Üretilen kodun modele karşı doğrulanması azaltılıyor ya da kaldırılıyorsa	Ölçüt 1
Simülatör	Modelin davranışını yanlış hesaplar, hatalı modeli doğru gösterir	Simülasyon sonucundan kredi alınıyor ve sonuç başka yolla doğrulanmıyorsa	Çoğunlukla Ölçüt 3
Model kapsam aracı	Uyarılmamış öğeyi kapsanmış gösterir	Kapsam raporu ayrıca doğrulanmadan kanıt olarak kullanılıyorsa	Ölçüt 3
Modelleme standardı denetleyicisi	Kural ihlalinin gözden kaçırır	Aracın denetlediği kurallar için gözden geçirme yapılmıyorsa	Ölçüt 3

Simülatörde sınıflandırma kullanım biçimine bağlıdır: simülasyon sonucu, aracın otomatikleştirdiği faaliyetin ötesinde başka bir geliştirme ya da doğrulama sürecini azaltmanın gerekçesi yapılıyorsa Ölçüt 2 tartışması açılır. Hangi ölçütün neden seçildiği PSAC'ta gerekçelendirilir. DO-330'un açıklayıcı soru-yanıtları simülatör için bir ayrım daha yapar: simülasyon yalnızca modelin doğrulanmasında kullanılıyor ve sonuçlar, senaryolarda önceden tanımlanmış beklenen sonuçlara karşı doğrulanıyorsa başvuru sahibi simülatörü kalifiye etmemeyi önerebilir. Sonuçların karşılaştırılmasını ya da prosedür üretimini otomatikleştiren işlevler ise, çıktıları ayrıca doğrulanmıyorsa kalifikasyon ister.

Kod üretici için iki strateji vardır. Kalifiye edilmemiş üreteçte üretilen kod elle yazılmış kod gibi ele alınır: tasarım modeline karşı gözden geçirilir, analiz edilir ve test edilir. Kalifiye üreteçte ise üretilen kaynak kodun tasarım modeline uygunluğunu gösteren gözden geçirme ve analizlerden kredi istenir. Bu kredinin üç sınırı vardır. Birincisi, üreteç Ölçüt 1 aracıdır; Seviye A yazılımda bu, en ağır kalifikasyon seviyesi demektir. İkincisi, kredi üreticinin kalifiye edildiği ayarlar ve model öğeleri alt kümesiyle sınırlıdır; modelleme standardı bu alt kümeyi zorunlu kılmalıdır. Üçüncüsü, kalifikasyon üreticinin çıktısında, yani kaynak kodda biter: derleyici ve bağlayıcı kapsam dışındadır. Çalıştırılabilir nesne kodunun hedefte test edilmesi, donanım/yazılım entegrasyonu, kod üzerindeki yapısal kapsam, en kötü durum yürütme süresi ve yığın kullanımı analizleri yerinde durur. DO-330'un açıklayıcı soru-yanıtları, kalifiye üreteçle düşük seviyeli gereksinimlere dayalı testlerin ve yapısal kapsam analizinin amaçlarına hangi ek analizlerle başka yoldan ulaşılabileceğini senaryolarla tartışır; bunlar rehber değil açıklamadır ve her biri otoriteyle mutabakat ister. Üreticinin girdisinin üstündeki gereksinimlere dayalı testler, entegrasyon testleri ve derleyicinin eklediği kodun doğrulanması her senaryoda projede kalır.

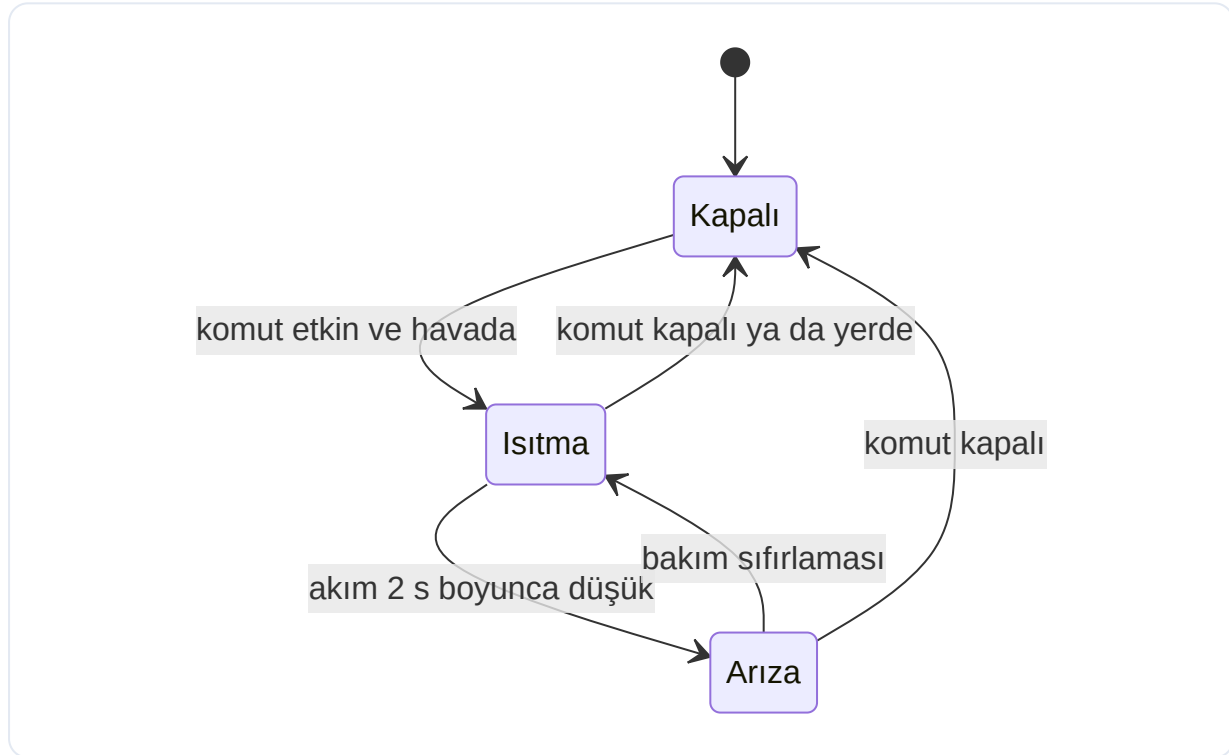
Araçlarla ilgili gözden kaçan bir ayrıntı daha vardır: simülatör modeli çoğu zaman yorumlayarak ya da simülasyona özgü bir kod üreterek çalıştırır; bu, uçuş için üretilen kodla aynı kod değildir. Simülasyonun gösterdiği, modelin davranışdır. Üretilen kodun aynı davranışı sergilediği ayrıca — kalifiye üreteçle, kodun doğrulanmasıyla ve hedef testleriyle — gösterilir.

Çalışılmış örnek: ısıtıcı mod mantığı

Aşağıdaki kurgusal örnek, en yaygın düzeni — metinsel yüksek seviyeli gereksinimler ve onlardan geliştirilen bir tasarım modeli — baştan sona izler. İşlev, bir ısıtıcı çıkışı denetleyen küçük bir mod mantığıdır. Yüksek seviyeli gereksinimler metin olarak yazılmıştır:

- **G-1:** Isıtma komutu etkinken ve uçak havadayken ısıtıcı çıkışı etkinleştirilir; bu iki koşuldan biri ortadan kalktığında çıkış kesilir.
- **G-2:** Çıkış etkinken ölçülen ısıtıcı akımı kesintisiz 2 saniye boyunca alt sınırın altında kalırsa çıkış kesilir ve arıza bildirilir.
- **G-3:** Arıza bildirimi, ısıtma komutu kapatılana dek korunur.

Tasarımcı bu gereksinimlerden bir durum makinesi geliştirir. Model tasarım modelidir: durumlar, geçişler ve 2 saniyelik sayacın nasıl tutulduğu tasarım kararlarıdır.



Doğrulama mühendisi modele değil gereksinimlere bakarak simülasyon senaryolarını yazar; beklenen sonuçlar da gereksinimlerden çıkar:

Senaryo	Gereksinim	Uyarım	Beklenen sonuç
S1	G-1	Yerde komut etkinleştirilir, ardından uçak havalanır	Çıkış yerdeyken kapalı kalır, havalanınca etkinleşir
S2	G-1	Isıtma sürerken komut kapatılır	Çıkış kesilir, arıza bildirilmez

Senaryo	Gereksinim	Uyarım	Beklenen sonuç
S3	G-2	Akım 1,9 saniye düşük kalır, sonra normale döner	Çıkış etkin kalır, arıza bildirilmez
S4	G-2, G-3	Akım kesintisiz 2 saniye düşük kalır; komut etkin tutulur	Çıkış kesilir ve arıza bildirilir; komut etkin kaldıkça bildirim korunur
S5	G-3	Arıza durumundayken komut kapatılır	Arıza bildirimi kalkar, çıkış kapalıdır

Beş senaryo da geçer. Ekip burada durursa model "doğrulanmış" görünür; oysa model kapsam analizi iki boşluk gösterir:

- *Isıtma* durumundan *Kapalı* durumuna geçişin "yerde" koşulu hiç uyarılmamıştır. Öğe G-1'e izlenir; sorun senaryo eksikliğidir. Isıtma sürerken inişi deneyen bir senaryo eklenir.
- *Arıza* durumundan *Isıtma* durumuna "bakım sıfırlaması" geçişi hiç uyarılmamıştır ve hiçbir gereksinime izlenmez. Tasarımcı bunu bakım kolaylığı için eklemiştir. Geçiş yalnızca izlenemeyen bir öğe değildir; G-3'ün tanımladığı davranışı da değiştirir. Bu yüzden tasarım içinde çözülemez: ya gereksinim sahiplerine götürülür ve emniyet etkisi değerlendirilerek gereksinim değiştirilir, ya da geçiş modelden kaldırılır.

İkinci boşluk, model kapsam analizinin varlık nedenini gösterir: bütün gereksinim tabanlı senaryolar geçtiği hâlde modelde kimsenin istemediği bir işlev durmaktadır ve hiçbir senaryo ona dokunmadığı için simülasyon bunu kendiliğinden ortaya çıkarmaz.

Model düzeltildikten sonra kod üretilir. Üreteç kalifiye değilse üretilen kod modele karşı gözden geçirilir. Ardından çalıştırılabilir nesne kodu hedefte test edilir: gereksinimlerden türetilen test durumları ile tasarım modeline dayalı test durumları hedefte koşulu, yapısal kapsam bu testler sırasında kod üzerinde ölçülür. Hedefin simülasyona eklediği bilgi somuttur. Simülasyonda 2 saniye ideal bir zaman adımıyla sayılır ve akım kusursuz bir sinyaldir; hedefte ise süre görevin gerçek çalışma periyoduyla sayılır ve akım ölçümü, çıkış etkinleştikten sonra bir süre oturmamış olabilir. G-2'nin bu koşullarda da sağlandığı ancak hedefte gösterilebilir.

Sık yapılan hatalar

- **Tek modele iki rol yüklemek.** Aynı model hem gereksinim hem tasarım olarak kullanılır; doğrulama modelin kendisiyle karşılaştırılmasına dönüşür.
- **Gereksinimi modelden geri yazmak.** Önce model kurulur, sonra modelin yaptığı metne dökülüp "gereksinim" diye adlandırılır. Böyle bir gereksinim modeldeki hatayı da içerir; ondan türetilen senaryolar hatayı doğrular.
- **Beklenen sonucu modelden üretmek.** Senaryo modelde koşulu, çıkan değer beklenen sonuç olarak kaydedilir. Bu, regresyon (regression) denetimi için işe yarar ama gereksinime uygunluk kanıtı değildir.

- **Kütüphane bloklarını doğrulama dışında bırakmak.** Hazır blokların davranışı ve ürettikleri kod, uçuş yazılımının parçasıdır.
- **Model ile kod arasındaki bağı koparmak.** Üretilen koda elle dokunmak ya da modelin hangi sürümünden, hangi üretici ayarlarıyla kod üretildiğini kaydetmemek, model üzerindeki bütün doğrulamayı o kod için geçersiz kılar.
- **Model kapsamını kod kapsamının, simülasyonu hedef testinin yerine sunmak.** İkisi de erken hata yakalar; ikisi de kod ve hedef üzerindeki kanıtın yerini tutmaz.

Bu bölümden akılda kalması gerekenler

- DO-331 tek başına okunmaz; DO-178C'nin hedeflerine, model kullanılan yerlerde ekleme ve değişiklik yapar. Model, izlenebilir ve doğrulanması gereken bir yaşam döngüsü verisidir.
- Modelin türü (belirtim mi tasarım mı) planlarda açıkça beyan edilmeli; bir model kendi kendisinin doğrulama referansı olamaz ve her zaman, geliştirildiği gereksinimlere karşı doğrulanır.
- Modelleme standartları, model öge kütüphanelerinin doğrulanması ve model ögesi düzeyinde izlenebilirlik planlama verisidir; izlenemeyen ama gerekli öge türetilmiş gereksinimdir.
- Simülasyon, disiplinli yürütülürse modelin doğrulanmasında kredi sağlar; çalıştırılabilir nesne kodunun test hedeflerine ise en çok kısmi kredi sağlar ve hedef ortam testini tümüyle ikame etmez.
- Model kapsam analizi tasarım modelindeki istenmeyen işlevi ve eksik gereksinimi erken yakalar; kod üzerindeki yapısal kapsam analizinin yerini almaz.
- Simülatör, kapsam aracı ve kod üretici için kalifikasyon sorusu, çıktılarına ne kadar güvenildiğine göre yanıtlanır; kalifiye kod üretici bile hedef testlerini ve yapısal kapsamı ortadan kaldırmaz.

15. DO-332 ve Nesne Yönelimli Teknoloji ve İlgili Teknikler

Nesne yönelimli teknoloji (object-oriented technology) soyutlama ve yeniden kullanım sağlar; ancak kalıtım (inheritance), çok biçimlilik (polymorphism) ve dinamik bellek yönetimi gibi özellikler doğrulamada yeni sorular doğurur. Bu bölüm, DO-178C'nin bu konuya ayrılmış teknoloji eki DO-332'nin getirdiklerini — yerel tip tutarlılığı, dinamik bellek yönetiminin doğrulanması ve zafiyet analizi — ve bunların tasarım ile kodlama kurallarına nasıl yansıdığını anlatır.

DO-332, "Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A" adıyla 2011'de DO-178C ile birlikte yayımlanmıştır; EUROCAE karşılığı ED-217'dir. Adından anlaşıldığı gibi yalnızca uçuş yazılımına değil, DO-278A kapsamındaki yer tabanlı haberleşme, seyrüsefer, gözetim ve hava trafik yönetimi (CNS/ATM) yazılımına da ek olarak uygulanır. Ek, nesne yönelimli tasarımı ne yasaklar ne de özendirir; bu tekniklerin hangi kanıtlarla emniyetli biçimde kullanılabileceğini tanımlar.

Havacılıkta nesne yönelimli teknolojinin kullanımı

Nesne yönelimli programlama, 1990'ların ortasından itibaren yer sistemlerinde ve kabin/eğlence gibi düşük kritiklikli alanlarda hızla yaygınlaştı. Emniyet-kritik uçuş yazılımına girişi ise çok daha temkinli oldu; çünkü dönemin geçerli rehberi olan DO-178B; kalıtım, çok biçimlilik ve dinamik bağlama (dynamic binding) gibi mekanizmaları hiç öngörmüyordu. Proje ekipleri C++ veya Ada 95 kullanmak istediğinde, sertifikasyon otoriteleri her seferinde projeye özel sorular soruyor, her başvuru sahibi (applicant) aynı tartışmaları yeniden yaşıyordu.

Tartışmaların odağında birkaç somut doğrulama sorunu vardı:

- **Dinamik bağlama ve yapısal kapsam analizi (structural coverage analysis) ilişkisi.** Sanal bir çağrı noktasında hangi alt sınıf metodunun çalışacağı ancak çalışma zamanında (run time) belli olur. Kaynak kod üzerinde satır kapsama (statement coverage) veya karar kapsama (decision coverage) yüzde yüz görünse bile, çağrı hedeflerinin tamamının denendiği bunu göstermez. "Bir sanal çağrı, olası her hedefi için ayrı ayrı mı kapsanmalı?" sorusu yıllarca net bir cevaba kavuşamadı.
- **Kaynak kod ile çalıştırılabilir nesne kodu arasındaki mesafe.** Sanal metot tabloları, örtük kurucu/yıkıcı çağrıları ve derleyicinin ürettiği yardımcı kod, izlenebilirlik (traceability) zincirini kaynak kod düzeyinde görünmez hâle getirebiliyordu.
- **Dinamik bellek yönetimi.** Nesnelerin çalışma zamanında yaratılıp yok edilmesi, bellek tükenmesi ve parçalanma (fragmentation) gibi zamanla ortaya çıkan hata kiplerini gündeme getirdi; en kötü durum davranışını kanıtlamak zorlaştı.

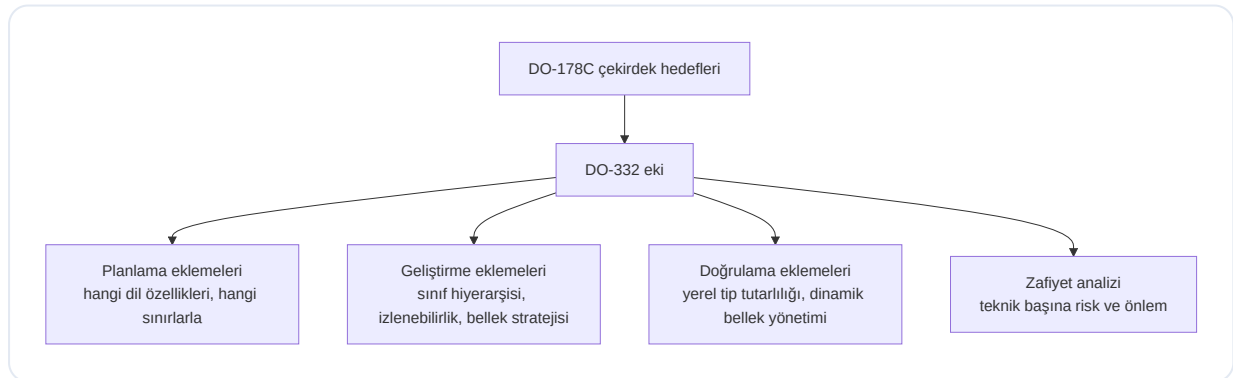
- **Test kapsamının anlamı.** Temel sınıfa karşı yazılmış testlerin alt sınıflar için de geçerli sayılıp sayılamayacağı, yani test mirasının (test inheritance) meşruiyeti belirsizdi.

Bu belirsizliği azaltmak için 2000'li yılların başında FAA ve NASA'nın desteğiyle sektör çalıştayları düzenlendi; bunların çıktısı olarak FAA, 2004'te dört ciltlik bir el kitabı yayımladı: Handbook for Object-Oriented Technology in Aviation (OOTiA). OOTiA bağlayıcı bir standart değildi, ama riskleri ve olası önlemleri sistemli biçimde ilk kez derledi. Aynı dönemde sertifikasyon otoritelerinin yazılım uzmanlarından oluşan ekip (Certification Authorities Software Team, CAST), nesne yönelimli teknolojinin doğurduğu sertifikasyon kaygılarını (CAST-4, 2000) ve C++ kullanımını (CAST-8, 2002) ele alan tutum belgeleri (position papers) yayımladı. Bu belgeler resmî politika değildi; otoritelerin ortak bakışını sektöre duyuruyordu. Bugün güncel rehber olarak sürdürülmezler ve DO-178B döneminden kalma tarihî başvuru kaynaklarıdır.

DO-178C hazırlanırken bu birikim doğrudan girdi oldu. Çekirdek dokümanı dil ve teknolojiden bağımsız tutma kararı alınınca, nesne yönelimli teknolojiye özgü hususlar ayrı bir teknoloji ekine, DO-332'ye taşındı. Böylece OOTiA'nın tavsiye niteliğindeki içeriği, hedefler (objective) ve faaliyetler düzeyinde tanımlı, denetlenebilir bir rehbera dönüştü. Tasarım yaklaşımı olarak nesne yönelimin yapısal tasarımıyla karşılaştırması [7. Yazılım Tasarımı](#) bölümündedir; bu bölüm, o seçimin sertifikasyon kanıtına etkisini ele alır.

DO-332'nin yapısı

DO-332 kendi başına okunacak bağımsız bir standart değildir; DO-178C'nin bir **teknoloji ekidir (supplement)**. Çekirdek dokümanın süreç ve hedef iskeletini aynen kullanır; nesne yönelimli teknolojinin etkilediği yerlerde hedefleri genişletir, faaliyet ekler veya mevcut faaliyetlerin nasıl yorumlanacağını netleştirir. Uygulamada bu şu anlama gelir: proje zaten DO-178C hedeflerini karşılamak zorundadır; DO-332 bunların üzerine, kullanılan dil özelliklerine bağlı ek yükümlülükler getirir. Ekin hedef düzeyindeki katkısı sınırlıdır: yalnızca iki yeni hedef tanımlar, geri kalan rehberliği mevcut DO-178C hedeflerine eklenen faaliyetler olarak verir.



Kapsam: "ilgili teknikler" dile bağlı değildir. Başlıktaki ikinci yarı en az birincisi kadar önemlidir. Şablonlar ve genel türler (templates, generics), aşırı yükleme (overloading), tip dönüşümü (type conversion), istisna işleme (exception handling), dinamik bellek yönetimi ve sanallaştırma

(virtualization) nesne yönelimli olmayan koddan da karşımıza çıkar. C ile yazılmış ama öbek (heap) belleği ya da gömülü bir yorumlayıcı kullanan proje, sınıf tanımlamaya da ekini ilgili rehberini dikkate alır. "DO-332 yalnızca C++ içindir" düşüncesi bu yüzden yanıltıcıdır.

Planlama eklemeleri. Bir projede C++ ya da Ada'nın nesne yönelimli özellikleri kullanılacaksa DO-332'nin uygulanacağı yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) beyan edilir. Hangi dil özelliklerinin kullanılacağı, hangilerinin kodlama standardıyla yasaklanacağı, yerel tip tutarlılığının hangi yöntemle gösterileceği ve belleğin hangi stratejiyle yönetileceği planlama kararıdır; doğrulama aşamasına bırakılırsa mimari çoktan bu kararları örtük olarak vermiş olur. Planların içeriği [5. Yazılım Planlama](#) bölümünde, otoritenin planlama denetiminde bu beyanı nasıl aradığı [SW SOI-1](#) kontrol listesinde yer alır.

Geliştirme eklemeleri. Sınıf hiyerarşisi yazılım mimarisinin parçasıdır ve tasarım verisinde tanımlanır. İzlenebilirlik de buna göre kurulur: nesne yönelimli tasarımda işlev metotlarda gerçekleştirildiği için gereksinimler, onları karşılayan metotlara ve niteliklere izlenir. Yeniden tanımlanan (override) her metot kendi gereksinimine izlenebilmelidir; "üst sınıftaki gereksinim zaten kapsıyor" varsayımı, davranışı değiştiren alt sınıfı gözden geçirir. Bellek yönetimi ve istisna işleme stratejisi de kod düzeyine bırakılmaz, mimari kararı olarak tasarımda yazılır.

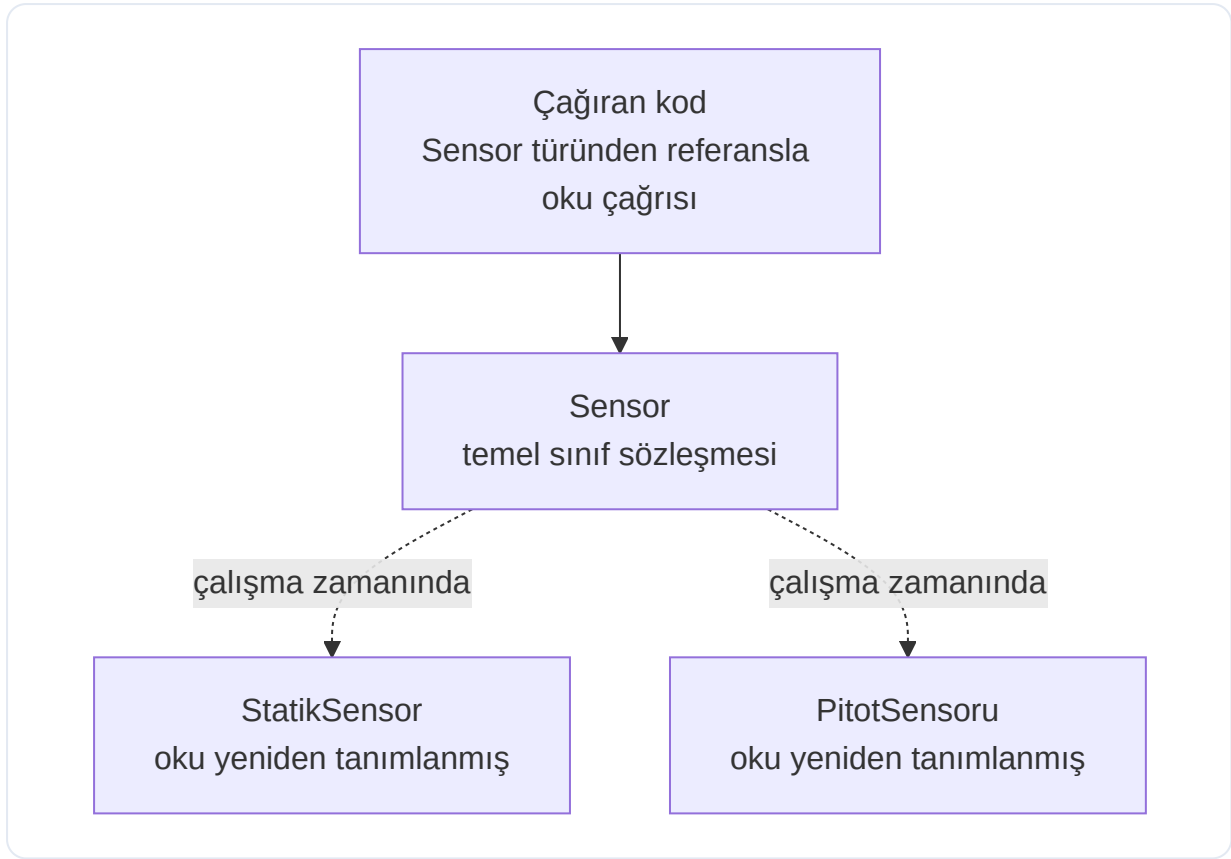
Doğrulama eklemeleri ve zafiyet analizi. İki yeni hedef doğrulama sürecindedir ve aşağıda ayrı ayrı ele alınır. Zafiyet analizi ise ek faaliyetlerin gerekçesini açıklayan destekleyici kısımdır.

Doğrulama eklemeleri: iki yeni hedef

Yerel tip tutarlılığı

Ekin en bilinen katkısı **yerel tip tutarlılığının doğrulanmasıdır (local type consistency verification)**. Fikir, yerine geçme ilkesine (Liskov substitution principle) dayanır: bir alt sınıf nesnesi temel sınıfın beklendiği her yerde kullanılabilirliğine göre, alt sınıfın davranışının temel sınıf sözleşmesini bozmadığı **gösterilmelidir** — varsayılmaz. Sözleşme açısından kural üç maddedir: yeniden tanımlanan metot önkoşulu (precondition) sıkılaştıramaz, artkoşulu (postcondition) gevşetemez ve temel sınıfın değişmezlerini (invariant) korur.

Küçük bir örnek sorunu görünür kılar. Hava verisi işleyen bir bileşen, sensörlere ortak bir temel sınıf üzerinden erişsin:



Sınıf	oku için önkoşul	oku için artkoşul	Yerine geçme ilkesi
Sensor (temel sınıf)	Başlatma tamamlanmış olmalı	Değer tanımlı aralıktadır ya da geçersiz olarak işaretlenmiştir	—
StatikSensor	Aynı	Aynı; ek olarak değer süzgeçten geçirilmiştir	Korunur: artkoşul yalnızca güçlenmiştir
PitotSensoru	Başlatma tamamlanmış ve ısınma süresi dolmuş olmalı	Aynı	Bozulur: önkoşul sıkılaştırılmıştır

Çağırıcı kod yalnızca **Sensor** sözleşmesini bilir ve başlatmadan hemen sonraki çevrimde **oku** çağırır. Hedef **PitotSensoru** ise önkoşul ihlal edilmiştir ve dönen değer ne olacağını hiçbir gereksinim söylemez. Sorunun sinsi yanı, tek tek bakıldığında her şeyin doğru görünmesidir: **PitotSensoru** kendi gereksinimlerine göre yazılmış testlerden geçer, çünkü o testler ısınma süresini bekler; çağırıcı kod da temel sınıfı test edildiğinde geçer. Kusur yalnızca ikisinin birleşiminde ortaya çıkar. Çözüm sözleşmeyi onarmaktır: alt sınıf ısınma süresince değeri geçersiz işaretleyerek temel sınıfın artkoşuluna uyar ya da ısınma koşulu temel sınıfın sözleşmesine taşınır ve bütün çağırıcılar buna göre gözden geçirilir.

"Yerel" sözcüğü kapsamı belirler. Hiyerarşideki bütün sınıfların her bağlamda birbirinin yerine geçebildiğini göstermek gerekmez; gösterim, kodda gerçekten bulunan dinamik çağrı noktaları ve o noktalara fiilen ulaşabilen tipler için yapılır. Ek, bunun için üç yol tanır ve seçimi başvuru sahibine bırakır:

- **Biçimsel ispat.** Her yeniden tanımlanan metodun temel sınıf sözleşmesini sağladığı [16. DO-333 ve Biçimsel Yöntemler](#) bölümünde anlatılan tekniklerle ispatlanır. Sözleşmelerin biçimsel dille yazılmış olmasını gerektirir.
- **Üst sınıf testlerinin alt sınıf örnekleriyle koşulması (test mirası).** Temel sınıfın gereksinim tabanlı testleri (requirements-based testing), onun yerine geçebilen her alt sınıfın nesnesiyle yeniden koşulur ve hepsinin geçmesi beklenir. Derin hiyerarşilerde en ekonomik yol çoğunlukla budur. Yukarıdaki örnekte temel sınıfın başlatmadan hemen sonra `oku` çağırın testi, `PitotSensoru` nesnesiyle koşulduğunda kusuru ortaya çıkarır.
- **Kötümser test (pessimistic testing).** Yerine geçebilirlik iddia edilmez; her dinamik çağrı noktası, oraya ulaşabilecek her hedef için ayrı ayrı gereksinim tabanlı testle sınanır. Az sayıda alt sınıfı olan sığ hiyerarşilerde pratiktir, derinlik arttıkça birleşim sayısı hızla büyür.

Bu hedef yapısal kapsam analizini tamamlar, onun bir türü değildir: kapsam analizi hangi kodun çalıştırıldığını ölçer, yerel tip tutarlılığı ise dinamik çağrının olası hedef kümesindeki her metodun çağırının güvendiği sözleşmeyi karşıladığını gösterir. Hedef, Seviye A, B ve C yazılım için aranır.

Dinamik bellek yönetiminin doğrulanması

DO-332'nin eklediği ikinci hedef, dinamik bellek yönetiminin gürbüz (robust) olduğunun doğrulanmasıdır. Kapsamı ilk bakışta sanıldığından geniştir: yalnızca açık ayırma ve serbest bırakma çağrılarını değil, çöp toplamayı (garbage collection) ve kap (container) sınıfları gibi üst düzey yapıların örtük bellek kullanımını da içerir. Gösterilmesi beklenen özellikler yedi başlıkta toplanır:

Gösterilecek özellik	Kapatılan risk	Tipik kanıt
Ayırıcı her seferinde geçerli ve başka kullanımda olmayan bir alan döndürür	Aynı alanın iki nesneye verilmesi (belirsiz referans)	Ayırıcının gereksinim tabanlı testi ve gözden geçirmesi
Yeterli boş alan varken ayırma, parçalanma yüzünden başarısız olmaz	Parçalanmadan doğan açıklık	Sabit boyutlu bloklar ya da parçalanma analizi
Erişilemez hâle gelen bellek, ihtiyaç doğmadan geri kazanılır	Geri kazanımın gecikmesinden doğan açıklık	Serbest bırakma disiplini ya da çöp toplayıcının hız analizi
Uygulamanın ihtiyaç duyduğu toplam bellek her an mevcuttur	Öbek belleğin tükenmesi	En kötü durum bellek ihtiyacı analizi
Nesne ancak artık kullanılmadığında serbest bırakılır	Erken serbest bırakma, sarkan referans	Sahiplik kuralları, statik analiz
Bellek yöneticisi nesneleri taşıyorsa referanslar tutarlı kalır	Kayıp güncelleme, bayat referans	Taşıma işleminin bölünmezliğinin gösterilmesi
Ayırma ve serbest bırakma sınırlı sürede tamamlanır	Belirlenimci (deterministic) olmayan yürütme süresi	En kötü durum yürütme süresi (worst-case execution time) analizi

Bu özelliklerin kimin tarafından sağlanacağı seçilen tekniğe bağlıdır. Ek, uygulama kodu ile bellek yönetimi altyapısını (çalışma zamanı kütüphanesi, işletim sisteminin ayırıcısı, çöp toplayıcı) ayırır ve her teknik için sorumluluğun hangisine düştüğünü tartışır. Çöp toplayıcı kullanıldığında yükün çoğu altyapıya geçer; ama o altyapı da uçuş yazılımıdır ve aynı seviyede doğrulanır. Uygulamanın kendi yazdığı nesne havuzunda ise yükün çoğu uygulama kodundadır. Toplam bellek ihtiyacının karşılandığını göstermek, teknik ne olursa olsun uygulamanın işidir.

Uygulamada üç strateji görülür ve kanıt yükü bu sırayla artar:

1. **Yalnızca başlatmada ayırma, hiç serbest bırakmama.** En yaygın ve en kolay savunulan yoldur. Parçalanma, erken serbest bırakma ve geri kazanım sorunları tanım gereği ortadan kalkar; geriye başlatma sırasında yeterli belleğin bulunduğunu ve normal çalışmaya geçildikten sonra ayırma yapılmadığını göstermek kalır.
2. **Sabit boyutlu havuzlar.** Çalışma sırasında nesne alınıp geri verilir; bloklar eş boyutlu olduğu için parçalanma oluşmaz ve süre sınırlıdır. Havuzun en kötü durumda yetmesi ve geri verilen bloğu gösteren referans kalmaması ayrıca gösterilir.
3. **Çalışma sırasında genel amaçlı öbek kullanımı.** Yedi özelliğin tamamı açık kalır; özellikle parçalanmanın sınırlı olduğunu göstermek güçtür. Emniyet-kritik projelerin çoğu bu yüzden bu seçeneği kodlama standardıyla yasaklar.

Belirlenimci ayırma mekanizmalarının işletim sistemi tarafı [20. Gerçek Zamanlı İşletim Sistemleri](#) bölümünde anlatılır.

Zafiyet analizi

DO-332, teknik başına "bu mekanizma neyi bozabilir, hangi önlem bunu kapatır" sorusunu işleyen sistematik bir **zafiyet analizi (vulnerability analysis)** içerir. Bu kısım yeni hedef koymaz; ekin istediği faaliyetlerin neden istendiğini açıklar ve projenin kendi kullandığı özellikler için benzer bir değerlendirme yapmasına örnek olur. Sık başvurulanan başlıklar:

Dil özelliği / teknik	Tipik zafiyet	Beklenen önlem
Kalıtım, çok biçimlilik	Sözleşmeyi bozan alt sınıf, belirsiz çağrı hedefi	Yerel tip tutarlılığı doğrulaması, kalıtım derinliği sınırı
Aşırı yükleme	Örtük tip dönüşümüyle beklenmeyen fonksiyonun seçilmesi	Kodlama standardı kısıtları, statik analiz, kaynak kod gözden geçirmesi
Tip dönüşümü	Daraltan dönüşümde veri kaybı ya da taşma; hiyerarşide aşağı doğru dönüşümde yanlış tip varsayımı	Örtük dönüşümlerin kısıtlanması, açık ve denetimli dönüşüm
Şablonlar, genel türler	Aynı şablonun farklı somutlaştırmalarının farklı davranması, kapsam boşluğu	Her somutlaştırmının ayrı test edilmesi ve kapsanması

Dil özelliği / teknik	Tipik zafiyet	Beklenen önlem
İstisna işleme	Kontrol akışının görünmez dallanması, yakalanmayan istisna, yarım güncellenmiş veri	Mimari düzeyinde istisna stratejisi, kısıtlı kullanım, istisna yollarının testi
Dinamik bellek yönetimi	Tükenme, parçalanma, sarkan referans, belirlenimci olmayan süre	Yukarıdaki yedi özelliğin gösterilmesi, ayırma stratejisinin kısıtlanması
Çöp toplama	Kesilemeyen duraklamalar, zamanlama belirsizliği	En kötü durum duraklama analizi ya da tamamen yasaklama
Sanallaştırma (sanal makine, yorumlayıcı)	Yorumlanan kodun "veri" sayılıp doğrulamanın dışında kalması	Sanal makine uçuş yazılımı olarak DO-178C'ye göre doğrulanır; yorumlanan kod çalıştırılabilir kod gibi ele alınır

Son satır sık karıştırılan bir noktayı düzeltir. Uçakta çalışan sanal makine ya da yorumlayıcı bir araç değildir; çalıştırılabilir nesne kodunun parçasıdır ve yazılım seviyesinin gerektirdiği bütün hedeflerle geliştirilip doğrulanır. Araç kalifikasyonu, uçuş yazılımının parçası olmayan geliştirme ve doğrulama araçları içindir (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)). Sanal makinenin yorumladığı ara kod (örneğin bayt kodu) da bir yapılandırma dosyası gibi geçirilemez: gereksinimlere izlenir, gözden geçirilir, test edilir ve kapsamı ölçülür.

Dile bağlı olmayan sorun: C'de fonksiyon işaretçisiyle dağıtım

Fonksiyon işaretçileriyle tablo tabanlı dağıtım yapan bir C tasarımı resmen DO-332'nin kapsamında sayılmasa da yerel tip tutarlılığının sorduğu soruyu aynen doğurur. Aşağıdaki parça, dilde sınıf olmadan da aynı doğrulama sorusunun doğduğunu gösterir:

```

#include <stdbool.h>
#include <stddef.h>

/* baro_* ve radar_* işlevlerinin bildirimleri kısalık için gösterilmemiştir. */

typedef struct {
    bool (*saglikli_mi)(void);
    bool (*oku)(float *irtifa_m); /* true: *irtifa_m geçerli */
} sensor_arayuzu_t;

#define SENSOR_SAYISI ((size_t)2)

/* Olası hedef kümesi: sabit tablo, derleme zamanında sayılabilir. */
static const sensor_arayuzu_t SENSOR_TABLOSU[SENSOR_SAYISI] = {
    { baro_saglikli_mi, baro_oku },
    { radar_saglikli_mi, radar_oku }
};

bool irtifa_oku(size_t sensor_no, float *irtifa_m)
{
    bool gecerli = false;

    if ((irtifa_m != NULL) && (sensor_no < SENSOR_SAYISI)) {
        const sensor_arayuzu_t *s = &SENSOR_TABLOSU[sensor_no];

        if (s->saglikli_mi()) {
            gecerli = s->oku(irtifa_m); /* dinamik çağrı */
        }
    }
    return gecerli; /* false ise *irtifa_m kullanılmaz */
}

```

`s->oku()` çağrısının hangi fonksiyona gideceği çalışma zamanında belirlenir; dolayısıyla "olası bütün hedefler sözleşmeye uyuyor mu ve test edildi mi?" sorusu, tıpkı sanal metotlarda olduğu gibi burada da cevaplanmalıdır. Örnekteki iki tasarım kararı cevabı kolaylaştırır. Tablo sabit (`const`) olduğu için olası hedef kümesi kaynak koddan okunur: iki sensör, dört fonksiyon; çalışma zamanında yeni hedef eklenemez ve işaretçiler boş kalmaz. Arayüzün sözleşmesi de imzada görünür: `oku` geçerlilik bilgisini döndürür, böylece "sağsız sensörde ne döner" sorusu her gerçekleştirim için aynı biçimde cevaplanır. `baro_oku` ile `radar_oku` bu sözleşmeye karşı ayrı ayrı test edilir; `irtifa_oku` ise her iki hedefle sınanır. DO-332'yi resmen uygulamayan projelerde bile bu bakış açısı iyi bir mühendislik alışkanlığıdır.

Uygulamada dil alt kümesi ve kaynak-nesne kodu ilişkisi

DO-332 bir dil alt kümesi tanımlamaz; hangi özelliğin kullanılacağına proje karar verir ve bu karar kodlama standardıyla bağlayıcı hâle gelir. Standardın yasakladığı özellik için kanıt üretmek gerekmez, izin verdiği her özellik için ise ekteki ilgili faaliyet planlanır. Bu yüzden projeler sıfırdan kural yazmak yerine yerleşik bir standardı temel alıp üzerine kendi kısıtlarını ekler. C++ için güncel

başvuru MISRA C++:2023'tür; C++17'yi hedefler ve MISRA C++:2008 ile AUTOSAR C++14 yönergelerini tek belgede birleştirerek onların yerini almıştır. Askerî bir uçak programı için 2005'te hazırlanan JSF AV C++ kodlama standardı da hâlâ sık anılan bir kaynaktır. Ada tarafında dilin kendi kısıtlama mekanizmaları ve SPARK alt kümesi aynı işi görür; Ada 2012 ile gelen sınıf geneli önkoşul ve artkoşullar, yerine geçme ilkesini dilin içinde ifade etmeyi sağlar. Kodlama standardının projede nasıl uygulandığı [8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#) bölümündedir.

Başta anılan kaynak kod–nesne kodu mesafesi için DO-178C'nin cevabı değişmez: Seviye A'da derleyicinin ürettiği ve kaynak koda doğrudan izlenemeyen nesne kodu belirlenir ve doğruluğu ek doğrulamayla gösterilir. Nesne yönelimli özellikler bu ek doğrulamanın konusunu büyütür: sanal metot tabloları ve dağıtım kodu, örtük kurucu ve yıkıcı çağrılar, geçici nesneler, istisna çözme tabloları, şablon somutlaştırmaları. Dil alt kümesini daraltmak ve derleyici seçeneklerini (örneğin istisna desteğini kapatmak) planlama aşamasında sabitlemek, bu iş yükünü belirleyen asıl karardır.

Mevcut doğrulama faaliyetlerinin yorumu da genişler. Yapısal kapsam analizinde her şablon somutlaştırması ayrı kod sayılır ve ayrı kapsanır. Kalıtım ve dinamik çağrı, bileşenler arasında parametre, çağrı ve ortak veriye ek olarak yeni bir bağlaşım biçimi yaratır; veri ve kontrol bağlaşımı analizi bunu da kapsamalıdır. En kötü durum yürütme süresi ve yığın kullanımı analizlerinde ise dinamik çağrının olası hedeflerinin en kötüsü esas alınır; pratik yol, süre ve yığın sınırını temel sınıf sözleşmesine yazıp her alt sınıfın buna uyduğunu göstermektir. Bu analizlerin genel anlatımı [9. Yazılım Doğrulama](#) bölümündedir.

Pratik tasarım ve kodlama kuralları

Aşağıdaki kurallar DO-332'nin metninden değil, onun istediği kanıtı üretilebilir kılma kaygısından doğar. Her biri bir doğrulama maliyetine karşılık gelir.

- **Kalıtım derinliğini ve çoklu kalıtımı sınırlayın.** Her ek düzey, yerel tip tutarlılığı kanıtına yeni bir sınıf ve yeni metot birleşimleri ekler. Çoklu kalıtım çağrı hedefinin belirlenmesini ve nesne yerleşimini karmaşıklaştırdığı için kodlama standartlarında çoğunlukla yasaklanır ya da saf arayüz sınıflarıyla sınırlanır.
- **Temel sınıf sözleşmesini düşük seviyeli gereksinim olarak yazın.** Önkoşul, artkoşul, değişmezler ve kaynak sınırları yazılı değilse "alt sınıf sözleşmeyi bozdu mu?" sorusu gözden geçirmede de testte de cevaplanamaz.
- **Yeniden tanımlamayı açık ve izlenebilir kılın.** Yeniden tanımlanan her metot kendi gereksinimine izlenir; imza uyumsuzluğu yüzünden kazara yeni bir metot tanımlamak derleyici denetimiyle (C++'ta `override` belirteci) engellenir. Gizli davranış değişiminin en sık kaynağı budur.
- **Dinamik çağrının hedef kümesini kapalı tutun.** Hedefler derleme zamanında sayılabilmeli, bağlama başlatmadan sonra değişmemelidir. Çalışma zamanında sınıf yükleme ya da işaretçi tablosunu güncelleme, doğrulanmış hedef kümesinin dışına çıkmanın yoludur.

- **Belleği başlatmada ayırın, örtük ayırmaları arayın.** Standart kütüphane kapları, dizgi sınıfları ve bazı dil yapıları öbeği sessizce kullanır; kural yalnızca açık ayırma çağrılarını yasaklıyorsa bunlar gözden kaçır.
- **Yan etkileri görünür kılın.** Kurucu ve yıkıcılarda iş mantığı, operatör aşırı yüklemesi ve örtük tip dönüşümleri kaynak kodda görünmeyen çağrılar üretir; gözden geçiren kişinin okuduğu ile çalışan kod arasındaki fark büyür.
- **Soyutlamayı gereksinim sürüklesin.** Tek alt sınıfı olan soyut katmanlar ve "ileride lazım olur" diye eklenen sanal metotlar, gereksinime izlenemeyen ya da hiç çalıştırılmayan kod üretir ve kapsam analizinde açıklanması gereken boşluklar olarak geri döner.
- **Alt sınıf testlerini temel sınıf testlerinin üzerine kurun.** Alt sınıfı yalnızca kendi gereksinimlerine karşı test etmek, yukarıdaki sensör örneğindeki türden kusurları görmez; test düzeneği temel sınıf testlerini her alt sınıfla koşacak biçimde kurulur.

Bu bölümden akılda kalması gerekenler

- DO-332 bağımsız bir standart değil, DO-178C ve DO-278A'nın teknoloji ekidir; nesne yönelimli teknikleri yasaklamaz, kullanılan özelliğe bağlı kanıt yükümlülüğü tanımlar.
- Ek yalnızca iki yeni hedef getirir: yerel tip tutarlılığının ve dinamik bellek yönetiminin gürbüzlüğünün doğrulanması. Geri kalan rehberlik mevcut hedeflere eklenen faaliyetlerdir.
- Yerel tip tutarlılığında alt sınıfın temel sınıf sözleşmesine uyduğu varsayılmaz; biçimsel ispatla, üst sınıf testlerinin alt sınıflarla koşulmasıyla ya da her çağrı noktasında her hedefin test edilmesiyle gösterilir.
- Dinamik bellek için gösterilecek özellikler bellidir; yalnızca başlatmada ayırma ve sabit boyutlu havuzlar bu kanıtı üretilebilir kılan stratejilerdir.
- "İlgili teknikler" dile bağlı değildir: şablon, tip dönüşümü, istisna, dinamik bellek ve sanallaştırma C projelerini de ilgilendirir. Sanal makine araç değil uçuş yazılımıdır; yorumladığı kod veri değil çalıştırılabilir koddur.
- Dinamik çağrı sorunu da dile özgü değildir: C'de fonksiyon işaretçisiyle dağıtım yapan tasarım aynı soruyu cevaplar; sabit bir tablo hedef kümesini sayılabilir kılar.
- Hangi özelliğin kullanılacağı planlama kararıdır ve kodlama standardıyla bağlanır; bu karar hem yerel tip tutarlılığı kanıtının hem de Seviye A'da kaynak koda doğrudan izlenemeyen nesne kodu için yapılacak ek doğrulamanın büyüklüğünü belirler.

16. DO-333 ve Biçimsel Yöntemler

Biçimsel yöntemler (formal methods), yazılımın davranışını matematiksel bir gösterimle ifade edip istenen özelliklerin sağlandığını örnek denemelerle değil ispatla gösterir; DO-333 bu tür bir analizin hangi koşullarda gözden geçirme, analiz ve bazı testlerin yerine doğrulama kanıtı sayılabileceğini tanımlar. Bu bölüm DO-333'ün getirdiklerini; model kontrolü (model checking), teorem ispatı (theorem proving) ve soyut yorumlama (abstract interpretation) yaklaşımlarını; biçimsel analiz testin yerini aldığına değişenleri; pratikteki zorlukları ve iki küçük örneği ele alır.

DO-333'ün tam adı "Formal Methods Supplement to DO-178C and DO-278A"dır; 2011'de DO-178C ile birlikte yayımlanmıştır ve EUROCAE karşılığı ED-216'dır. Bir ek (supplement) olduğu için tek başına okunmaz: DO-178C'nin bölüm yapısını izler, biçimsel yöntem kullanılan yerlerde ana belgenin metnine ekleme ve değişiklik yapar, hedef (objective) tablolarına da biçimsel analize özgü hedefler ekler. Yazılımın biçimsel yöntemle doğrulanmayan kısımları için DO-178C olduğu gibi geçerlidir (belge ailesinin bütünü için bkz. [4. DO-178C ve Destekleyici Dokümanlara Genel Bakış](#)). Ekin ana fikri şudur: biçimsel analiz testi tümüyle ortadan kaldırmaz; ama koşulları sağlandığında bazı gözden geçirme, analiz ve testlerin yerine geçebilecek güçte bir kanıttır. Amaç daha az doğrulama yapmak değil, aynı hedefe daha kesin bir kanıtla ulaşmaktır.

Neden biçimsel yöntem?

Test, doğası gereği örneklemedir: mümkün girdilerin ve durumların küçük bir alt kümesi seçilir, yazılım bu noktalarda çalıştırılır ve sonuç beklenenle karşılaştırılır. 32 bitlik iki girdisi olan durumsuz bir fonksiyonun bile 2^{64} farklı girdi çifti vardır; iç durum, zamanlama ve eşzamanlılık (concurrency) eklendiğinde uzay pratikte sayılamaz hâle gelir. Gereksinim tabanlı test (requirements-based testing) ve yapısal kapsam analizi (structural coverage analysis) bu örnekleme disiplini eder, ama seçilmeyen noktalar hakkında bir şey söylemez.

Biçimsel analiz soruyu tersinden sorar: "şu girdide ne oluyor?" yerine "bu özelliğin bozulduğu herhangi bir girdi ya da durum var mı?". Cevap, modelin ve yazılı varsayımların kapsadığı uzayın tamamı için geçerlidir. Bu fark en çok şu tür mantıkta karşılığını verir: çok sayıda durumu ve geçişi olan mod mantığı; "şu koşul yokken bu komut asla verilmez" türünden emniyet kilitlemeleri (interlock); artıklık yönetimi ve oylama mantığı; taşma ya da sıfıra bölme gibi yalnızca dar bir girdi aralığında ortaya çıkan aritmetik hatalar. Ortak yanları, hatanın nadir bir olay sıralamasında ya da bir sınırdaki saklanması ve testle bulunmasının, birinin o senaryoyu akıl etmesine bağlı olmasıdır.

Bu gücün iki sınırı vardır: sonuç, model ve varsayımlar gerçeği yansıttığı ölçüde geçerlidir ve gerçek donanım üzerindeki davranışı tek başına göstermez. DO-333'ün büyük kısmı bu iki sınırın nasıl yönetileceğiyle ilgilidir.

DO-333 ne getirir?

Biçimsel model ve biçimsel analiz

DO-333'e göre bir biçimsel yöntem iki parçadan oluşur; DO-178C'nin sözlüğü de biçimsel yöntemi aynı ikiliyle tanımlar. **Biçimsel model (formal model)**, yazılımın bir yönünün — gereksinimlerin, tasarımın ya da kodun — sözdizimi ve anlamı matematiksel olarak tanımlı, tek biçimde yorumlanabilen bir gösterimle (formal notation) ifadesidir. **Biçimsel analiz (formal analysis)**, bu modelin belirli bir özelliği sağladığını matematiksel akıl yürütmeye gösterir. Anlamı araçtan araca ya da okurdan okura değişen bir kutu-ok diyagramı ya da serbest metin bu anlamda model değildir. Kaynak kod ise, dilin anlamı kesin tanımlandığı ölçüde, biçimsel model olarak ele alınabilir; soyut yorumlama araçlarının ayrı bir model kurmadan doğrudan kod üzerinde çalışabilmesi bundandır.

Kredinin üç ön koşulu

Bir analizin doğrulama kanıtı sayılabilmesi için ek, yöntemin türünden bağımsız üç şey ister:

- **Kesin gösterim.** Kullanılan her biçimsel gösterimin sözdizimi ve anlamı matematiksel olarak tanımlı ve belirsizlikten uzak olmalıdır.
- **Sağlamlık (soundness).** Sağlam bir analiz yöntemi, özelliğin sağlanmadığı bir durumda asla "sağlanıyor" demez. Ters serbesttir: gerçekte sorun yokken uyarı verebilir. Her yöntemin sağlamlığı gerekçelendirilir. Aradığı hatayı kaçırabilen sezgisel statik analiz araçları yararlı olabilir, ama bu anlamda biçimsel analiz değildir. Buradaki sağlamlık yöneme ilişkindir; yöntemi uygulayan aracın hatasız çalışıp çalışmadığı ayrı bir sorudur ve araç kalifikasyonu (tool qualification) ele alınır.
- **Yazılı ve gerekçeli varsayımlar.** Girdi aralıkları, hedef işlemcideki tamsayı genişlikleri ve kayan nokta davranışı, kesme ve çizelgeleme düzeni gibi analizin dayandığı her kabul kayda geçirilir ve gerekçelendirilir. Gerekçesi olmayan varsayım, ispatın sessizce dışarıda bıraktığı bir durum demektir.

Ek hedefler

Biçimsel analizin kullanıldığı doğrulama hedefi tablolarına, yöntemin kendisine bakan hedefler eklenir. Özü dört soruya iner:

1. Biçimsel analiz durumları (formal analysis case) ve prosedürleri doğru mu? Bunlar kabaca test durumu ile test prosedürünün karşılığıdır: neyin hangi varsayımlarla gösterileceği ve analizin hangi araç ayarlarıyla nasıl yürütüleceği.
2. Analiz sonuçları doğru mu ve beklenmeyen her sonuç açıklanmış mı?
3. Gereksinimlerin biçimsel gösterime aktarımı — biçimselleştirme — aslına sadık mı?
4. Seçilen yöntem doğru tanımlanmış, gerekçelendirilmiş ve bu iş için uygun mu?

Bu sorular uygulamada gözden geçirmeyle yanıtlanır. İspat ne kadar otomatik olursa olsun, "doğru şeyi mi ispatladık?" sorusunun cevabı insandan beklenir.

Hangi hedeflerde kullanılabilir?

İlke olarak doğrulama hedeflerinin çoğu biçimsel analize adaydır. Aşağıdaki tablo standardın hedef tablolarının dökümü değil, kredinin nerelerde arandığının özetidir; hedef grupları [9. Yazılım Doğrulama](#) bölümünde anlatılmıştır.

Doğrulama hedefi grubu	Biçimsel analizle karşılanabilir mi?	Koşul
Yüksek seviyeli gereksinimlerin (high-level requirements) gözden geçirme ve analizi (üst gereksinimlere uyum, doğruluk ve tutarlılık, algoritma doğruluğu)	Evet	Gereksinimler biçimsel gösterimle yazılmış ya da biçimselleştirilmiş olmalı; aktarımın doğruluğu gözden geçirilir
Tasarımın gözden geçirme ve analizi (düşük seviyeli gereksinimler (low-level requirements), mimari)	Evet	Tasarım biçimsel bir modelle ifade edilmiş olmalı
Kaynak kodun gözden geçirme ve analizi (gereksinimlere uyum, doğruluk ve tutarlılık)	Evet	Dilin anlamı ile derleyici ve hedef işlemciye ilişkin varsayımlar yazılı olmalı
Çalıştırılabilir nesne kodunun (executable object code) gereksinimlere uyumu ve gürbüzlüğü (robustness)	Evet, kısmen ya da tümüyle	Analiz nesne kodu üzerinde yapılır ya da kaynak kodda ispatlanan özelliklerin nesne kodunda korunduğu gösterilir
Çalıştırılabilir nesne kodunun hedef bilgisayarla uyumluluğu	Hayır; test kalır	Donanım/yazılım entegrasyon testleri (hardware/software integration testing) hedefte koşulur
Doğrulamanın yeterliliği (kapsam analizleri)	Biçimsel analizle doğrulanan kısımda yapısal kapsamın yerini başka ölçütler alır	Bkz. aşağıda "Biçimsel analiz testin yerini aldığı"

Standartlara uygunluk gibi biçimsel özellik olarak ifade edilmesi güç hedeflerde gözden geçirme çoğunlukla yerinde kalır. Hangi hedefte tam, hangisinde kısmi kredi isteneceği proje bazında belirlenir.

Planlarda beyan

Biçimsel analizden kredi almak sonradan eklenebilecek bir ayrıntı değildir. Yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) ekin hangi bileşenlere uygulanacağı ve hangi hedeflerde ne ölçüde kredi isteneceği; yazılım doğrulama planında (Software Verification

Plan, SVP) yöntem, gösterim, araçlar, varsayımların nasıl yönetileceği ve testle biçimsel analiz arasındaki iş bölümü yer alır. Bu beyan sertifikasyon otoritesiyle (certification authority) varılacak mutabakatın temelidir; planların içeriği [5. Yazılım Planlama](#) bölümünde işlenmiştir.

Biçimsel yöntem kategorileri

DO-333 biçimsel analizi üç ailede toplar: tümdengelimli yöntemler (deductive methods; yaygın adıyla teorem ispatı), model kontrolü ve soyut yorumlama. Üçü de matematiksel temellidir; ancak sordukları soru, verdikleri cevabın biçimi ve gerektirdikleri emek birbirinden oldukça farklıdır.

Model kontrolü

Model kontrolü, sistemin davranışını sonlu bir durum uzayı olarak modeller ve "istenmeyen durum hiçbir çalışma senaryosunda oluşmaz" gibi bir özelliği bu uzayın **tamamını** otomatik tarayarak denetler. En değerli çıktısı, özellik ihlal edildiğinde üretilen **karşı örnektir (counterexample)**: araç, hataya götüren somut olay dizisini adım adım gösterir. Bu, hata ayıklamada test kayıtlarından çok daha yönlendirici bir bilgidir. Tasarım, anlamı kesin tanımlı bir modelleme diliyle yazılmışsa (bkz. [14. DO-331 ve Model Tabanlı Geliştirme ve Doğrulama](#)) model kontrolü ayrı bir model kurmadan doğrudan tasarım modeli üzerinde çalıştırılabilir.

- **Güçlü yönleri:** yüksek otomasyon, durum makinesi (state machine) ve eşzamanlılık mantığında derinlemesine tarama, anlaşılır karşı örnekler.
- **Zayıf yönleri:** durum uzayı patlaması (state space explosion) — durum sayısı değişken sayısı ile üstel büyür; sürekli (analog) büyüklükler ve sınırsız veri yapıları doğrudan modellenemez, soyutlama gerekir.

Teorem ispatı

Teorem ispatı, sistemi ve istenen özelliği bir mantık dilinde ifade eder; ardından özelliğin sistem tanımından mantıksal çıkarım kurallarıyla türetililebildiğini gösterir. İspat, çoğunlukla bir ispat asistanı (proof assistant) eşliğinde, insan yönlendirmesiyle kurulur. Durum uzayını saymadığı için sonsuz durumlu sistemlerde ve genel (parametrik) özelliklerde çalışabilir: "her N için bu tampon taşmaz" türünden bir iddia, tek bir ispatla kapatılabilir.

Kod düzeyindeki yaygın biçimi sözleşme (contract) tabanlı ispattır: her fonksiyona bir önkoşul (precondition) ve bir artkoşul (postcondition) yazılır; araç bunlardan ispat yükümlülükleri (proof obligation) üretir ve çoğunu otomatik çözücülerle kapatır. Nesne yönelimli tasarımda alt sınıfın temel sınıf sözleşmesini bozmadığı da bu yolla gösterilebilir (bkz. [15. DO-332 ve Nesne Yönelimli Teknoloji ve İlgili Teknikler](#)).

- **Güçlü yönleri:** ifade gücü çok yüksektir; ölçek sınırı model kontrolündeki gibi mekanik değildir; ispat, gerekçesiyle birlikte kalıcı bir kanıt nesnesidir.

- **Zayıf yönleri:** ciddi uzmanlık ister; ispat kurmak emek yoğundur; özellik ihlal edildiğinde otomatik bir karşı örnek üretmez — ispatın "çıkarması" tek başına hata yerini göstermez.

Soyut yorumlama

Soyut yorumlama, programın somut çalışmasını her tek tek yürütmeyi izlemeden, değer kümelerini soyut alanlar (örneğin değer aralıkları) üzerinden **aşırı-yaklaşık** (over-approximation) biçimde hesaplar. Tipik hedefi, çalışma zamanı hatası (run-time error) sınıflarının — taşma, sıfıra bölme, dizi sınırı aşımı, tanımsız davranış — **yokluğunu** kaynak kod üzerinde doğrudan göstermektir. En kötü durum yürütme süresi (worst-case execution time, WCET) ve yığın kullanımı için üst sınır hesaplayan, çoğunlukla nesne kodu üzerinde çalışan analizler de bu ailedendir; bunların doğrulamadaki yeri [9. Yazılım Doğrulama](#) bölümünde anlatılmıştır.

- **Güçlü yönleri:** gerçek kaynak kod üzerinde, model kurmadan, yüksek otomasyonla çalışır; "bu hata sınıfı bu kodda yoktur" gibi kesin negatif iddialar üretir.
- **Zayıf yönleri:** aşırı yaklaşıklık nedeniyle **yanlış alarmlar (false alarm)** üretebilir; her uyarının gerçek hata mı, analiz kabalığı mı olduğunun elle gerekçelendirilmesi gerekir; işlevsel (gereksinim düzeyi) doğruluk için uygun değildir.

Karşılaştırma

Özellik	Model kontrolü	Teorem ispatı	Soyut yorumlama
Otomasyon düzeyi	Yüksek	Düşük-orta (insan yönlendirmeli)	Yüksek
Tipik hedef	Durum makinesi / eşzamanlılık özellikleri	Genel işlevsel doğruluk, algoritma ispatı	Çalışma zamanı hatası yokluğu, WCET
Ölçek sınırı	Durum uzayı patlaması	İspat emeği	Yanlış alarm oranı
Hata bulunduğunda	Karşı örnek verir	Doğrudan vermez	Uyarı verir (yanlış alarm olabilir)
Uzmanlık ihtiyacı	Orta	Yüksek	Orta
Bilinen araç örnekleri	SPIN, nuXmv	SPARK, Frama-C (WP), Isabelle	Astrée, Polyspace Code Prover, Frama-C (Eva)

Araç adları yalnızca aileyi somutlaştırmak içindir; öneri ya da kalifikasyon durumu beyanı değildir.

Bu üç yaklaşım rakip değil, tamamlayıcıdır. Aynı projede durum makinesi mantığı model kontrolüyle, kritik bir aritmetik algoritma ispatla, tüm kaynak kod tabanı ise soyut yorumlamayla doğrulanabilir; her biri farklı bir doğrulama hedefine kanıt üretir.

Biçimsel analiz testin yerini aldığında

DO-178C'de çalıştırılabilir nesne kodu gereksinim tabanlı testle doğrulanır; testin yeterliliği de gereksinim kapsamı ve yapısal kapsam analiziyle sınılanır. Bu testlerin bir kısmı yerine biçimsel analiz konduğunda üç soru doğar: kapsam nasıl gösterilecek, kaynak kodda ispatlanan şey nesne kodu için de geçerli mi ve hangi testler yine de koşullanacak?

Yapısal kapsamın yerini alan ölçütler

Yapısal kapsam analizi testle birlikte anlamlıdır: koşulan testlerin kodun ne kadarını çalıştırdığını ölçer. Çalıştırılmadan ispatlanan bir bileşende ölçülecek bir "çalıştırma" yoktur; ama kapsam analizinin yanıtladığı sorular — gereksinimler eksik mi, doğrulanmamış davranış kaldı mı, gereksinime dayanmayan kod var mı — yanıtı bırakılmaz. DO-333 bu yüzden, yalnızca biçimsel analizle doğrulanana kısım için dört şeyin gösterilmesini ister:

Gösterilecek olan	Pratikteki anlamı
Her gereksinimin tam kapsandığı	Gereksinim, varsayımlarla daraltılmış bir alt kümede değil, geçerli olduğu tüm koşullarda ispatlanmıştır; ispatın dayandığı her varsayım ayrıca doğrulanmıştır
Gereksinim kümesinin tam olduğu	Olası her girdi koşulu için beklenen çıktı tanımlıdır; tanımsız bırakılmış davranış yoktur
İstenmeyen veri akışının bulunmadığı	Kodda, gereksinimlerin öngörmediği bir girdi-çıkı bağımlılığı yoktur
Gereksiz kodun bulunmadığı	Ölü kod (dead code) dahil gereksiz kod (extraneous code) yoktur; devre dışı bırakılmış kod (deactivated code) belirlenmiştir

Son satırdaki kavramlar [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#) bölümünde tanımlanmıştır. Çoğu projede bileşenlerin bir kısmı testle, bir kısmı biçimsel analizle doğrulanır. Testle doğrulanana kod için DO-178C'nin yapısal kapsam ölçütleri aynen geçerlidir; aynı kod parçasında iki yöntem birlikte kullanılıyorsa aralarında boşluk kalmadığı ayrıca gösterilir.

Kaynak koddan nesne koduna özellik korunumu

İspatların çoğu kaynak kod üzerinde yapılır; uçakta çalışan ise derleyicinin ve bağlayıcının ürettiği çalıştırılabilir nesne kodudur. Testte bu fark kendiliğinden kapanır, çünkü test edilen şey nesne kodunun kendisidir. İspatta ise derleyici güven zincirine girer: kaynak düzeyinde gösterilen bir özelliğin nesne kodu hedeflerine sayılabilmesi için o özelliğin derlemeden sonra da korunduğu (property preservation) gösterilmelidir.

Bunun tek bir reçetesi yoktur. Analiz doğrudan nesne kodu üzerinde yapılabilir; derleyicinin ilgili özellikleri koruduğuna ilişkin kanıt sunulabilir (kısıtlı eniyileme (optimization) seçenekleri, derleyici çıktısının ek doğrulaması); ya da kaynak düzeyinde ispatlanan özellikler hedefte koşulan testlerle örneklenerek çapraz denetlenebilir. Hangi yolun izleneceği planlama aşamasında

otoriteyle kararlaştırılır. Her durumda, analizin hedefe ilişkin kabulleri — tamsayı genişlikleri, kayan nokta yuvarlama kipi, hizalama — gerçek derleyici ve işlemciyle aynı olmalıdır; farklı bir hedef modeliyle yapılmış ispat, başka bir programın ispatıdır.

Hedefte kalan testler

Hangi yöntem seçilirse seçilsin, çalıştırılabilir nesne kodunun hedef bilgisayarla uyumluluğunu gösteren donanım/yazılım entegrasyon testleri koşulur. Model gerçek donanımı ancak varsayımlar yoluyla içerir; kesme gecikmeleri, çevre birimi zamanlaması, bellek haritası, açılış sırası ve donanım arızalarına tepki hedefte gözlenir. WCET ve yığın kullanımında da analiz sonucu hedefteki ölçümlerle desteklenir. Pratik sonuç şudur: biçimsel analiz test sayısını azaltır ve testin ağırlığını birim düzeyinden entegrasyon düzeyine kaydırır; test laboratuvarını kapatmaz.

Zorluklar

Biçimsel yöntemlerin sağladığı kanıt gücü bedava gelmez. Bir projede bu yöntemleri plana yazmadan önce dört pratik engelin dürüstçe değerlendirilmesi gerekir.

Uzmanlık gereksinimi

Biçimsel bir özellik yazmak, gereksinim yazmaktan farklı bir beceridir: zamansal mantık (temporal logic), önkoşul/artkoşul sözleşmeleri veya ispat taktikleri gibi kavramlara hâkimiyet ister. Ekipte bu birikim yoksa iki tipik sonuç görülür: ya özellikler o kadar zayıf yazılır ki ispat neredeyse hiçbir şey söylemez, ya da çalışma birkaç uzmanın darboğazına dönüşür. İşe yarayan yaklaşım, yöntemi dar ve yüksek değerli bir alanda (örneğin mod geçiş mantığı) başlatıp ekip birikimini kademeli büyütmeektir.

Ölçeklenebilirlik

Yöntemlerin her biri farklı bir noktada ölçek duvarına çarpar: model kontrolünde durum uzayı patlaması, teorem ispatında ispat emeğinin bileşen sayısı ile büyümesi, soyut yorumlamada büyük kod tabanlarında artan yanlış alarm sayısı. Bu nedenle gerçek projelerde biçimsel analiz genellikle **seçici** uygulanır: sistemin tamamına değil, hata etkisi en ağır olan bileşenlere. "Her şeyi ispatlarız" hedefi, çoğu zaman "hiçbir şeyi bitiremeyiz" ile sonuçlanır.

Biçimsel modelin geçerlenmesi

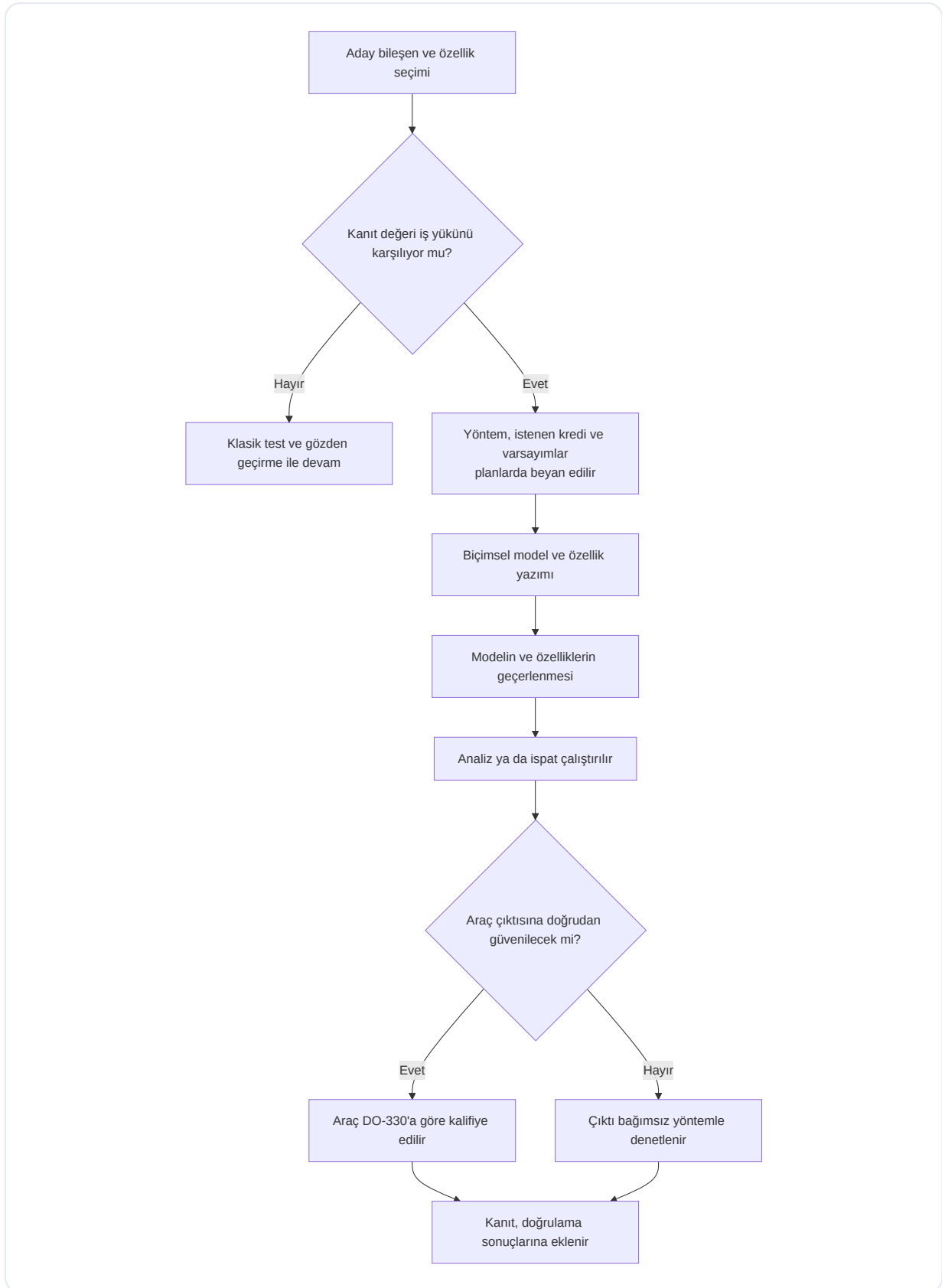
Bir ispat, ancak dayandığı model ve özellikler gerçeği yansıttığı ölçüde değerlidir. Burada iki ayrı soru vardır:

- **Model, gerçek yazılımı/donanımı doğru temsil ediyor mu?** Modelde yapılan her soyutlama (zamanlamanın yok sayılması, değer aralıklarının daraltılması) bir varsayımdır ve kayıt altına alınıp gerekçelendirilmelidir.
- **Yazılan biçimsel özellik, gerçekten kastedilen gereksinimi mi ifade ediyor?** Yanlış formüle edilmiş bir özellik "ispatlanmış ama alakasız" bir sonuç üretir.

Bu yüzden biçimsel özelliklerin gereksinimlere karşı gözden geçirilmesi ve izlenebilirliğinin kurulması, ispatın kendisi kadar önemlidir. İspat doğrulamayı otomatikleştirir; geçerleme (validation) — doğru şeyi ispatladığımızdan emin olma — insan işi olarak kalır. DO-333'ün ek hedefleri tam da bu boşluğa bakar.

Araç kalifikasyonu ihtiyacı

Biçimsel analiz pratikte neredeyse her zaman araçla yürütülür ve çıktısına çoğu zaman elle yeniden denetlenmeden güvenilir; bu da araç kalifikasyonu sorusunu açar. Yöntemin sağlamlığını gerekçelendirmek aracın doğru çalıştığını göstermez; bu güven DO-330'a göre kalifikasyonla kurulur. DO-330 ayrıca araçta uygulanan kuramın olgunluğunun ve bu işe uygunluğunun PSAC'ta gerekçelendirilmesini bekler. Kalifikasyonun derinliğini aracın türü değil, sonucundan alınan kredi belirler. Araç yalnızca kendi otomatikleştirdiği doğrulama faaliyeti için kullanılıyorsa — örneğin kaynak kodun doğruluk ve tutarlılık analizinde — Ölçüt 3'tedir ve her yazılım seviyesinde araç kalifikasyon seviyesi (tool qualification level, TQL) TQL-5'tir. Sonucu ayrıca bazı testleri ya da gözden geçirmeleri yapmamanın gerekçesi yapılıyorsa Ölçüt 2'ye geçer; Seviye A ve B'de TQL-4, Seviye C ve D'de TQL-5 gerekir. Ölçütler ve kalifikasyon verisi [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#) bölümünde anlatılmıştır. Bu, lisans maliyetinin üzerine ek bir planlama ve kanıt yükü getirir. Bazı ispat asistanlarında, ispatın küçük ve bağımsız bir çekirdek tarafından yeniden denetlenebilmesi bu yükü hafifletebilir; ama bu strateji de sertifikasyon otoritesiyle erken aşamada konuşulmalıdır.



Bu engellerin hiçbiri aşılmaz değildir; ancak hepsi planlama aşamasında görünür kılınmalı ve maliyeti kabul edilmelidir. Biçimsel yöntemler, "sona doğru eklenen bir araç" olarak değil, en baştan doğrulama stratejisinin parçası olarak kurgulandığında karşılığını verir.

İki küçük örnek

Aşağıdaki örnekler öğretim amacıyla sadeleştirilmiştir; amaç bir biçimsel özelliğin, bir analiz uyarısının ve bir karşı örneğin neye benzediğini göstermektir.

Aritmetik hata ve sözleşme

Kalan yakıtla uçulabilecek süreyi hesaplayan şu fonksiyonu ele alalım:

```
#include <stdint.h>

#define SAATTEKI_DAKIKA (60)

/* Kalan yakıtla uçulabilecek süre, dakika cinsinden. */
int32_t kalan_sure_dk(int32_t yakit_kg, int32_t akis_kg_saat)
{
    return (yakit_kg * SAATTEKI_DAKIKA) / akis_kg_saat;
}
```

Kod gözden geçirmede masum görünür ve tipik değerlerle yazılmış testlerden geçer. Girdiler hakkında hiçbir bilgi verilmeden çalıştırılan sağlam bir soyut yorumlama aracı ise en az şu iki uyarıyı üretir:

Uyarı	İfade	Hangi koşulda
İşaretsiz tamsayı taşması olabilir	<code>yakit_kg * SAATTEKI_DAKIKA</code>	<code>yakit_kg</code> mutlak değerce yaklaşık 35,8 milyonu aşarsa
Sıfıra bölme olabilir	<code>/ akis_kg_saat</code>	<code>akis_kg_saat</code> sıfırsa

İlki, gerçek yakıt miktarları düşünüldüğünde bir yanlış alarmdır; ama araç bu aralığı bilemez. İkincisi gerçek bir hatadır: motor durduğunda ya da algılayıcı arızasında akış sıfır okunabilir. Çözüm, koda gömülü örtük kabulleri sözleşmeye dönüştürmektir. Aşağıda sözleşme, C için bir belirtim dili olan ACSL (ANSI/ISO C Specification Language) gösterimiyle özel bir yorum bloğuna yazılmıştır:

```
#include <stdint.h>

#define SAATTEKI_DAKIKA      (60)
#define YAKIT_UST_SINIR_KG  (200000)
#define AKIS_ALT_SINIR_KG_SAAAT (100)
#define AKIS_UST_SINIR_KG_SAAAT (20000)

/*@ requires 0 <= yakit_kg <= YAKIT_UST_SINIR_KG;
   @ requires AKIS_ALT_SINIR_KG_SAAAT <= akis_kg_saat <= AKIS_UST_SINIR_KG_SAAAT;
   @ assigns \nothing;
   @ ensures \result == (yakit_kg * SAATTEKI_DAKIKA) / akis_kg_saat;
   @ ensures 0 <= \result;
   @*/
int32_t kalan_sure_dk(int32_t yakit_kg, int32_t akis_kg_saat)
{
    return (yakit_kg * SAATTEKI_DAKIKA) / akis_kg_saat;
}
```

`requires` satırları önkoşul, `ensures` satırları artkoşuldur. Sözleşmedeki aritmetik matematiksel tamsayılar üzerindedir; dolayısıyla ilk artkoşul "sonuç, taşma olmadan hesaplanan değere eşittir" demektir. Bu önkoşullar altında her iki uyarı da kapanır ve artkoşullar ispatlanabilir.

İş burada bitmez: yükümlülük çağırın tarafa geçmiştir. Araç her çağrı noktasında önkoşulun sağlandığını göstermek zorundadır; değer bir algılayıcıdan geliyorsa bu ancak arayüz sınırındaki bir aralık denetimiyle ve akış alt sınırın altındayken hesabın hiç çağrılmamasıyla mümkündür. DO-333'ün "varsayımlar yazılır ve gerekçelendirilir" koşulunun pratikteki karşılığı budur: bir giridi aralığı, ya başka bir ispatla ya da çalışan bir denetim koduyla kapatılana kadar açık kalemdir.

Kilitleme özelliği ve karşı örnek

Gereksinim şöyle olsun: "İtki ters çevirici (thrust reverser) açma komutu yalnızca uçak yerdeyken verilir." Doğrusal zamansal mantıkla (linear temporal logic, LTL) bu özellik tek satırdır; `G` "her zaman" işlecidir:

```
G (ac_komutu -> yerde)
```

Tasarımcı, iniş sırasındaki sekmelerde tekerleklerde ağırlık (weight-on-wheels) sinyali anlık kesildiğinde komut düşmesin diye bir mandal eklemiştir: izin, uçak yerdeyken ters itki kolu (`kol`) çekildiğinde kurulur ve kol çekili kaldıkça korunur. Her çevrimde şu iki atama işletilir:

```
izin      = (yerde VE kol) VEYA (onceki_izin VE kol)
ac_komutu = izin
```

Model kontrol aracı özelliğin sağlanmadığını bildirir ve iki çevrimlik bir karşı örnek verir:

Çevrim	yerde	kol	onceki_izin	izin	ac_komutu	Özellik
1	doğru	doğru	yanlış	doğru	doğru	sağlanıyor
2	yanlış	doğru	doğru	doğru	doğru	ihlal

İkinci çevrimde uçak yerden kesilmiş, kol hâlâ çekilidir ve mandal komutu ayakta tutmaktadır. Bu senaryo, "kol çekiliyken yerde bilgisi düşerse" diye düşünülmediği sürece test listesinde yer almaz; araç ise sormadan bulur.

Karşı örnek aynı zamanda gereksinimdeki bir boşluğu açığa çıkarır. Mandal havada açma komutuna izin vermektedir; ama sekme sırasında komutun anında kesilmesi de istenmeyebilir. Yazılmamış karar şudur: yerde bilgisi ne kadar süre kesilirse komut düşmelidir? Gereksinim bir doğrulama süresiyle netleştirilir, özellik buna göre yeniden yazılır ve model yeniden denetlenir. Model kontrolünün asıl getirisi çoğu zaman budur: ispatın kendisinden önce, gereksinimi kesinleştirmeye zorlaması.

Bu bölümden akılda kalması gerekenler

- DO-333 biçimsel yöntemi, biçimsel model ile sağlam bir biçimsel analizin birleşimi olarak tanımlar; gösterim kesin, yöntem sağlam, varsayımlar yazılı ve gerekçeli olmalıdır.
- Biçimsel analiz bazı gözden geçirme, analiz ve testlerin yerine kanıt olabilir; testi tümüyle ortadan kaldırmaz. Hedef bilgisayarla uyumluluğu gösteren donanım/yazılım entegrasyon testleri her durumda kalır.
- Testin yerini alan ispatta yapısal kapsam yerine gereksinimlerin tam kapsandığı, gereksinim kümesinin tamlığı, istenmeyen veri akışının ve gereksiz kodun bulunmadığı gösterilir; kaynak kodda ispatlanan özelliğin nesne kodunda korunduğu da gerekçelendirilir.
- Üç ana yaklaşım — model kontrolü, teorem ispatı ve soyut yorumlama — rakip değil, tamamlayıcıdır; her biri farklı bir doğrulama hedefine uygundur.
- İspat, doğru şeyin ispatlandığını göstermez: modelin ve özelliklerin geçerlenmesi, biçimselleştirmenin gözden geçirilmesi ve izlenebilirlik insan işi olarak kalır.
- Yöntem, istenen kredi ve araç kalifikasyonu (Ölçüt 2 ya da Ölçüt 3) planlama aşamasında PSAC ve SVP'ye yazılır ve otoriteyle erken konuşulur; uzmanlık ve ölçek maliyeti de o aşamada hesaba katılır.

KISIM V

Özel Konular

17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar

Yapısal kapsam analizinde (structural coverage analysis) testlerin hiç çalıştırmadığı kod, eksik bir testin ya da gereksinimin işareti olabileceği gibi ölü kod (dead code), gereksiz kod (extraneous code) ya da devre dışı bırakılmış kod (deactivated code) da olabilir. Bu bölüm, DO-178C'nin farklı ele aldığı bu üç kavramı ayıran ölçütleri, kapsanmayan kodun kök neden analizini ve her durumda beklenen kanıtı anlatır.

Kapsam ölçütlerinin kendisi ve boşlukların kısa sınıflandırması [9. Yazılım Doğrulama](#) bölümündedir; burada o sınıflandırmanın ayrıntısına ve pratikteki kararlara inilir.

Kapsanmayan kod neden önemlidir?

Yapısal kapsam analizi iki soruya yanıt arar: gereksinim tabanlı testler (requirements-based testing) kod yapısını yazılım seviyesinin istediği ölçüde çalıştırdı mı, ve kodda gereksinimlere dayanmayan, niyet edilmemiş bir işlev var mı? Kapsanmayan kod iki soruyu da açık bırakır: ya testler eksiktir ya da kodda gereksinimlerin istemediği bir şey vardır. Hangisi olursa olsun, çalıştırılabilir nesne kodunda (executable object code) bulunan ama hiçbir testte çalışmamış bir parçanın davranışı bilinmez. O parça bir gün çalışırsa — bozulmuş bir işlev göstercisi, hatalı bir konfigürasyon değeri ya da onu erişilebilir kılan sonraki bir değişiklik yüzünden — ne yapacağını gösteren kanıt yoktur.

Emniyet gerekçesinin yanında gündelik bedeller de vardır. Kaynakta duran ama çalışmayan bir yol, her gözden geçirmede "bu etkin mi?" sorusunu yeniden sordurur. Bakım yapan kişi kodu gereksinimlerle eşleştiremez ve değişikliğinin etkisini kestiremez. Kapsam raporunda açıklanmamış her boşluk da denetimde tek tek hesabı sorulan bir kalemdir.

Üç kavram ve ayırt edici ölçütler

Üç kavramı birbirinden ayıran, kodun görünüşü değil, üç sorunun yanıtıdır:

Kavram	Gereksinime izlenir mi?	Tasarım gereği mi var?	Çalıştırılabilir mi?	Beklenen işlem
Gereksiz kod	Hayır	Hayır	Olabilir	Kaldırılır; kaldırmanın etkisi analiz edilir

Kavram	Gereksinime izlenir mi?	Tasarım gereği mi var?	Çalıştırılabilir mi?	Beklenen işlem
Ölü kod	Hayır	Hayır; bir geliştirme hatasının kalıntısıdır	Hayır; hiçbir işletim konfigürasyonunda	Kaldırılır; gereksiz kodun özel hâlidir
Devre dışı bırakılmış kod	Evet	Evet	Ya hiç ya da yalnızca belirli konfigürasyonlarda	Korunur; istem dışı çalışmasının önlendiği gösterilir

Tabloyu okurken dört noktaya dikkat etmek gerekir:

- **Ölü kod, gereksiz kodun alt kümesidir.** Gereksiz kod çalışabilir de çalışamayabilir de; hiçbir işletim konfigürasyonunda çalıştırılamayan ve bir geliştirme hatası sonucu orada bulunan kısma ölü kod denir. İkisinin de sonu aynıdır: kaldırılır.
- **Tanımlar veri için de geçerlidir.** Hiç okunmayan bir sabit tablo ya da kullanılmayan bir değişken de aynı ölçütlerle değerlendirilir; kod için "çalıştırılma" ne ise veri için "kullanılma" odur.
- **Düzye önemlidir.** Ölü kod ve devre dışı bırakılmış kod, çalıştırılabilir nesne kodunda bulunan kod ve veri üzerinden tanımlanır. Gereksiz kod ise kaynak kod ya da nesne kodu düzeyinde bulunabilir.
- **Ölçüt kodun yaşı değildir.** Önceden geliştirilmiş bir bileşenin kullanılmayan eski kısmı, gereksinime izleniyor ve tasarım kararıyla açıklanıyorsa devre dışı bırakılmış koddur. Gereksinimi silindiği hâlde yerinde unutulmuş eski kod ise gereksiz koddur.

Sık yapılan yanlış sınıflandırmalar

Kapsam raporundaki her kırmızı satır ölü kod değildir; her "bilerek bıraktık" açıklaması da devre dışı bırakılmış kod üretmez. Sahada en çok karıştırılan yapılar şunlardır:

Yapı	Sık yapılan hata	Doğru ele alış
Savunmacı programlama (defensive programming) yapısı; ör. ulaşılmaması gereken <code>default</code> dalı	Ölü kod sayıp silmek	Ölü kod değildir; gereksinim ya da tasarım dayanağı gösterilir, testle ulaşılamayan kısım analizle gerekçelendirilir
Gömülü tanımlayıcı (embedded identifier); ör. imaja (image) yerleştirilmiş parça numarası, sürüm dizgesi	Kullanılmayan veri diye ölü kod saymak	Ölü kod değildir; varlık nedeni tasarımda açıklanır

Yapı	Sık yapılan hata	Doğru ele alış
Bağlanan ama çağrılmayan kütüphane ya da işletim sistemi işlevi	Ölü kod saymak ya da hiç anmamak	Gereksinime izleniyor ve tasarım gereği çalıştırılmıyorsa devre dışı bırakılmış koddur; planlarda beyan edilir ve etkinleşemeyeceği gösterilir. Hiçbir gereksinime izlenmiyorsa gereksiz koddur
Derleyicinin ya da bağlayıcının imaja almadığı kaynak kod	"İmajda yok, sorun yok" deyip bırakmak	Gereksinime izlenmiyorsa kaynak düzeyinde gereksiz koddur; kaldırılır ya da aşağıdaki iki koşul sağlanır
Koşullu derleme (conditional compilation) ile dışarıda bırakılan kod	Çalışma zamanında kapatılan kodla aynı kanıtı yeterli saymak	İmajda bulunmaz; sözlük tanımı çalıştırılabilir nesne kodunu esas aldığından tanımın harfiyle devre dışı bırakılmış kod değildir, ama uygulamada o başlık altında planlandığı da görülür. Her derleme varyantı ayrı doğrulanır
Derleyicinin eklediği, kaynak koda doğrudan izlenemeyen nesne kodu	Gereksiz ya da devre dışı bırakılmış kod saymak	Aralık ve dizi indisi denetimi gibi eklemeler gerçekleştirmenin parçasıdır; Seviye A'da belirlenir ve doğruluğu ek doğrulamayla gösterilir

Savunmacı yapı etiketi kolayca kötüye kullanılır: açıklanamayan her boşluğa "savunmacı" demek cazip gelir. Denetimde sorulan soru bellidir: bu yapıyı hangi gereksinim, hangi tasarım kararı ya da kodlama standardının hangi kuralı istiyor? Yanıt yoksa yapı savunmacı değil, gereksizdir.

Kütüphane ve gerçek zamanlı işletim sistemi (real-time operating system, RTOS) işlevleri ayrı bir dikkat ister, çünkü bağlayıcı çoğu zaman projenin çağırmadığı işlevleri de imaja çeker. Bunların hangi sınıfa girdiğini kullanılmamaları değil, ardındaki veri belirler; ayrıntısı [20. Gerçek Zamanlı İşletim Sistemleri](#) bölümünde ve [Ek B: Gerçek Zamanlı İşletim Sistemlerinde Endişe Alanları](#) sayfasındadır. Önceden geliştirilmiş yazılımın (previously developed software, PDS) kullanılmayan kısımları için aynı soru [24. Yazılım Yeniden Kullanımı](#) bölümünde ele alınır.

Kaynakta bulunan ama imaja girmeyen gereksiz kod için DO-178C dar bir kapı bırakır: kodun çalıştırılabilir nesne kodunda bulunmadığı analizle gösterilir ve sonraki derlemelere girmesini önleyen prosedürler tanımlanırsa kod kaynakta kalabilir. Eniyileme düzeyi ya da bağlayıcı seçeneği değiştiğinde aynı kod imaja geri dönebileceği için ikinci koşul birincisi kadar önemlidir. Temiz çözüm yine kaldırmaktır.

Ölü ve gereksiz kodun ele alınması

Ölü kod ve gereksiz kod çoğunlukla yapısal kapsam analizi sırasında kendini gösterir: gereksinim tabanlı testlerin tamamı koşulmuş, ama bazı kod parçaları hiç çalışmamıştır. Buradaki kritik nokta şudur: kapsanmayan kod, kendi başına bir sonuç değil, bir **belirtidir**. Asıl iş, bu belirtinin arkasındaki kök nedeni bulmaktır.

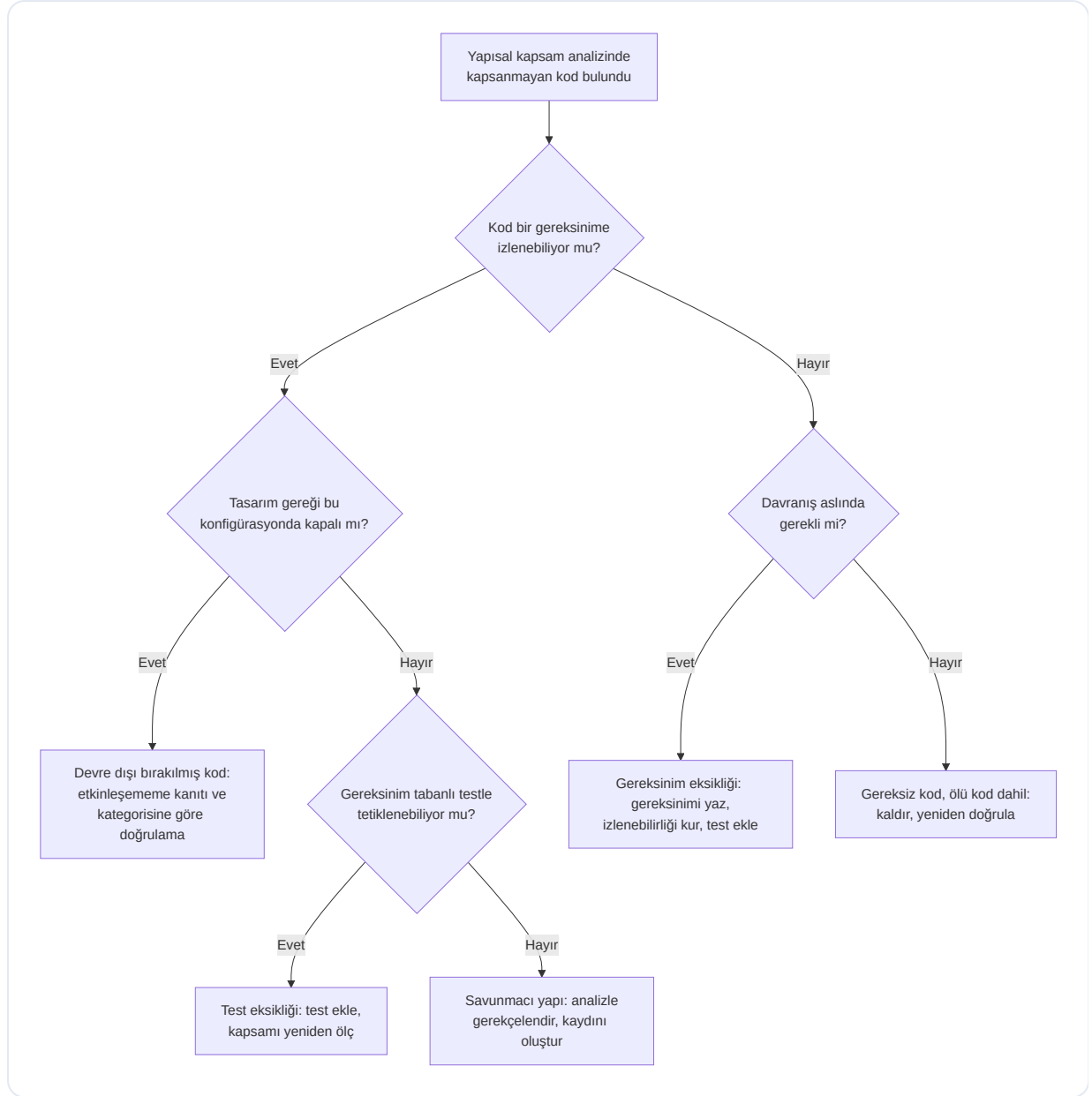
Kök neden analizi

DO-178C, kapsanmayan kod için dört olası neden sayar:

1. **Test eksikliği** — Kod bir gereksinimi gerçekleştiriyor, ama gereksinim tabanlı test durumları ya da prosedürleri o yolu tetiklememiş. Çözüm: test durumu (test case) eklemek ya da mevcut testi güçlendirmek; boşluğu gereksinim kapsam analizinin neden yakalamadığına da bakılır.
2. **Gereksinim eksikliği** — Kod gerekli bir davranışı gerçekleştiriyor, ama bu davranış hiçbir yüksek ya da düşük seviyeli gereksinimde yazmıyor. Çözüm: gereksinimi yazmak, izlenebilirliği (traceability) kurmak ve ardından testi eklemek.
3. **Gereksiz kod (ölü kod dahil)** — Kod hiçbir gereksinime izlenmiyor: kopyala-yapıştır kalıntısı, terk edilmiş bir tasarım denemesi, unutulmuş bir hata ayıklama işlevi. Çözüm: kaldırmak.
4. **Devre dışı bırakılmış kod** — Kod gereksinime izleniyor ve tasarım gereği bu konfigürasyonda çalışmıyor. Çözüm: korumak ve aşağıda anlatılan kanıtı üretmek.

Savunmacı programlama yapıları bu sınıflandırmada özel bir yer tutar: kod gereksinime ya da tasarıma izlenir ve her konfigürasyonda etkindir, ama onu tetikleyecek koşul test ortamında üretilemeyebilir. Testle ulaşılamayan kısım analizle gerekçelendirilir.

İzlenebilirlik sorusu işlev düzeyinde değil, kapsanmayan yapının düzeyinde sorulur. İşlevin bütünü bir gereksinime bağlı olabilir; önemli olan, çalışmayan dalın ya da deyimini o gereksinimin hangi cümlesini gerçekleştirdiği. Bu ayrımı akışa dökersek:



"Davranış aslında gerekli mi?" sorusuna kolayca "evet" denmemelidir. Kodu haklı çıkarmak için sonradan gereksinim yazmak tersine mühendisliktir: gereksinim gerçek bir ihtiyacı anlatmalı, gözden geçirilmeli ve türetilmiş gereksinimse (derived requirement) sistem süreçlerine geri bildirilmelidir (bkz. [25. Tersine Mühendislik](#)).

İki kısa örnek

İlki ölü kod. Aşağıdaki işlevde dıştaki koşul hızın üst sınırı aşmadığını zaten güvenceye alıyor; içteki son `else` dalına hiçbir girdiyle ulaşılamaz:

```

#define HIZ_UST_SINIR_KNOT    (350u)
#define HIZ_BANT_SINIRI_KNOT (150u)
#define KAZANC_DUSUK_HIZ     (12)
#define KAZANC_YUKSEK_HIZ    (7)
#define KAZANC_VARSAYILAN    (0)

bool kazanc_sec(uint16_t hiz_knot, int32_t *kazanc)
{
    bool gecerli = false;

    if ((kazanc != NULL) && (hiz_knot <= HIZ_UST_SINIR_KNOT)) {
        if (hiz_knot < HIZ_BANT_SINIRI_KNOT) {
            *kazanc = KAZANC_DUSUK_HIZ;
        } else if (hiz_knot <= HIZ_UST_SINIR_KNOT) {
            *kazanc = KAZANC_YUKSEK_HIZ;
        } else {
            *kazanc = KAZANC_VARSAYILAN; /* ulaşılamaz: dış koşul bu durumu dışlar */
        }
        gecerli = true;
    }

    return gecerli;
}

```

Kapsam raporunda bu durum iki izle görünür: son atama deyimi hiç çalışmamıştır ve `else if` kararı hiç yanlış sonuçlanmamıştır. Dal, aralık denetimi dışarı taşınırken içeride unutulmuş bir kalıntıdır; hiçbir gereksinim "varsayılan kazanç" diye bir davranış tanımlamaz. Çözüm ikinci koşulu ve ölü dalı silmektir. Aynı satırlar bir tasarım kararına dayansaydı savunmacı yapı sayılırdı; sınıfı belirleyen kodun biçimi değil, dayanağıdır. Derleyici bu dalı eniyileme sırasında atabilir; o zaman kod imajda yoktur ama kaynakta gereksiz kod olarak durmaya devam eder.

İkincisi ölü olmayan gereksiz kod. Bakım portundan gelen komutları işleyen tabloda bir geliştirme kalıntısı duruyor:

```

static const komut_girdisi_t komut_tablosu[] = {
    { KOMUT_DURUM_SORGULA, durum_sorgula },
    { KOMUT_OZ_TEST,      oz_test_baslat },
    { KOMUT_BELLEK_DOKUMU, bellek_dokumu_gonder } /* gereksinimi yok */
};

```

Hiçbir gereksinim bellek dökümü komutundan söz etmediği için hiçbir test bu komutu göndermez; kapsam raporunda `bellek_dokumu_gonder()` işlevinin tamamı çalışmamış görünür. Kod ölü değildir: komut kodu porta ulaştığı anda çalışır ve davranışı hiç doğrulanmamıştır. Gereksiz kodun emniyet açısından asıl rahatsız edici türü budur.

Kaldırma mı, gerekçelendirme mi?

Varsayılan karar **kaldırmaktır**. Gereksinime izlenemeyen kodun kaynak kodda kalması, hem gözden geçirme yükünü artırır hem de gelecekteki değişikliklerde istem dışı etkinleşme (inadvertent enabling) riski taşır. Kaldırma kararı verildiğinde değişiklik, olağan değişiklik sürecinden geçer: problem raporu (problem report) açılır, değişiklik etki analizi (change impact analysis) yapılır, etkilenen doğrulama faaliyetleri (yeniden test, yeniden gözden geçirme) tekrarlanır ve konfigürasyon yönetimi (configuration management) kayıtları güncellenir. "Sadece siliyoruz, test gerekmez" yaklaşımı tehlikelidir; silme işlemi de bir kod değişikliğidir ve derleyicinin üreteceği çalıştırılabilir nesne kodunu değiştirebilir.

Kaldırmanın pratik olmadığı durumlar da yaşanır — örneğin sertifikasyon sürecinin çok geç bir aşamasında bulunan, işlevsel etkisi olmayan küçük bir artık. Kodu bu sürümde bırakmak standardın öngördüğü bir çözüm değildir; standarttan sapma olarak ele alınır ve uygulamada üç şey gerektirir: sertifikasyon otoritesiyle önceden mutabakat, kodun hiçbir koşulda emniyeti etkilemeyeceğini gösteren ve sistem emniyet değerlendirme süreciyle desteklenen bir analiz, ve konunun açık problem raporu olarak sunulması. Gerekçe "vakit yoktu"dan fazlasını söylemelidir. Açık raporların nasıl sınıflandırıldığı [9. Yazılım Doğrulama](#) bölümünde anlatılır.

Durum	Tipik karar	Beklenen kanıt
Test eksikliği	Test durumu ekle	Yeni test durumu ve prosedürü, güncel kapsam sonucu
Gereksinim eksikliği	Gereksinim ve test ekle	Güncel gereksinim, izlenebilirlik kaydı, test
Gereksiz kod (ölü kod dahil)	Kaldır	Problem raporu, değişiklik etki analizi, yeniden doğrulama
Savunmacı yapı	Analizle gerekçelendir	Gereksinim ya da tasarım dayanağı, yazılı analiz, gözden geçirme kaydı
Devre dışı bırakılmış kod	Koru	Planlarda beyan, etkinleşememe kanıtı, kategorisine göre doğrulama
Geç aşamada bulunan gereksiz kod	Otoriteyle mutabakat; sonraki sürümde kaldır	Emniyet analizi, açık problem raporu

Sonuç ne olursa olsun, her kapsanmayan kod bulgusu ve verilen karar **kanıt dosyasına yansıtılmalıdır**: bulgular ve gerekçeleri yapısal kapsam analizi sonuçlarında, açık kalan konular ise yazılım başarı özetinde (Software Accomplishment Summary, SAS) görünür olmalıdır (bkz. [12. Sertifikasyon İrtibatı](#)). Denetçinin sormadan önce cevabı dosyada bulması, sürecin olgunluğunun en iyi göstergesidir.

Devre dışı bırakılmış kodun ele alınması

Devre dışı bırakılmış kod, ölü ve gereksiz koddan temel bir noktada ayrılır: varlığı **kasıtlıdır**, gereksinime izlenir ve tasarım kararıyla açıklanır. DO-178C iki kategoriye ayırır ve doğrulama beklentisi kategoriye göre değişir:

- **Birinci kategori: sertifikalı hiçbir ürünün güncel konfigürasyonunda çalıştırılması amaçlanmayan kod** — önceden geliştirilmiş bir bileşenin kullanılmayan kısmı, kullanılmayan kütüphane işlevi, gelecekteki büyüme için eklenmiş kod.
- **İkinci kategori: hedef ortamın yalnızca belirli onaylı konfigürasyonlarında çalışan kod** — donanım pinleriyle ya da yazılımla programlanan bir seçenекle etkinleşen işlev.

Meşru kullanım senaryoları

Sahada en sık karşılaşılan durumlar şunlardır:

- **Opsiyonel donanım:** Aynı yazılım, bazı uçaklarda takılı olan bir sensörü veya harici bir üniteyi destekler; donanım yoksa ilgili sürücü kodu hiç çalışmaz.
- **Müşteri konfigürasyonları:** Tek bir yazılım parçası, konfigürasyon verisiyle farklılaşan birden çok müşteri seçeneğini barındırır (birim tercihleri, opsiyonel ekran sayfaları vb.).
- **Uçak tipi/model farklılıkları:** Aynı ekipman ailesi birden çok platformda kullanılır; platforma özgü yollar diğer platformlarda devre dışıdır.
- **Bakım ve fabrika modları:** Yalnızca yerde, özel bir bağlantı veya komutla etkinleşen test ve kalibrasyon işlevleri.
- **Yeniden kullanılan bileşenler ve kütüphaneler:** Başka bir proje için geliştirilmiş bileşenin ya da bir kütüphanenin, bu projenin çağırmadığı ama imaja giren işlevleri.
- **Gelecek sürüm işlevleri:** Kodda hazır bekleyen ama bu sertifikasyon temel çizgisinde (baseline) etkinleştirilmeyen özellikler. Bu senaryo meşru olmakla birlikte en riskli olanıdır ve otoriteyle erken konuşulmalıdır.

İstem dışı etkinleşmeye karşı önlemler

Devre dışı bırakılmış kodun kabul edilebilirliği, tek bir soruya verilen cevaba bağlıdır: **bu kod, hedef konfigürasyonda istem dışı çalışabilir mi?** Tasarımdan beklenen, devre dışı kodun etkin işlevleri olumsuz etkilemesini önleyen bir mekanizmanın bilerek tasarlanıp gerçekleştirilmesidir. Bozulmuş bir program sayacı gibi anormal sistem koşullarından doğan istem dışı çalışmayı ise standart, etkin kodun istem dışı çalışmasından ayrı tutmaz. Pratikte iki ayırım yolu görülür:

1. **Derleme zamanı ayırımı** — Kod, koşullu derleme ile nesne koduna hiç girmez. En güçlü yalıtımı sağlar; çalıştırılabilir nesne kodunda kod fiilen yoktur. Sözlük tanımı çalıştırılabilir nesne kodunu esas aldığından bu kod tanımın harfiyle devre dışı bırakılmış kod sayılmaz; standart bu durumu ayrıca ele almaz, uygulamada ise yine bu başlık altında planlandığı sık

görülür. Adı ne olursa olsun kanıt yükü etkinleşememeyi göstermekten, kodun imajda gerçekten bulunmadığını teyit etmeye, her derleme varyantını ayrı bir çalıştırılabilir nesne kodu olarak doğrulamaya ve derleme seçeneklerini konfigürasyon kontrolünde tutmaya kayar.

2. **Çalışma zamanı ayrımı** — Kod nesne kodunda vardır; bir konfigürasyon parametresi (pin programlama, konfigürasyon dosyası, strap girişi) onu kapalı tutar. Bu durumda parametrenin kendisi de doğrulanması gereken bir veri hâline gelir ve kapalı yolun kapalı kaldığı testle gösterilmelidir. Parametre ayrı bir dosyada tutuluyorsa parametre verisi ögesi (parameter data item, PDI) kuralları da devreye girer (bkz. [22. Konfigürasyon Verisi](#)).

Basit bir çalışma zamanı örneği:

```
/* Konfigürasyon verisi: uçak kablaından okunan pin programlama girişi. */
typedef enum {
    SENSOR_YOK = 0,
    SENSOR_VAR = 1
} sensor_konfig_t;

void opsiyonel_sensoru_oku(void);
void ekran_sayfasini_guncelle(void);
void varsayilan_degeri_yayimla(void);
void konfigurasyon_arizasi_bildir(void);

void sensor_kanalini_isle(sensor_konfig_t konfig)
{
    if (konfig == SENSOR_VAR) {
        /* Opsiyonel donanım takılıysa çalışan yol. */
        opsiyonel_sensoru_oku();
        ekran_sayfasini_guncelle();
    } else if (konfig == SENSOR_YOK) {
        /* Sensör yolu çalıştırılmaz; varsayılan emniyetli değer yayımlanır. */
        varsayilan_degeri_yayimla();
    } else {
        /* Tanımsız konfigürasyon değeri: sensör yolu kapalı kalır, arıza bildirilir. */
        varsayilan_degeri_yayimla();
        konfigurasyon_arizasi_bildir();
    }
}
```

Bu örnekte `opsiyonel_sensoru_oku()` yolunun "sensörsüz" konfigürasyonda asla çalışmayacağı, `konfig` değerinin nereden geldiğine ve nasıl doğrulandığına bağlıdır. Tanımlı iki değer dışındaki her değer kapalı tarafa düşer ve arıza olarak bildirilir; bozuk bir değer sessizce "sensör var" diye yorumlanması önlenmiştir. Pin programlama girişinin açık devre, kısa devre gibi arıza durumlarında hangi değere düştüğü de analiz edilmelidir; aksi hâlde bir kablaj arızası, devre dışı kodu istem dışı etkinleştirebilir.

Gereken doğrulama kanıtı

Devre dışı bırakılmış kod da gereksinime izlenir ve geliştirilmesi etkin kodla aynı hedeflere (objective) tabidir: gereksinimi, tasarımı ve gözden geçirmesi vardır. Bunun üzerine beklenen kanıt seti kabaca şöyledir:

- **Planlarda beyan:** Devre dışı kodun varlığı ve ele alınma yöntemi yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) baştan açıklanır, doğrulama yaklaşımı yazılım doğrulama planında (Software Verification Plan, SVP) yer alır; denetimde sürpriz olarak çıkmaz (bkz. [5. Yazılım Planlama](#)).
- **Tasarımda tanım:** Hangi kodun hangi konfigürasyonda devre dışı olduğu ve onu kapalı tutan mekanizma, yazılım mimarisi ve tasarım tanımında açıkça belirtilir.
- **Birinci kategori için etkinleşememe kanıtı:** Kodu çalıştırabilecek yollar (çağrı, işlev göstericisi, konfigürasyon değeri) tek tek çıkarılır; her birinin kapalı, yalıtılmış ya da hiç var olmadığı analizle ortaya konur ve kapatma mekanizması testle sınanır. Bu koda farklı bir yazılım seviyesi atanacaksa gerekçesi sistem emniyet değerlendirme sürecinden, doğrulaması hafifletilecekse gerekçesi yazılım geliştirme süreçlerinden gelir; ikisi de PSAC'a yazılır.
- **İkinci kategori için kendi doğrulaması:** Kodun çalıştığı konfigürasyon kurulur ve kapsam hedefleri ek test durumları ve prosedürleriyle o konfigürasyonda karşılanır. Hedef konfigürasyon için ise yolun kapalı kaldığı gösterilir.
- **Konfigürasyon yönetimi kaydı:** Hangi temel çizgide hangi konfigürasyonun geçerli olduğu ve hangi kodun devre dışı kaldığı izlenebilir olmalıdır.

Gelecek sürüm işlevleri birinci kategoriye girer; bu sürümde yapısal kapsam açısından beklenen, etkinleşememelerinin gösterilmesidir. Ancak "nasılsa kapalı" diyerek gereksinimsiz ve gözden geçirmesiz bırakılan kod, etkinleştirileceği gün borç olarak geri döner: o gün kendi konfigürasyonunda testleri ve kapsam analizi tamamlanmadan kredi alınamaz.

Gereksiz kodla devre dışı bırakılmış kodu ayıran çizgi tam da budur: devre dışı kod gereksinime izlenir, tasarım kararıyla açıklanır ve **doğrulanmış bir kapatma mekanizması** vardır. Bu izlenebilirlik yoksa kod gereksiz kod — çalışması olanaksızsa ölü kod — olarak ele alınır ve kaldırılması beklenir.

Önleme: sorunu kapsam analizinden önce yakalamak

Kapsam analizi bu kodları bulmanın son ve en pahalı yeridir; bulgu oraya kaldığında testler yazılmış, temel çizgiler atılmıştır. Daha erken ve ucuz üç denetim noktası vardır:

- **Kod gözden geçirmesinde izlenebilirlik denetimi.** Her işlevin, hatta her dalın hangi gereksinimi gerçekleştirdiği sorulur; bağlanamayan kod o anda açıklanır ya da silinir (bkz. [8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#)).
- **Statik analiz ve derleyici uyarıları.** Erişilemeyen kod, hiç çağrılmayan işlev ve kullanılmayan değişken, araçların iyi bulduğu kusurlardır. Bağlayıcının harita dosyası da imaja hangi işlevlerin gerçekten girdiğini gösterir.
- **Kapsamın geliştirme boyunca izlenmesi.** Kapsam verisi testler yazıldıkça toplanırsa her boşluk, bağlamı tazeyken tek tek çözülür.

Bir de süreç alışkanlığı vardır: bir gereksinim silindiğinde ya da değiştiğinde, ona izlenen kod ve testler aynı değişiklik kapsamında ele alınır. Gereksiz kodun en yaygın kaynağı, gereksinimi kaldırılıp kodu yerinde bırakılan işlevlerdir. Otoritenin kapsam boşluklarında baktığı noktalar [SW SOI-3](#) kontrol listesindedir.

Bu bölümden akılda kalması gerekenler

- Kapsanmayan kod bir belirtidir; önce kök neden bulunur: test eksikliği, gereksinim eksikliği, gereksiz kod (ölü kod dahil) ya da devre dışı bırakılmış kod.
- Sınıfı kodun görünüşü değil üç soru belirler: gereksinime izleniyor mu, tasarım gereği mi var, çalıştırılabilir mi? Aynı ölçütler veri için de geçerlidir.
- Ölü kod gereksiz kodun özel hâlidir. İkisi de kaldırılır; kaldırma da bir değişikliktir, etkisi analiz edilir ve gereken doğrulama yinelenir.
- Savunmacı yapılar, gömülü tanımlayıcılar ve koşulları sağlanan kullanılmayan kütüphane işlevleri ölü kod değildir; ama her biri dayanağını göstermek zorundadır.
- Devre dışı bırakılmış kod gereksinime izlenir, planlarda beyan edilir ve kategorisine göre doğrulanır: ya istem dışı çalışmasının önlendiği gösterilir ya da çalıştığı konfigürasyonda test edilir.
- Gereksiz kodu geç aşamada yerinde bırakmak standardın yolu değildir; otorite mutabakatı, emniyet analizi ve açık problem raporu gerektirir.
- Her bulgu ve karar kanıt dosyasına yansıtılır; denetçi cevabı dosyada bulmalıdır.

18. Sahada Yüklenebilir Yazılım

Sahada yüklenebilir yazılım (field-loadable software, FLS), ekipman uçaktan sökülmeden bir veri portu üzerinden yüklenebilen yazılımdır. Bu esneklik bütünlük (integrity), uyumluluk ve konfigürasyon kimliği şartlarını beraberinde getirir.

Bu bölüm, DO-178C'nin sahada yüklemeyi neden bir sistem emniyet konusu olarak ele aldığını, emniyet değerlendirmesinden hangi somut tasarım şartlarının türediğini ve yükleme sürecinin konfigürasyon yönetimi ile bilgi güvenliği ayaklarını açıklar.

Sahada yükleme ne demektir?

DO-178C'nin tanımı kısadır: sahada yüklenebilir yazılım, sistem ya da ekipman kurulu olduğu yerden — tipik olarak uçaktan veya motordan — sökülmeden yüklenebilen yazılımdır (§2.5.5). Standartta yazılım, çalıştırılabilir nesne kodunu (executable object code) da veriyi de kapsadığı için tanım **yazılımı da veri tablolarını da** içine alır: yüklenen şey çalıştırılabilir kod olabileceği gibi, davranışı belirleyen bir veri kümesi de olabilir (yüklenebilir verinin kendi yaşam döngüsü için bkz. [22. Konfigürasyon Verisi](#)). Kavramın karşıtı, fabrika yüklemeli yazılımdır (factory-loadable software): orada yazılım, birimin mührü bozulup kutusu açılarak ya da devre kartı programlanarak yüklenir. FLS'de ise aktarım bir veri portu üzerinden yapılır; birim yerinden sökülmez, kutusu açılmaz. Yükleme genellikle hangarda bir yer veri yükleyicisi (data loader) aracılığıyla yapılır; yetkili bir onarım istasyonunda veya servis merkezinde veri portundan yapılan yüklemeler de aynı kapsamda değerlendirilir.

Taşıyıcı ortam teknolojiyle birlikte değişti — disket setlerinden optik disklere, USB belleklere ve bugün ağ üzerinden dağıtım — ama bu bölümde anlatılan ilkeler ortamdaki bağımsızdır. Sahada yüklenebilirlik belirli bir ekipman sınıfına özgü de değildir: emniyet önlemleri alınmış ve her sürüm otoritece onaylanmışsa, uçuş kumandaları ve motor kontrolü gibi en kritik işlevleri taşıyan birimler de bu yolla güncellenir.

Baştan netleştirilmesi gereken üç ayrım vardır:

- **Onaylanan şey medya değil, imajdır.** İmaj (image), çalıştırılabilir nesne kodunun ve ona eşlik eden verinin hedef bilgisayara yüklenen biçimidir. Sertifikasyon açısından onaylanan ürün, yazılımı taşıyan disk, USB bellek veya sunucudaki dosya değil, bu imajın kendisidir. Medya yalnızca taşıyıcıdır; parça numarası (part number) ve bütünlük kontrolleri imaja aittir.
- **FLS ile kullanıcı tarafından değiştirilebilir yazılım (user-modifiable software, UMS) aynı şey değildir.** UMS, ilk sertifikasyonda onaylanan değişiklik kısıtları içinde kaldığı sürece kullanıcının otorite gözden geçirmesi olmadan değiştirebildiği yazılımdır; FLS ise yazılımın ekipmana hangi yoldan ulaştığını anlatır. Onaylı yazılımın her yeni FLS sürümü kendi

geliştirme, doğrulama ve onay sürecinden geçer; UMS için ayrı koşullar geçerlidir. İki kavram birbirini dışlamaz: değiştirilebilir bileşen de sahada yüklenebilir ve o zaman iki konunun koşulları birlikte ele alınır (bkz. [19. Kullanıcı Tarafından Değiştirilebilir Yazılım](#)).

- **Havacılık veri tabanları FLS gibi ele alınmaz.** Seyrüsefer veya arazi veri tabanları da sahaya yüklenir; ancak bunlar DO-178C'nin değil, havacılık verisinin işlenmesini düzenleyen DO-200B'nin kapsamına girer ve kendi güvence zinciriyle yönetilir (bkz. [23. Havacılık Verileri](#)).

Faydaları ve zorlukları

Sahada yüklenebilir yazılım yaklaşımının en görünür faydası, bir yazılım güncellemesi için donanımın uçaktan sökülüp üretici tesisine gönderilmesine gerek kalmamasıdır. Klasik yaklaşımda yazılım, hat değiştirilebilir birimin (line replaceable unit, LRU) fabrikada programlanan kalıcı bir parçasıdır; her yazılım değişikliği, birimin sökülmesi, yeniden programlanması ve uçağa geri takılması demektir. Sahada yükleme bu döngüyü, hangarda birkaç saatlik bir bakım işlemine indirger.

Filo ölçeğinde bakıldığında fayda daha da belirginleşir:

- **Hızlı hata düzeltme:** Operasyonda keşfedilen bir yazılım hatası, tüm filoya haftalar yerine günler içinde dağıtılabilir.
- **Kademeli işlev ekleme:** Yeni işlevler, donanım değişikliği olmadan sonraki yazılım sürümleriyle devreye alınabilir.
- **Yedek parça sadeleşmesi:** Aynı donanım parça numarası farklı yazılım sürümleriyle kullanılabilirdiğinden, depoda tutulması gereken donanım çeşidi azalır.
- **Filo yönetimi:** Hangi uçakta hangi yazılım sürümünün bulunduğu merkezi olarak izlenebilir ve güncelleme kampanyaları planlı biçimde yürütülebilir.

Bu esnekliğin bedeli, konfigürasyon yönetimi (configuration management) yükünün belirgin biçimde artmasıdır. Yazılım artık donanımın içine gömülü tek bir bütün değil, kendi parça numarasına sahip ayrı bir konfigürasyon ögesidir. Üstelik bu yük tek bir tarafın omzunda da değildir: yazılım ve ekipman geliştiricisi, uçak veya motor üreticisi, havayolu ve sertifikasyon otoritesi (certification authority) aynı sürecin paydaşlarıdır ve zincirin her halkası kendi payına düşeni yönetmek zorundadır. Başlıca zorluklar şunlardır:

- **Parça numarası yönetimi:** Donanım parça numarası ile yazılım parça numarası ayrışır; her yazılım sürümü ayrı bir parça numarası alır ve uçak kayıtlarında ayrı izlenir. Yanlış parça numarasının yüklenmesi, fiziksel olarak yanlış parçanın takılmasıyla eşdeğer bir bakım hatasıdır.
- **Uyumluluk matrisi:** Hangi yazılım sürümünün hangi donanım revizyonu, hangi komşu sistem sürümleriyle ve hangi uçak konfigürasyonu ile birlikte kullanılabileceği bir uyumluluk matrisinde tanımlanır ve her sürümde güncellenir. Matris dışı bir kombinasyon, tek tek onaylı iki parçanın birlikte onaysız bir sistem oluşturması anlamına gelir.

- **Onaylı yükleme prosedürleri:** Yükleme işlemi, bakım dokümantasyonunda tanımlı, eğitilmiş personelce uygulanan ve kayıt altına alınan onaylı bir prosedürle yapılır. Yükleme sonrasında sürüm doğrulaması yapıp bakım kaydına işlenmeden uçak servise verilmez.
- **Sertifikasyon kanıtı:** Yükleme mekanizmasının kendisi de (yer ekipmanı, veri yükleyici, uçaktaki yükleme yazılımı) güvenilirliğini gösteren kanıtlarla desteklenmelidir; bütünlük kontrolleri yeterince güçlüyse aktarım zincirinin her halkasını ayrı ayrı kalifikasyona tabi tutmak gerekmebilir, ancak bu gerekçe açıkça yazılmalıdır.
- **Mevzuatın yorumlanması:** Parça işaretleme ve onarım istasyonu kuralları gibi düzenlemeler, uçak ve motorlar ile onlara takılan donanım düşünülerek yazılmıştır. FLS ile yazılım bağımsız bir parça hâline gelince, donanım için yazılmış bu kuralların yazılım düzeyinde nasıl uygulanacağı yorum gerektirir; ekipman, uçak ve operasyon seviyelerinin her birinde ayrı düzenlemeler devreye girer.

Boyut	Fayda	Karşılığında gelen yük
Bakım süresi	Söküm yok, hangarda güncelleme	Onaylı prosedür ve kayıt zorunluluğu
Hata düzeltme	Filoya hızlı dağıtım	Her sürüm için ayrı parça numarası
Donanım lojistiği	Daha az donanım çeşidi	Yazılım/donanım uyumluluk matrisi
İşlev geliştirme	Donanımsız işlev ekleme	Sürüm başına yeniden doğrulama kapsamı

Deneyim şunu gösteriyor: sahada yükleme kararı geç alındığında, faydalar aynı kalır ama zorluklar katlanır. Bu dengeyi lehinize çevirmenin tek gerçekçi yolu, bu bölümün geri kalanının ana teması olan "baştan tasarım" yaklaşımıdır.

DO-178C'nin bakışı: yükleme bir sistem emniyet konusudur

DO-178C sahada yüklemeyi öncelikle standardın sistem yönlerini anlatan 2. bölümünde (§2.5.5), yani bir **sistem tasarımı konusu** olarak ele alır; yazılım süreçlerinde konuya yalnızca birkaç noktada döner (entegrasyon testleri, konfigürasyon kimliği, yükleme kontrolü ve yazılım konfigürasyon indeksi (Software Configuration Index, SCI) — sırası geldikçe anılacak). Yaklaşımın özü şudur: yükleme işlevine ilişkin emniyet gereksinimleri sistem gereksinimlerinin parçasıdır ve hangi gereksinimlerin gerektiği sistem emniyet değerlendirme sürecinde (system safety assessment process) belirlenir (bkz. [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#)). Başka bir deyişle "yükleme nasıl yapılır" sorusundan önce "yükleme neyi bozabilir" sorusu cevaplanır.

Standardın bu değerlendirmede ele alınmasını beklediği hususlar dört soruda toplanabilir; her biri aşağıdaki alt bölümlerde anlatılan somut tasarım şartlarına dönüşür:

Soru	Emniyet değerlendirmesinin ele aldığı husus	Tipik tasarım karşılığı
Yüklenen şey sağlam mı?	Bozuk ya da yarım kalmış yüklemenin tespiti	Yükleme sırasında ve her açılışta bütünlük kontrolü
Yüklenen şey doğru şey mi?	Uyumsuz yazılımın yüklenmesinin etkileri; donanım/yazılım, yazılım/yazılım ve uçak/yazılım uyumluluğu	Paket üst verisiyle (metadata) hedef doğrulama, uyumluluk matrisi, uyumsuz yüklemenin reddi
Yükleme yanlış zamanda başlayabilir mi?	Yükleme işlevinin istem dışı etkinleşmesi (inadvertent enabling)	Tekerleklerde ağırlık + bakım modu kilitleri
Yüklü olan biliniyor mu?	Konfigürasyon kimliği gösteriminin kaybı ya da bozulması	Tekil parça numarası ve elektronik parça işaretleme

Bu hususların ortak paydası, hepsinin sistem seviyesinde gereksinim ve tasarım işi olmasıdır; dolayısıyla emniyet ekibi sürece **erken** katılmalıdır. Yükleme stratejisi emniyet değerlendirmesinden bağımsız kurulursa kusur çoğu zaman ancak yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC) otoriteyle paylaşıldığında, yani mimari çoktan oturduktan sonra görünür olur; o noktada düzeltme, projenin geç bir evresinde sistem seviyesinde yeniden tasarım demektir.

Bütünlük tespiti ve seviye ataması

Sistem, bozuk veya yarım kalmış bir yüklemeyi tespit edebilmelidir; asıl soru bu tespit mekanizmasının **hangi seviyede** geliştirileceğidir. DO-178C'nin kuralı yalındır: tespit mekanizmasına, yüklenen yazılımı kullanan işlevle ilişkili en ağır arıza durumu (failure condition) sınıfı ya da en yüksek yazılım seviyesi (software level) atanır — sistem emniyet değerlendirme süreci aksini gerekçelendirmedikçe. Seviye A yazılım taşıyan bir birimde açılıştaki bütünlük kontrolü — çoğu tasarımda bir döngüsel artıklık denetimi (cyclic redundancy check, CRC) — bu yüzden Seviye A olarak geliştirilir; "bu yalnızca birkaç satırlık bir CRC hesabı" diyerek seviyeyi düşürmek, ancak emniyet değerlendirmesiyle savunulabiliyorsa mümkündür.

Pratik görünümü: imaj, hem yükleme sırasında hem de **her açılışta** doğrulanır. Doğrulama başarısızsa uygulama hiç başlatılmaz; birim başlatma (boot) veya bakım modunda kalır ve durumunu raporlar. Yaklaşım "kabul et, sonra düzelt" değil, "doğrulanamayanı hiç çalıştırma"dır.

Bir bütünlük kontrolünün var olması, **yeterli** olduğu anlamına gelmez. n bitlik bir kontrol değeri, rastgele bozulmuş bir imajı kabaca 2 üzeri –n olasılıkla kaçırır (32 bit için yaklaşık dört milyarda bir); ayrıca bir CRC polinomunun belirli sayıda bit hatasını kesin yakalama garantisi yalnızca sınırlı bir veri uzunluğuna kadar geçerlidir. Megabaytlarca imajı tek bir kısa kontrol değeriyle korumak bu yüzden savunulamaz: ya kontrol değeri uzatılır ya da imaj ayrı ayrı korunan bloklara bölünür. Hangi yol seçilirse seçilsin kaçırma olasılığının yüklenen yazılımın seviyesiyle bağdaştığı gösterilir; bu gerekçe, ileride görüleceği gibi planlarda yer alır.

Uyumluluk üç ekseninde doğrulanır

- **Donanım/yazılım:** Hedef birimin donanım parça numarası ve revizyonu, paketin üst verisindeki hedef tanımıyla eşleşmelidir; yanlış hedefe yükleme reddedilir.
- **Yazılım/yazılım:** Birbiriyle konuşan veya yedekli çalışan birimlerin sürümleri, birlikte onaylanmış bir kombinasyon oluşturmalıdır. En sinsi senaryo filo geçiştir: güncelleme kampanyası sırasında aynı uçakta sol birim yeni, sağ birim eski sürümde kalabilir. Tipik örnek çift kanallı motor kontrolüdür: iki tam yetkili sayısal motor kontrol biriminden (full authority digital engine control, FADEC) yalnızca biri güncellenebiliyorsa, ortaya çıkan eski-yeni kombinasyonun ya açıkça onaylanmış olması ya da ikisinin birlikte güncellenmesinin prosedür ve mekanizmayla garanti edilmesi gerekir. Aynı ilke tek bir birimin içinde de geçerlidir: yazılım birden çok konfigürasyon ögesinden oluşuyorsa (uygulama, parametre verisi, başlatma yazılımı) bunların birbiriyle uyumu da güvence altına alınır.
- **Uçak/yazılım:** Aynı ekipman, farklı uçak tiplerinde veya konfigürasyonlarında farklı yazılım sürümleri gerektirebilir; paketin o uçak konfigürasyonuna uygunluğu doğrulanır.

Bu kontroller, yükleme kabul edilmeden önce paket üst verisi üzerinden otomatik yapılır. DO-178C yükleme işlevini destek sistemleri ve prosedürleriyle bir bütün olarak görür: yanlış yazılım, donanım ve uçak birleşimleri yakalanabilmeli, koruma da ilgili işlevin arıza durumuyla orantılı olmalıdır. Prosedür korumanın parçasıdır, ama arıza durumu ağırlaştıkça "dikkatli olun" demesi yetmez; koruma mekanizmaya dayanmalıdır. Yanlış ya da bozuk yükleme her şeye rağmen gerçekleşir ve sistem bunu algılayıp bir **varsayılan moda** ya da güvenli duruma düşüyorsa, bu moddaki davranış da tanımsız bırakılamaz: sistemin bölümlenmiş her bileşeni için bu moda geçiş ve bu modda çalışma emniyet gereksinimleriyle tanımlanır. Hangi işlevlerin hangi durumda olduğu, mürettebatın (flight crew) nasıl uyarıldığı ve arayüzdeki komşu sistemlerin bu modla doğru çalışıp çalışmadığı bu kapsamda ele alınır.

İstem dışı etkinleşmeye karşı koruma

Yükleme işlevinin istem dışı — özellikle uçuşta — etkinleşmesi başlı başına bir arıza durumu doğurabilir; DO-178C böyle bir durumda yükleme işlevi için sistem gereksinimlerine bir emniyet gereksinimi yazılmasını ister. Pratikte bu, yüklemenin yalnızca emniyetli koşullarda etkinleştirilebilmesi demektir. Tipik kilit kombinasyonu:

- tekerleklerde ağırlık (weight-on-wheels) sinyali — uçak yerde,
- ayrık bakım girişi veya bakım modu seçimi — bilinçli bir bakım eylemi,
- gerekiyorsa ek koşullar (motorların çalışmıyor olması, park freni gibi).

Uçuşta yüklemenin engellenmesi tek bir yazılım bayrağına bırakılmaz; donanım ve yazılım kilitleri birlikte kullanılır. Kilidin kendisi de arıza analizine girer: tekerleklerde ağırlık sinyali yanlış tarafta arızalanırsa yükleme kapısı açılıyor mu?

Konfigürasyon kimliği: yüklü olan neyse, görünen o olmalı

Her FLS parçası **tekil bir parça numarası** taşır ve yazılım değiştiğinde parça numarası da değişir. Donanım kutusunun üzerindeki fiziksel etiket artık içindekini anlatmaz; bu yüzden yüklü parça numarası uçak üzerinde **elektronik olarak doğrulanabilir** olmalıdır (elektronik parça işaretleme — electronic part marking): bakım terminalinden sorgulama, ARINC 429 kelimeleriyle yayınlama veya kokpit/bakım ekranında gösterme gibi. DO-178C'nin konfigürasyon tanımlama faaliyetleri de aynı yere varır: ürünün kimliği fiziksel bakışla belirlenemiyorsa çalıştırılabilir nesne kodu ve varsa parametre verisi ögesi (parameter data item, PDI) dosyaları, sistemin başka parçalarının erişebildiği bir konfigürasyon kimliği taşır (§7.2.1).

İncelikli nokta şudur: uçağın onaylı konfigürasyona uygunluğu bu gösterim işleviyle kanıtlanıyorsa, gösterimin kendisinin bozulması da değerlendirilmelidir; yanlış parça numarasını "doğru" diye göstermek, hiç göstermemekten daha tehlikelidir. Bu nedenle gösterim mekanizmasının parçası olan yazılım ya yüklenecek yazılımların en yüksek yazılım seviyesinde geliştirilir ya da uçtan uca kimlik kontrolünün bütünlüğü emniyet değerlendirmesiyle ayrıca gerekçelendirilir.

Kutunun üzerindeki donanım parça numarası ile içindeki yazılımın parça numarası arasındaki ilişki de baştan karara bağlanır: kimi üretici her yazılım yüklemesini donanımın parça numarasına ya da modifikasyon durumuna yansıtır, kimi iki numarayı birbirinden bağımsız yönetir. Hangisinin seçildiğinden çok, seçimin sistem tasarımında açıkça yapılmış ve kayıt, etiketleme, gösterim mekanizmalarının ona göre kurulmuş olması önemlidir. Bu bölümdeki anlatım, günümüzde yaygın olan ayrı yazılım parça numarası yaklaşımını izler.

Sistemin sahada yüklenebilir tasarlanması

Sahada yüklenebilirlik, sonradan eklenebilecek bir özellik değildir; bellek düzeninden gereksinim setine kadar sistemin pek çok katmanını etkiler. Baştan tasarlandığında dört ana yapı taşı öne çıkar: yükleme arayüzü, paket biçimi, bütünlük mekanizmaları ve yükleme sonrası kimlik raporlama.

Yükleme arayüzü. Uçaktaki birim ile yer ekipmanındaki veri yükleyici arasındaki fiziksel ve mantıksal arayüz erken tanımlanmalıdır. Endüstride yaygın uygulama, standartlaşmış veri yükleme protokolleridir: ARINC 615 bu iletişimi ARINC 429 veri yolu üzerinden, ARINC 615A ise Ethernet tabanlı ağlar üzerinden tanımlar; yer tarafında havayollarının kullandığı taşınabilir çok amaçlı erişim terminali için de ARINC 644A rehberlik sağlar. Özel arayüzler de mümkündür, ama her özel çözüm yer ekipmanı tarafında ek geliştirme ve bakım yükü demektir. Arayüz tasarımında yalnızca "mutlu yol" değil; bağlantı kopması, zaman aşımı ve yarıda kesilen aktarım gibi durumların davranışı da gereksinim olarak yazılmalıdır.

Paket biçimi. Yüklenen şey tek bir ikili dosya değil, üst veri ile birlikte paketlenmiş bir yazılım parçasıdır. İyi bir paket biçimi en azından şunları içerir:

- yazılım parça numarası ve sürüm bilgisi,
- hedef donanım tanımı (hangi birime, hangi donanım revizyonuna),
- dosya listesi ve her dosyanın bütünlük değeri,
- paketin tamamını kapsayan bir bütünlük değeri.

Bu alanlar sayesinde yükleyici, aktarıma başlamadan önce "bu paket bu birime uygun mu" sorusunu yanıtlayabilir; yanlış paketin yanlış birime yüklenmesi mekanizma düzeyinde engellenir. Bu alanların endüstrideki standart karşılığı ARINC 665'tir: yüklenebilir yazılım parçasının (loadable software part, LSP) başlık yapısını, dosya ve yük CRC'lerini, parça numarası biçimini ve medya setlerini tanımlar. ARINC 665'e uyum, farklı üreticilerin yükleyicileri ile hedef birimleri arasında birlikte çalışabilirlik sağlar.

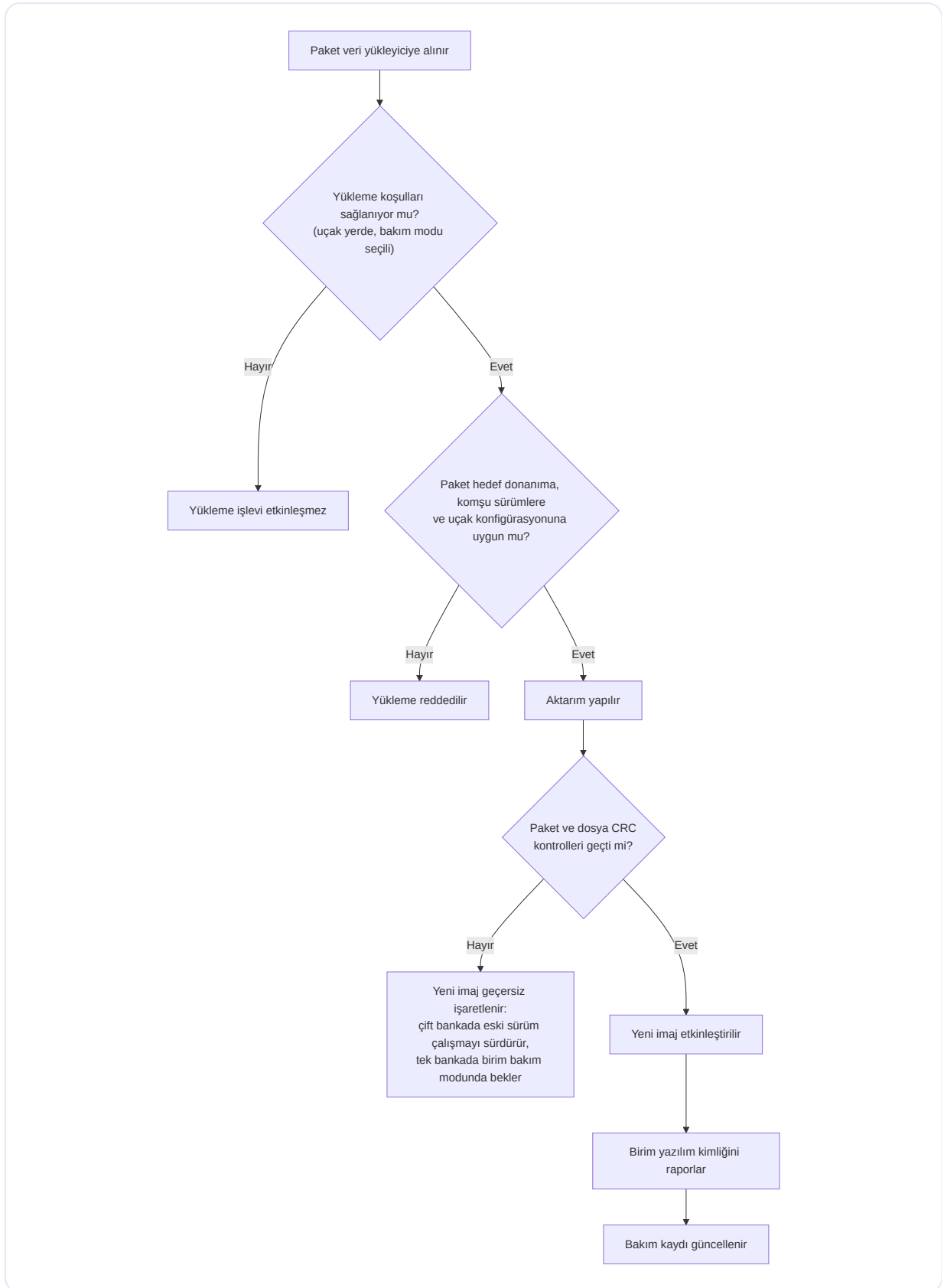
Bütünlük mekanizmaları. Aktarım ve saklama sırasındaki bozulmayı yakalamak için her dosyaya ve paketin tamamına bir bütünlük değeri — sağlama toplamı (checksum) ya da CRC — eklenir. Yukarıda anlatılan ilke burada koda dönüşür: imaj bellekten okunup hesaplanan değer beklenen değerle eşleşmeden uygulama başlatılmaz.

```
/* Açılışta yazılım bölgesinin bütünlük kontrolü (kavramsal örnek) */
uint32_t hesaplanan = crc32_hesapla(yazilim_baslangic, yazilim_boyut);

if (hesaplanan == yuklu_paket.beklenen_crc) {
    uygulamayi_baslat();
} else {
    /* Bozuk imaj: uygulama başlatılmaz, birim yükleme/bakım modunda bekler */
    hata_kaydi_yaz(HATA_IMAJ_BUTUNLUK);
    yukleme_moduna_gec();
}
```

Yükleme sonrası kimlik raporlama. Elektronik parça işaretleme yüklemeye anındaki karşılığıdır: yükleme bittiğinde birim, yazılım parça numarasını, sürümünü ve bütünlük değerini bakım terminaline veya kokpit ekranına raporlar. Bu rapor, bakım personelinin "yüklemeye başarılı" kararını dayandırdığı nesnel kanıttır ve uçak konfigürasyon kayıtlarına işlenir.

Dört yapı taşı ile yukarıda anlatılan etkinleştirme kilitleri tek bir yükleme akışında birleşir:



Bu yapı taşlarının ortak amacı şudur: yükleme sürecinin her adımı ya doğrulanabilir biçimde başarılıdır ya da sistem bilinen güvenli bir duruma (eski sürüm veya yükleme modu) döner. Ara durumda kalmış, kimliği belirsiz bir birim kabul edilemez.

Yarıda kalan yükleme ve güvenli durum

Yarıda kalan yükleme bir istisna değil, tasarımın baştan hesaba kattığı bir senaryodur: güç kesilebilir, kablo çekilebilir, aktarım hata verebilir. İki yaygın mimari cevap vardır:

- **Çift banka (dual bank):** Yeni imaj pasif bellek bankasına yazılır ve doğrulama tamamlanmadan aktif işaretlenmez. Yükleme yarıda kalırsa aktif bankadaki eski sürüm el değmemiş durur; birim bir sonraki açılışta bilinen konfigürasyonla kalkar. Bedeli, iki kat imaj belleğidir.
- **Tek banka + kalıcı başlatma yazılımı:** Uygulama alanı silinip yeniden yazılır. Yükleme yarıda kalırsa açılıştaki bütünlük kontrolü imajı reddeder; birim, sahada değiştirilmeyen ve ayrı korunan başlatma yazılımı sayesinde yükleme/bakım modunda bekler ve durumunu raporlar. Birim, yükleme tamamlanana kadar servis dışıdır.

İki mimaride de ilke aynıdır: birim hiçbir zaman kimliği belirsiz bir yazılımla uçuşa verilmez — ya bilinen onaylı sürüm çalışır ya da birim bakım modunda bekler. Eski sürüme bilinçli geri dönüş (rollback) de sıradan bir yükleme işlemidir: aynı uyumluluk ve bütünlük kontrollerinden geçer ve dönülen sürümün o uçak konfigürasyonu için hâlâ onaylı olması gerekir.

Geliştirme ve planlara yansımaları

Sahada yüklenebilir olmak, yazılıma hiçbir doğrulama muafiyeti kazandırmaz: FLS de her uçuş yazılımı gibi DO-178C'ye (ve uygulanabilir eklerine) göre geliştirilir. Fark, üzerine binen ek tasarım yükümlülükleridir — bütünlük kontrolü, koruma kilitleri ve çoğu tasarımda uçuş uygulamasından **ayrı bir yükleme uygulaması** (birimde kalıcı duran başlatma/bakım yazılımı). Bu ek parçalar da onaylı yazılımın bileşenidir ve kendi gereksinimleriyle geliştirilip doğrulanır.

Sahada yükleme kararının en görünür olduğu yer planlardır (bkz. [5. Yazılım Planlama](#)). DO-178C'nin buradaki payı sınırlıdır: FLS'yi PSAC'ın ek hususları arasında sayar, yükleme tasarımının tasarım tanımında yer almasını bekler ve donanım/yazılım entegrasyon testlerinin yakalaması beklenen hatalar arasında, yüklenen yazılımın doğruluğunu ve uyumluluğunu teyit eden mekanizmaların yanlış çalışmasını anar (§11.1, §11.10, §6.4.3). Gerisi otorite rehberlerinden ve yerleşik uygulamadan gelir; tipik dağılım şöyledir:

Plan	Sahada yüklemeğe özgü içerik
PSAC	Yükleme stratejisi; seçilen bütünlük kontrolünün yazılım seviyesi için yeterliliğinin gerekçesi; geliştirme sonrası konfigürasyon yönetimi sorumluluğunun kimde olduğu
Yazılım geliştirme planı (Software Development Plan, SDP)	Koruma mekanizmalarının ve bütünlük kontrollerinin geliştirilme yaklaşımı; uyulan endüstri standartları (örneğin ARINC 615A, ARINC 665)

Plan	Sahada yüklemeye özgü içerik
Yazılım doğrulama planı (Software Verification Plan, SVP)	Bütünlük kontrolünün, koruma kilitlerinin ve yükleme uygulamasının doğrulanma yöntemi; kesilen ve bozuk yükleme senaryolarının test kapsamı
Yazılım konfigürasyon yönetimi planı (Software Configuration Management Plan, SCMP)	FLS parça numaralandırması, medya tanımı ve yükleme prosedürlerinin yönetimi; teslim sonrası sorumluluk devri

Planların özellikle cevaplaması gereken soru şudur: **geliştirme bittikten sonra konfigürasyon yönetimini kim yürütecek?** Geliştirme sırasında FLS'nin konfigürasyon yönetimi, sahada yüklenemeyen yazılımınkiyle büyük ölçüde aynıdır; ayrışma teslimle başlar. Sahadaki yüklemelerin kaydını kimin tutacağı, uyumluluk matrisini kimin güncelleyeceği planlarda tanımlanmamışsa, bu sorular filo büyüdükçe cevapsız kalır.

Yükleme sürecinin güvencesi ve konfigürasyon yönetimi

Her yükleme işleminin tek tek doğrulanması gerekip gerekmediği, güvencenin nereye yaslandığına bağlıdır:

- Bütünlük **hedef taraftaki kontrollerle** (başlık doğrulama, dosya ve imaj CRC'leri, açılış kontrolü) garanti ediliyorsa, her yüklemenin ayrıca doğrulanması gerekmez: mekanizma bir kez geliştirilip doğrulanır, sahadaki her yükleme onun güvencesinden yararlanır.
- Güvence **yer tarafındaki yükleyiciye** dayanıyorsa — örneğin hedef birim gelen veriyi sorgusuz kabul ediyorsa — bu kez yükleyicinin kendisinin doğrulanmış ve konfigürasyon kontrolü altında olması gerekir: araç kalifikasyonu (tool qualification) sorusu gündeme gelir (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)), yükleyici yazılımının DO-178C veya DO-330'a göre geliştirilmesi ve yüklemenin yalnızca bu onaylı yükleyiciyle yapılması şart koşulabilir. Hedef taraflı kontroller olgunlaştığı için bu yaklaşım günümüzde nadiren tercih edilir.

Konfigürasyon yönetimi tarafında sahada yükleme, DO-178C'nin yazılım yükleme kontrolü (software load control) başlığına girer; aynı başlık fabrikada programlanmış bellek aygıtlarının takılmasını da kapsar (bkz. [10. Yazılım Konfigürasyon Yönetimi](#)). Yöntem hangisi olursa olsun iki şey istenir: onaya sunulan yazılım konfigürasyonlarını belirleyen parça numaralandırma ve medya tanımlama prosedürleri ile yazılımın ekipman donanımıyla uyumluluğunu teyit eden kayıtlar (§7.4). SCI de buna göre kurulur: çalıştırılabilir nesne kodunu ve varsa PDI dosyalarını kimlikleriyle, arşiv ve sürüm medyasını, kullanılıyorsa nesne kodunun bütünlük kontrol değerlerini ve yazılımı hedef donanıma yükleme prosedürlerini tanımlar (§11.16); yazılım uygunluk gözden geçirmesi (software conformity review) de onaylı yazılımın yayımlanmış talimatlarla yüklenebildiğini teyit eder (§8.3). Yükleme prosedürleri yazılım ekibinin değil de

sistem veya uçak seviyesindeki dokümantasyonun sorumluluğundaysa, SCI en azından bunların nerede olduğunu açıkça söylemelidir. Sahadaki fiilî konfigürasyonun izlenebilirliği ise bakım kayıtları ve filo konfigürasyon yönetimiyle sağlanır.

Yüklemeyi bu iş için yetkilendirilmiş personel yapar. Aşağıdaki koşullar, bugünkü AC 20-115D / AMC 20-115D metninde madde madde sayılan şartlar değildir; önceki otorite rehberlerinden bu yana endüstride yerleşmiş iyi uygulamayı özetler. Her yüklemede:

1. Yüklene yazılım sürümü sertifikasyon otoritesince onaylıdır.
2. Onaylı yükleme prosedürü izlenmiştir.
3. Donanım/yazılım kombinasyonu onaylıdır: yazılım, üzerine yüklendiği donanım için onaylanmış sürümdür.
4. Uçak veya motor konfigürasyonu onaylıdır — yedekli birimler dahil; karışık eski–yeni sürüm kombinasyonu ancak açıkça onaylıysa bırakılabilir.
5. Yükleme eksiksiz ve hatasız tamamlanmıştır: hesaplanan bütünlük değeri, yükleme talimatında verilen beklenen değerle karşılaştırılır.
6. Yükleme sonrası parça numarası (varsa sürüm numarasıyla birlikte) arayüzden okunarak teyit edilmiştir.
7. Konfigürasyon değişikliği, uçak veya motor konfigürasyon kayıtlarına ve ilgili bakım kayıtlarına işlenmiştir.

İşin bakım ve işletme tarafı bugün ayrı rehberlerle ele alınır: FAA'da uçak bakımı sırasında yazılım yönetimini AC 43-216A (2023) anlatır; filo çapında yüklenebilir yazılımın yönetimi için de ARINC 667 endüstri rehberliği sağlar.

Sahada yüklenebilir yazılımın değiştirilmesi

Sahada yüklemenin varlık nedeni, yazılımın ekipman üreticiye dönmeden değiştirilebilmesidir; ama bu kolaylık, değişikliğin kendisine uygulanan disiplini hafifletmez. FLS'de yapılan her değişiklik, diğer uçuş yazılımlarıyla aynı yoldan geçer:

- Süreç, değişiklik etki analiziyle (change impact analysis) başlar: hangi gereksinimler, kod ve veriler değişti; hangileri dolaylı olarak etkilendi?
- Değişen ve etkilenen her öge yeniden doğrulanır — ve doğrulama, kurulumun hedeflendiği donanım ile uçak veya motor konfigürasyonu üzerinde geçerli olmalıdır.
- Değiştirilmiş yazılım, kuruluma alınmadan **önce** o ekipman ve uçak/motor konfigürasyonu için otorite onayı alır; yeni parça numarası uyumluluk matrisine ve konfigürasyon kayıtlarına işlenir.

Sahada yüklemenin kazandırdığı hız, doğrulama ve onaydan kırpılan zaman değil, söküm–nakliye–takma döngüsünden kazanılan zamandır. "Küçük bir yamaydı, hangarda yükleyiverdik" cümlesi, bu bölümün anlattığı çerçevenin tam karşıtıdır.

Bütünlük yetmez: bilgi güvenliği boyutu

Bu bölümdeki kontrollerin tamamı **kazara bozulmayı** hedefler: aktarım hatası, bellek bozulması, yanlış paket. DO-178C'nin çerçevesi budur — bütünlüğü kapsar, kötü niyetli müdahaleyi kapsamaz. Oysa CRC'yi, imajı bilerek değiştiren biri de yeniden hesaplayabilir; sağlama toplamı sahtecilik karşısında koruma sağlamaz.

Özgünlük (authenticity) ve müdahale tespiti gereksinimleri, uçuşa elverişlilik güvenliği (airworthiness security) standartlarından gelir: süreç için DO-326A, yöntemler için DO-356A, sürekli uçuşa elverişlilik için DO-355A (süreç standardının Avrupa'daki karşılığı 2024'te ED-202B olarak güncellendi). Yeni bir veri yükleme sistemi veya mevcut sistemde büyük bir değişiklik, günümüzde bir güvenlik risk analiziyle desteklenir.

Endüstri pratiğinde bu ihtiyaç, açık anahtar altyapısına dayalı **dijital imza** ile karşılanır: yüklenebilir parçalar üreticide imzalanır; elektronik dağıtım paketleri için ARINC 827, parça imzalama için ARINC 835 kullanılır. İmza, parçanın geliştiricinin elinden çıktığı andan uçağa yüklendiği ana kadar kaynağının ve içeriğinin doğrulanabilmesini sağlar. Dijital imza, açılıştaki hızlı bütünlük kontrolünün yerine geçmez; saklama sırasındaki bozulmayı yakalamak yine CRC'nin işidir — iki mekanizma farklı tehditlere karşı birlikte çalışır.

Onay çerçevesi: hangi doküman neyi anlatır

Sahada yüklenebilir yazılımın kuralları tek bir dokümanda toplanmış değildir; üç katmana yayılır: standardın sistem seviyesi hükümleri, otorite rehberleri ve endüstri standartları.

Doküman	Katkısı
DO-178C §2.5.5 (ayrıca §6.4.3, §7.2.1, §7.4, §11.16)	Sistem tasarımı hususları: emniyet değerlendirmesinin ele alacağı yükleme riskleri; yazılım süreçlerinde test, kimlik, yükleme kontrolü ve SCI karşılıkları
AC 20-115D / AMC 20-115D (8. bölüm)	Geliştiriciye düşen şartlar (FAA ve EASA metinleri teknik olarak aynıdır)
AC 43-216A	Uçak bakımı sırasında yazılım yönetimi (operatör ve bakım kuruluşu tarafı)
ARINC 615 / 615A	Yükleyici–hedef birim protokolü (ARINC 429 / Ethernet üzerinden)
ARINC 644A	Taşınabilir çok amaçlı erişim (bakım) terminali rehberi
ARINC 665	Yüklenebilir parça ve medya biçimi, parça numarası şeması, CRC'ler
ARINC 667	Filo çapında yüklenebilir yazılım yönetimi
ARINC 827 / 835	Elektronik dağıtım paketi ve dijital imza
DO-326A, DO-356A, DO-355A	Uçuşa elverişlilik güvenliği süreci, yöntemleri ve sürekli uçuşa elverişlilik

Doküman	Katkısı
DO-200B	Havacılık veri tabanları (FLS sayılmaz, kendi güvence zinciri vardır); 2024'te DO-200C yayımlandı

Otorite rehberi katmanının geliştiriciden istedikleri dört başlıkta özetlenebilir: sistem seviyesi hususları destekleyecek bilgiyi sağlamak; yazılımı kendi yazılım seviyesiyle orantılı bir bütünlük düzeyinde bozulmaya ve kısmi yüklemeye karşı korumak; yükleme sonrasında parça numarasının uçak üzerinde doğrulanabilir olmasını sağlamak; ve yükleme işlevinin uçuşun emniyet açısından kritik safhalarında istem dışı etkinleşmesini önleyecek korumaları koymak.

Tarihçe notu: FAA tarafında sahada yüklenebilir yazılımın onayı uzun yıllar Order 8110.49'un bu konuya ayrılmış bölümlerinde, EASA tarafında ise CM-SWCEH-002 sertifikasyon notunda düzenlendi. 2017'de FLS'ye ilişkin hükümler sadeleştirilerek AC 20-115D / AMC 20-115D'ye alındı; EASA'nın notu AMC 20-115D ile yürürlükten kalktı, Order'ın 2018'de yayımlanan 8110.49A sürümünde de bu bölümler artık yer almıyor. Eski projelerin planlarında bu iki dokümana atıf görmek bu yüzden olağandır; yeni bir projede dayanak AC/AMC 20-115D'dir. Değişmeyen ilke şudur: onaylanan şey taşıyıcı medya değil, hedef bilgisayara yüklenen imajdır.

Bu bölümden akılda kalması gerekenler

- FLS, ekipman uçaktan veya motordan sökülmeden veri portu üzerinden yüklenebilen yazılım **veya veri tablolarıdır**; onay medyaya değil imaja verilir. FLS, kullanıcı tarafından değiştirilebilir yazılımla da havacılık veri tabanıyla da aynı şey değildir; ikincisi DO-200B'ye tabidir.
- DO-178C sahada yüklemeyi sistem emniyet konusu olarak ele alır: bozuk/kısmi yükleme tespiti, üç eksenli uyumluluk (donanım, yazılım, uçak/motor), istem dışı etkinleşme ve kimlik gösterimi sistem emniyet değerlendirme sürecinde ele alınır; emniyet ekibinin geç katılımı, geç bir sistem yeniden tasarımı riskidir.
- Bozuk ya da yarım yüklemeyi tespit eden mekanizmaya, emniyet değerlendirmesi aksini gerekçelendirmedikçe, yüklenen yazılımı kullanan işlevin en ağır arıza durumu sınıfı ya da en yüksek yazılım seviyesi atanır; kontrolün yeterliliği imaj boyutuna göre gösterilir; doğrulanamayan imaj hiç çalıştırılmaz.
- Sahada yüklenebilirlik baştan tasarlanır: yükleme arayüzü (ARINC 615/615A), üst verili paket biçimi (ARINC 665), her açılıшта bütünlük kontrolü ve kimlik raporlama sistemin parçasıdır. Yarıda kalan yüklemede birim ya bilinen onaylı sürümle çalışır ya da bakım modunda bekler; kimliği belirsiz birim uçuşa verilmez.
- FLS de DO-178C'ye göre geliştirilir; planlar yükleme stratejisini, bütünlük kontrolünün yeterlilik gerekçesini ve **teslim sonrası konfigürasyon yönetimi sorumluluğunu** açıkça tanımlar. Değişiklik de aynı yoldan geçer: değişiklik etki analizi, yeniden doğrulama ve kurulum öncesi otorite onayı; kazanılan hız doğrulamadan değil, söküm–nakliye döngüsünden gelir.

- Faydanın bedeli konfigürasyon yönetimi yüküdür: her FLS parçası tekil parça numarası taşır, uyumluluk matrisi her sürümde güncellenir. Yükleme kontrolü parça numaralandırma ve medya tanımlama prosedürleri ile donanım uyumluluğu kayıtlarını ister; SCI yüklenen kodun kimliğini ve yükleme prosedürlerini tanımlar; her yükleme onaylı prosedürle yapılır, teyit edilir ve kayda işlenir.
- Bütünlük, güvenlik değildir: CRC kazara bozulmayı yakalar; kasıtlı müdahaleye karşı dijital imza gerekir (DO-326A ailesi, ARINC 827/835).

19. Kullanıcı Tarafından Değiştirilebilir Yazılım

Kullanıcı tarafından değiştirilebilir yazılım (user-modifiable software, UMS), kullanıcının onaylı kısıtlar içinde, otorite gözden geçirmesi olmadan değiştirebildiği yazılımdır. Bu bölüm UMS'yi benzer kavramlardan ayırır; değiştirilebilirliğin tasarımda nasıl kurulup korunduğunu ve teslimden sonra değişikliklerin nasıl yönetildiğini anlatır.

UMS nedir, ne değildir?

Tanımın üç ögesi vardır. Birincisi **niyet**: yazılım, kullanıcı değiştirsin diye tasarlanmıştır; sonradan açılmış bir arka kapı değildir. İkincisi **kısıt**: neyin, hangi aralıkta, hangi araç ve prosedürle değiştirilebileceği ilk sertifikasyonda belirlenir. Üçüncüsü **gözden geçirmeden muafiyet**: bu kısıtlar içinde kalan bir değişiklik için sertifikasyon otoritesine (certification authority), uçak üreticisine ya da ekipman üreticisine gidilmez. Buradaki kullanıcı, birimin başındaki pilot ya da teknisyen değil, uçağı işleten kuruluştur (operatör; tipik olarak bir havayolu). Değiştirilen şey veri, çalıştırılabilir kod ya da ikisi birden olabilir.

UMS'yi belirleyen, demek ki içeriğin türü değildir. "Parametre tablosu" ya da "konfigürasyon dosyası" olmak bir veriyi UMS yapmaz. Ölçüt şudur: kısıtlar içindeki değişikliğin emniyeti olumsuz etkileyemeyeceği sistem düzeyinde gösterilmiş midir? Gösterilmişse içerik kullanıcıya bırakılabilir; yanlış değeri emniyeti etkileyebilen veri ise aynı biçimde bir tablo olsa bile UMS olamaz.

Sahada karşılaşılan tipik UMS örnekleri şunlardır:

- havayolunun kendi düzenlediği, zorunlu olmayan kontrol listesi içerikleri,
- uçak durum izleme (aircraft condition monitoring) raporlarının tanımları: hangi parametrenin hangi koşulda ve hangi sıklıkta kaydedileceği,
- bakım ya da kabin ekranlarındaki dil, birim ve sayfa düzeni tercihleri.

UMS olmaya uygun olmayanlar da aynı açıklıkla sayılabilir: uçak ya da motor performans verisi, emniyet marjını belirleyen limitler, zorunlu uyarıların mantığı ve eşikleri. Bunlar değişecekse yol, değişikliğin geliştirici tarafından doğrulanıp onaya sunulmasıdır.

UMS en çok, yine "sahada bir şeyin değiştiği" üç kavramla karıştırılır:

Kavram	Kullanıcı ne yapar?	Yeni içerik gözden geçirme ve onaydan geçer mi?	Kitapta
Kullanıcı tarafından değiştirilebilir yazılım (UMS)	Kısıtlar içinde kodu ya da veriyi kendisi değiştirir	Hayır; onaylanan, kısıtlar ve korumadır	Bu bölüm
Parametre verisi ögesi (PDI)	Onaylı veri dosyasını kullanır; içeriği kendisi belirlemez	Evet; her PDI dosyası doğrulanır ve onaylı konfigürasyonun parçasıdır	22. Konfigürasyon Verisi
Seçenekle seçilebilir yazılım (option-selectable software)	Seçimi çoğu zaman uçak üreticisi ya da kurulumu yapan kuruluş yapar; kullanıcı yapsa bile yalnızca önceden onaylanmış seçeneklerden birini etkinleştirir, yeni davranış tanımlamaz	Seçenekler ve izin verilen birleşimleri baştan doğrulanıp onaylanmıştır	17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar
Sahada yüklenebilir yazılım (field-loadable software, FLS)	Onaylı imajı, donanımı sökmeden uçakta yükler	Evet; her sürüm kendi geliştirme, doğrulama ve onay sürecinden geçer	18. Sahada Yüklenebilir Yazılım

Seçenekle seçilebilir yazılımda seçilmeyen seçeneğin kodu birimde durmaya devam eder ve devre dışı bırakılmış kod (deactivated code) olarak ele alınır; seçimi kim yaparsa yapsın, yalnızca hazır seçenekler arasından seçilir ve DO-178C onaylanmamış bir konfigürasyonun yanlışlıkla seçilmesini önleyen bir önlem ister (§2.5.4). Sahada yüklenebilirlik ise içeriğin kimin sorumluluğunda olduğunu değil, birime nasıl ulaştığını anlatır: bir UMS tablosu da bir PDI dosyası da sahada yüklenebilir.

Bu kavramlar birbirini dışlamaz: standart, kullanımına göre bir PDI için UMS, seçenekle seçilebilir yazılım ve FLS rehberliğinin de ele alınmasını ister (§2.5.1) ve iki ekipman seçeneğinden birini seçen tek bir bellek bitini UMS örneği sayar (§5.2.3). Ayırt edici soru bu yüzden biçim değil, her zaman şudur: bu içeriği kim belirliyor ve belirlenen içerik kimin gözden geçirmesinden geçiyor?

Neden özel yönetim gerekir?

Parametre üzerinden davranış değiştirmek, kod değiştirmekten daha zararsız görünebilir; oysa bir değer değiştiğinde kod aynı kalsa da sistemin davranışı değişmiş olur. Onaylı yazılımda her değişikliğin arkasında gözden geçirme, doğrulama ve onay zinciri vardır. UMS'de bu zincir bilerek kaldırılır ve yerini tek bir iddia tutar: **kısıtlar içinde yapılan hiçbir değişiklik, doğru ya da yanlış yapılmış olsun, emniyeti, değiştirilemeyen bileşeni ve koruma mekanizmasını, operasyonel yetenekleri ya da mürettebat (flight crew) iş yükünü olumsuz etkileyemez.** DO-178C bu gösterilemiyorsa yazılımın kullanıcı tarafından değiştirilebilir diye sınıflandırılmayacağını açıkça

söyler (§2.5.2). Bu iddia yazılım ekibinin kendi başına kurabileceği bir iddia değildir. Kullanıcı değişikliğine izin veren ve onu sınırlayan mekanizmayı sistem gereksinimleri tanımlar; iddianın kendisi de sistem emniyet değerlendirme sürecinde savunulur.

Bunun pratik sonucu, aralık denetiminin anlamının değişmesidir. Onaya giren bir veri dosyasında aralık "geçerli" değerleri tanımlar; aralığın içinden doğru değerin seçildiğini göstermek doğrulamanın işidir. UMS'de değerin doğruluğunu kimse doğrulamayacağı için aralık "zararsız" değerleri tanımlamak zorundadır: izin verilen zarfın her noktası, kullanıcı onu yanlışlıkla seçmiş olsa bile kabul edilebilir olmalıdır.

Bir örnek ayrımı netleştirir. Havayolunun kendi eğilim izlemesi için kullandığı, zorunlu bakım programına girdi olmayan bir raporu tetikleyen sıcaklık eşiği kullanıcıya açılabilir: eşik yanlış girilirse rapor gereğinden sık ya da seyrek üretilir, uçuş emniyeti etkilenmez. Aynı sıcaklık için mürettebat uyarısını tetikleyen eşik kullanıcıya açılmaz. Bir limit tablosu ancak koruma mekanizmasının zorladığı aralıktaki her değerin emniyetli olduğu gösterilmişse UMS olabilir; "aralık denetimi var" demek bu gösterimin yerini tutmaz.

Aynı ilke gösterilen bilgi için de geçerlidir. Kullanıcının düzenlediği içerik mürettebata gösteriliyorsa onaylı bilgiyle karıştırılmayacak biçimde ayırt edilir; mürettebatın emniyetle ilgili kararını dayandırdığı bilgi ise kullanıcıya bırakılmaz.

Dayanak ve planlara yansımaları

DO-178C, UMS'yi başlıca iki yerde ele alır: sistem yönlerini anlatan 2. bölümünde (§2.5.2) sistem düzeyinde sağlanması gerekenleri sayar, yazılım tasarım sürecinde de (§5.2.3) değiştirilebilir bileşenle değiştirilemeyen bileşenin ayrılmasına ilişkin faaliyetleri verir. Standart, yazılımın operasyonel yönlerini ele almadığını, örneğin kullanıcı tarafından değiştirilebilir verinin sertifikasyon yönlerinin kapsamı dışında kaldığını baştan söyler (§1.2); §2.5.2 de sistem düzeyindeki faaliyetleri anlattığı için yazılım geliştiricisinin payına düşeni tek tek ayırmaz. FAA AC 20-115D ve EASA AMC 20-115D bu boşluğu 8. bölümlerindeki iki beklentiyle kapatır: geliştirici, sistem düzeyindeki hususların karşılanabilmesi için gereken bilgiyi sağlar; değiştirilebilir bileşen de kendisine atanan yazılım seviyesinden (software level) düşük olmayan bir seviyede geliştirilir. Bu dokümanlardan önce FAA tarafında konu Order 8110.49'un ayrı bir bölümünde ele alınıyordu; bugün başvurulacak rehber AC 20-115D ve AMC 20-115D'dir.

Projede UMS ilk olarak yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) görünür. UMS bir ek husus (additional consideration) olarak beyan edilir; değiştirilebilir bileşenin kapsamı, yazılım seviyesi, koruma yaklaşımı ve değişiklik için öngörülen araç ve prosedür burada otoriteyle mutabakata sunulur. DO-178C de UMS öngörülüyorsa ilgili süreç, araç, ortam ve veri öğelerinin planlarda belirtilmesini ister (§4.2). Bu sıra önemlidir: kısıtlar ilk sertifikasyonda belirlendiği için, planlama aşamasında açılmamış bir UMS kararını sonradan "bu tabloyu havayolu değiştirsin" diye eklemek yeni bir sertifikasyon tartışması başlatır.

Sonraki veriler aynı kararı izler. Koruma mekanizması ve değiştirilebilir bileşenle arayüz, tasarım tanımında yer alır. Yazılım konfigürasyon indeksi (Software Configuration Index, SCI) kullanıcı değişikliği için öngörülen prosedür, yöntem ve araçları tanımlar (§11.16). SCI'de tanımlanan yazılım ürünü temel çizgisi (baseline) ise UMS'nin yalnızca koruma ve sınır bileşenlerini içerir, değiştirilebilir içeriğin kendisini içermez (§7.2.2); kullanıcı değişikliğinin onaylı ürünün konfigürasyon kimliğini değiştirmemesini sağlayan da budur. Yazılım başarı özeti (Software Accomplishment Summary, SAS) PSAC'ta beyan edilen yaklaşımdan farkları açıklar; uygulamada korumanın nasıl sağlandığı ve kısıtların son hâli de burada özetlenir.

Sistemin değiştirilebilirliğe göre tasarlanması

UMS yaklaşımının işleyebilmesi için değiştirilebilirlik sonradan eklenen bir özellik değil, baştan verilmiş bir tasarım kararı olmalıdır. Mimari üç öğeden oluşur: değiştirilemeyen bileşen (non-modifiable component; bu bölümde kısaca çekirdek), kısıtlar içinde değiştirilebilir bileşen (modifiable component) ve ikisini ayıran koruma mekanizması (protection mechanism). Onay; çekirdeği, koruma mekanizmasını ve değişiklik kısıtlarını kapsar. Değiştirilebilir bileşenin kısıtlar içindeki güncel içeriği ise otoritenin tek tek gözden geçirmesine tabi değildir. Kullanıcı kısıtlar içinde kaldığı sürece sistemin onaylı durumunu bozmaz; koruma mekanizması da tam olarak bu sınırın aşılamayacağını garanti eder.

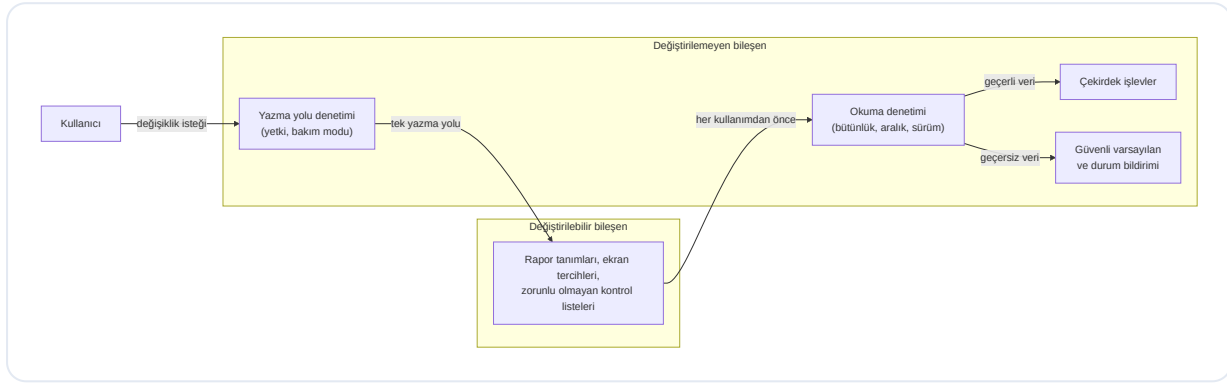
Tasarım dört soruya yazılı cevap verir: hangi alanlar değiştirilebilir, her alan için izin verilen zarf nedir, değişikliği kim ve hangi yolla yapabilir, değişiklikten sonra hangi kontrol yapılır? İlk iki soru bu başlığın, son ikisi bir sonraki başlığın konusudur.

Değiştirilebilir bileşenin ayrılması

Ayrımın somut olarak nasıl yapılacağı sistemin mimarisine bağlıdır; yaygın yöntemler şunlardır:

- **Fiziksel ayırım:** Değiştirilebilir içerik ayrı bir bellek aygıtında ya da ayrı bir donanım biriminde tutulur; çekirdeğin bulunduğu bellek, kullanıcı arayüzünden yazılamaz.
- **Yazılım bölümlemesi (software partitioning):** Değiştirilebilir bileşen, özellikle kod içeriyorsa, bölümlenmeli bir işletim sisteminde ayrı bir bölüme (partition) yerleştirilir. Bellek koruma birimi (memory protection unit, MPU) ya da bellek yönetim birimi (memory management unit, MMU) bu bileşenin çekirdeğin belleğine yazmasını, işletim sisteminin zaman çizelgesi de çekirdeğe ayrılan işlemci zamanını tüketmesini engeller; biri diğerinin yerini tutmaz (bkz. [21. Yazılım Bölümlemesi](#)).
- **Veri düzeyinde ayırım:** Değiştirilebilir içerik yalnızca veri (parametre tablosu, rapor tanımı) olarak tanımlanır; çekirdek bu veriyi her kullanımdan önce bütünlük ve aralık denetiminden geçirir.

Yöntem hangisi olursa olsun iki koşul değişmez. Koruma, **kullanıcının değiştiremeyeceği bir yerde** durur; kullanıcının erişebildiği bir denetim, denetim sayılmaz. DO-178C korumanın donanımla, yazılımla, değişikliği yapmakta kullanılan araçla ya da bunların birleşimiyle sağlanabileceğini kabul eder (§5.2.3); bu bölümdeki örnekler korumayı değiştiremeyen yazılım bileşenine yerleştirir. Ayrıca değişiklik için sağlanan yolun, değiştirilebilir bileşeni değiştirmenin **tek yolu** olduğu gösterilmelidir: bakım portundan, hata ayıklama arayüzünden ya da dosya sisteminden aynı belleğe ulaşan ikinci bir yol varsa yetki denetimi de kayıt da atlanabilir.



Koruma mekanizması iki noktada çalışır. Yazma yolunda, değişikliği yapanın yetkisini ve birimin değişikliğe uygun durumda olduğunu denetler. Okuma yolunda çekirdek, değiştirilebilir veriyi kullanmadan önce üç şeyi doğrular: verinin bozulmadığını, çekirdeğin beklediği biçim sürümünde olduğunu ve her alanın izin verilen zarfın içinde kaldığını. Bozulmayı yakalamanın tipik yolu döngüsel artıklık denetimidir (cyclic redundancy check, CRC). Basit bir örnek:

```

typedef struct {
    uint32_t sema_surumu;          /* çekirdeğin beklediği tablo biçimi */
    uint16_t rapor_esigi_degc;     /* bakım raporu tetikleme eşiği, °C */
    uint16_t ayrilmis;             /* örtük dolgu yerine açık alan; sıfır */
    uint32_t crc32;                /* önceki alanlar üzerinden CRC-32 */
} ums_tablo_t;

bool ums_tablo_gecerli(const ums_tablo_t *tablo)
{
    uint32_t hesaplanan;

    if (tablo == NULL) {
        return false;
    }
    hesaplanan = crc32_hesapla((const uint8_t *)tablo,
                               offsetof(ums_tablo_t, crc32));
    if (hesaplanan != tablo->crc32) {
        return false;                /* bozuk veri */
    }
    if (tablo->sema_surumu != UMS_SEMA_SURUMU) {
        return false;                /* sürüm uyumsuzluğu */
    }
    if (tablo->ayrilmis != 0U) {
        return false;                /* tanımsız alan dolu */
    }
    if ((tablo->rapor_esigi_degc < RAPOR_ESIGI_ENAZ_DEGC) ||
        (tablo->rapor_esigi_degc > RAPOR_ESIGI_ENCOK_DEGC)) {
        return false;                /* izin verilen zarfın dışı */
    }
    return true;
}

```

Yapıdaki ayrılmış alan süs değildir. Derleyicinin alanlar arasına kendiliğinden yerleştirdiği dolgu baytlarının (padding) yeri ve sayısı derleyiciye ve hedef mimariye göre değişir, içeriği de belirsizdir. Tabloyu üreten yer aracı ile hedef bilgisayar yapıyı bellekte farklı yerleştirirse CRC'nin kapsadığı bayt dizisi iki tarafta aynı olmaz ve geçerli tablo reddedilir. Bu yüzden yapı örtük dolgu bırakmayacak biçimde kurulur ya da CRC, alanların tek tek serileştirildiği bayt dizisi üzerinden hesaplanır.

Tablo geçersizse sistemin ne yapacağı da tasarımın parçasıdır: güvenli varsayılan değerlerle çalışmak, ilgili işlevi devre dışı bırakmak ya da durumu mürettebata veya bakım personeline bildirmek gibi seçenekler emniyet değerlendirmesine göre belirlenir. "Son geçerli değeri sessizce kullanmaya devam etmek" genellikle en riskli seçenektir, çünkü kullanıcı değişikliğinin alındığını sanır.

Koruma sınırının doğrulanması ve onay kapsamı

Yazılımla sağlanan koruma mekanizması, çekirdekle — daha kesin söylenirse değiştirilebilir bileşendeki hatalardan koruduğu işlevle — **aynı yazılım seviyesinde** tasarlanır ve doğrulanır (§2.5.2, §5.2.3). Doğrulama yalnızca "geçerli veri kabul ediliyor mu" sorusunu değil, asıl olarak olumsuz senaryoları hedefler: bozuk CRC, sınır dışı değerler, yanlış sürüm, kesilen yazma işlemi

sonrası yarım kalmış tablo, değiştirilebilir bileşenden çekirdek belleğe yazma girişimi, yetkisiz ya da yanlış durumda yapılan yazma isteği. Ayrıca kullanıcı değişikliğinin çekirdeğin **zamanlama ve bellek bütçesini** etkileyemediği gösterilmelidir; örneğin değiştirilebilir bir tablo, çekirdekte sınırsız bir döngüye ya da taşmaya yol açabiliyorsa ayırım kâğıt üzerinde kalmış demektir.

Değiştirilebilir bileşenin kendi yazılım seviyesi ayrı bir sorudur. Seviyeyi sistem emniyet değerlendirme süreci atar (bkz. [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#)): koruma mekanizması yerindeyken bu bileşenin katkıda bulunabileceği en ağır arıza durumuna (failure condition) bakılır. Koruma etkiliyse bu seviyenin çekirdeğinkinden düşük çıkması olağandır; ancak bu bir ön kabul değil, değerlendirmenin sonucudur. Geliştiricinin teslim ettiği ilk içerik, atanan seviyeden düşük olmayan bir seviyede geliştirilir.

Onay kapsamı da bu sınıra göre tanımlanır: sertifikasyon verisi, "kullanıcı şu alanları, şu aralıklar içinde, şu araç ve prosedürle değiştirebilir; bunun dışındaki her değişiklik tasarım değişikliğidir" ifadesini açıkça içermelidir. Kısıtlar içindeki değişiklikler yeniden onay gerektirmez.

Değiştirilemeyen yazılımı, korumasını ya da değiştirilebilir bileşenin sınırlarını etkileyen bir değişiklik (yeni bir alanın değiştirilebilir hâle getirilmesi, bir aralığın genişletilmesi) ise DO-178C'ye göre sıradan bir yazılım değişikliğidir ve önceden geliştirilmiş yazılımın değiştirilmesine ilişkin rehberliğe tabidir (§2.5.2, §12.1.1); emniyet değerlendirmesi ve koruma mekanizması yeniden ele alınır.

Değişikliklerin yönetimi ve bakımı

Kısıtlar içindeki değişiklikler otoritenin ve üreticinin gözden geçirmesinden geçmediği için, kullanıcı bir değişiklik yaptığı andan itibaren değiştirilen yazılımın bütün yönlerinin sorumluluğu **kullanıcıya** geçer: içeriğin konfigürasyon yönetimi, kalite güvencesi ve doğrulaması artık onun işidir (§2.5.2). Bu, geliştiricinin işinin bittiği anlamına gelmez: geliştirici, kullanıcının yazılımı uçağın emniyetini tehlikeye atmadan yönetebilmesi için gereken bilgiyi sağlamak zorundadır; prosedürler, araçlar ve kısıtlar bu amaçla tanımlanıp belgelenir. Sahada sık görülen bir sorun, koruma mekanizması sağlam tasarlanmış bir sistemin, değişiklik kayıtları tutulmadığı için "hangi uçakta hangi tablo yüklü" sorusuna cevap verilemez hâle gelmesidir.

Değişikliklerin kayıt altına alınması

Kullanıcı değişiklikleri, çekirdekten bağımsız ama onun kadar ciddi bir konfigürasyon yönetimi (configuration management) ister; tanımlama, değişiklik kaydı ve durum izleme ilkeleri geliştirme tarafındakilerle aynıdır (bkz. [10. Yazılım Konfigürasyon Yönetimi](#)). Asgari beklentiler:

- Her değiştirilebilir veri kümesinin (rapor tanımı, parametre tablosu) kendine ait bir **tanım ve sürüm kimliği** olmalıdır; çekirdek bu kimliği okuyup raporlayabilmelidir.
- Kim, ne zaman, hangi değeri, hangi gerekçeyle değiştirdi — bu bilgiler bir değişiklik kaydında tutulmalıdır. Sistem elektronik günlük tutabiliyorsa iyi; tutamıyorsa prosedürel kayıt (form, bakım kaydı) tanımlanmalıdır.

- Değişiklik yapma yetkisi sınırlandırılmalı (fiziksel anahtar, parola, bakım modu) ve yetkilendirme prosedürü yazılı olmalıdır.
- Değişiklik sonrası yapılacak kontrol (geri okuma, özet/CRC karşılaştırması, kısa bir işlev testi) prosedürün parçası olmalıdır; "yazdım, olmuştur" kabul edilmez.

Değişikliği üreten bir yer araçsa (örneğin parametre tablosunu derleyip ikili biçime çeviren bir yer destek aracı), bu aracın çıktısındaki bir hatanın koruma mekanizması tarafından yakalanıp yakalanamayacağına bakılır; yakalanamayan hata sınıfları varsa araç için araç kalifikasyonu (tool qualification) gündeme gelir (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)). Kısıtları yalnızca yer aracı zorluyor, birimdeki yazılım aynı denetimi yinelemiyorsa durum daha ağırdır: araç koruma mekanizmasının parçası olmuştur ve DO-178C bu durumda aracın araç kalifikasyonu kurallarına göre sınıflandırılıp kalifiye edilmesini ister (§5.2.3, §12.2).

Sürüm uyumunun izlenmesi

Değiştirilebilir veri ile çekirdek ayrı yaşam döngülerinde ilerlediği için zamanla **uyumsuzluk** riski doğar: çekirdek güncellenir, tablo biçimi değişir, ama sahadaki eski tablolar kalır. Bunu yönetmenin pratik yolları:

Mekanizma	Amaç
Tabloya gömülü şema/biçim sürümü	Çekirdeğin uyumsuz tabloyu reddetmesi
Uyumluluk matrisi (çekirdek sürümü × tablo sürümü)	Bakım personelinin doğru eşleşmeyi seçmesi
Açılıшта sürüm raporlama	Yüklü konfigürasyonun görünür ve denetlenebilir olması
Servis bülteni / bakım talimatı	Çekirdek güncellemesinde tabloların da ele alınması

Çekirdeğin her sürüm değişikliğinde, değiştirilebilir bileşenle olan arayüzün değişip değişmediği açıkça değerlendirilmeli ve sonuç kullanıcıya bildirilmelidir.

Kullanıcının sorumlulukları

Kullanıcı tarafında tipik sorumluluklar şunlardır:

- değişiklikleri yalnızca üreticinin tanımladığı prosedür ve araçlarla yapmak,
- değiştirdiği içeriği kendi süreciyle gözden geçirip doğrulamak ve bunun kaydını tutmak,
- tanımlı kısıtların dışına çıkan bir ihtiyaç doğduğunda bunu kendi başına zorlamak yerine üreticiye tasarım değişikliği olarak iletmek,
- filo genelinde hangi birimde hangi konfigürasyonun yüklü olduğunu izlemek,
- değişiklik kayıtlarını denetime hazır tutmak,
- şüpheli davranış raporlarında yüklü konfigürasyonu da olay bilgisine dahil etmek.

Bu sorumlulukların belirsiz kaldığı durumlarda değiştirilebilirlik, esneklik yerine izlenemeyen bir varyasyon kaynağına dönüşür.

Riskler ve karşı önlemler

Yukarıdaki mekanizmaların her biri belirli bir riski karşılar. Tablo, bir UMS tasarımını gözden geçirirken kontrol listesi olarak da kullanılabilir:

Risk	Nasıl ortaya çıkar?	Karşılayan mekanizma
Yetkisiz değişiklik	Bakım parolası elden ele dolaşır; aynı belleğe ulaşan ikinci bir yol açık kalır	Yazma yolunda yetki denetimi, tek değişiklik yolu, değişiklik kaydı
Zarf dışı ya da bozuk veri	Yazma yarıda kesilir, yer aracı hatalı çıktı üretir, dosya elle düzenlenir	Her kullanımdan önce bütünlük ve aralık denetimi; güvenli varsayılan ve durum bildirimi
Kayıtsız değişiklik	Değişiklik birimde yapılır, filo kaydına işlenmez	Sürüm kimliğinin birimden okunması, prosedürel kayıt, değişiklik sonrası kontrol
Sürümle uyuşmayan içerik	Çekirdek güncellenir, sahadaki eski tablolar kalır	Gömülü şema sürümü, uyumluluk matrisi, servis bülteni
Sınır sessizce genişlemesi	Emniyetle ilgili bir parametre "nasılsa veri" denerek değiştirilebilir alana taşınır	Sınır değişikliğinin tasarım değişikliği sayılması ve emniyet değerlendirmesinin yeniden ele alınması

Bu bölümden akılda kalması gerekenler

- UMS'yi belirleyen içeriğin türü değil, ilk sertifikasyonda belirlenen kısıtlar içinde otorite ve üretici gözden geçirmesi olmadan değiştirilmek üzere tasarlanmış olmasıdır. PDI, seçenekle seçilebilir yazılım ve FLS ile sınırı içeriğin biçimi değil, içeriği kimin belirlediği ve kimin gözden geçirdiği çizer; kavramlar birbirini dışlamaz.
- Kısıtlar içindeki her değişiklik, yanlış yapılmış olsa bile emniyeti etkilememelidir; aralık denetimi bu yüzden "geçerli" değil "zararsız" zarfı tanımlar ve bu iddia sistem düzeyinde savunulur.
- UMS kararı PSAC'ta beyan edilip otoriteyle baştan kararlaştırılır; SCI kullanıcı değişikliği için öngörülen prosedür ve araçları tanımlar. Ürün temel çizgisi UMS'nin yalnızca koruma ve sınır bileşenlerini içerir.
- Onay; değiştirilemeyen bileşeni, koruma mekanizmasını ve değişiklik kısıtlarını kapsar. Koruma donanımına, yazılıma ya da araçla sağlanabilir ama kullanıcının değiştiremeyeceği yerde durur: yazılımsa değiştirilemeyen yazılıma aynı seviyede doğrulanır, araçsa kalifiye edilir; değiştirilebilir bileşeni değiştirmenin tek yolu sağlanan yoldur.
- Kısıtlar içindeki değişiklikler yeniden onay gerektirmez; değiştirilemeyen yazılımı, korumayı ya da sınırı etkileyen değişiklik olağan yazılım değişikliği sürecinden geçer.

- Kullanıcı, yaptıęı deęiřiklikle birlikte deęiřtirilen içerięin konfigürasyon yönetimi, kalite güvencesi ve doęrulamasının sorumluluęunu üstlenir; geliřtirici bunun için gereken bilgiyi, prosedürü, aracı ve kısıtları saęlamakla yükümlüdür.

20. Gerçek Zamanlı İşletim Sistemleri

Gerçek zamanlı işletim sistemi (real-time operating system, RTOS), aviyonik yazılımda görev çizelgeleme, kesme işleme ve paylaşılan kaynakların korunması için öngörülebilir mekanizmalar sunar. Bu bölüm RTOS'u oluşturan katmanları, emniyet-kritik bir RTOS'ta aranan nitelikleri, RTOS seçimini ve hazır çekirdeğin sertifikasyondaki konumunu, RTOS'un mimariye etkilerini, çok çekirdekli işlemcileri ve doğrulamada projeye kalan işleri ele alır.

RTOS neden önemlidir?

Aviyonik bir işlevin doğru sonucu üretmesi yetmez; sonucu zamanında da üretmesi gerekir. Geç gelen bir kontrol komutu, yanlış hesaplanmış bir komut kadar tehlikeli olabilir. RTOS, işlemci zamanını görevler arasında paylaştıran, kesmeleri dağıtan ve ortak kaynaklara erişimi düzenleyen katman olduğu için bu zaman davranışının merkezinde durur.

Yine de RTOS zamanında çalışmayı tek başına güvence altına almaz. Sunduğu şey, davranışı önceden hesaplanabilen mekanizmalardır; görevlerin zaman sınırlarını tuttuğunu gösteren ise bu mekanizmaların üzerine kurulan tasarım ve analizdir. Bu yüzden işletim sisteminin seçimi ve kullanım biçimi bir altyapı ayrıntısı değil, emniyet mimarisinin parçasıdır: çekirdek, üzerinde çalışan her uygulamayla birlikte uçar ve onun hatası bütün uygulamaların hatasıdır.

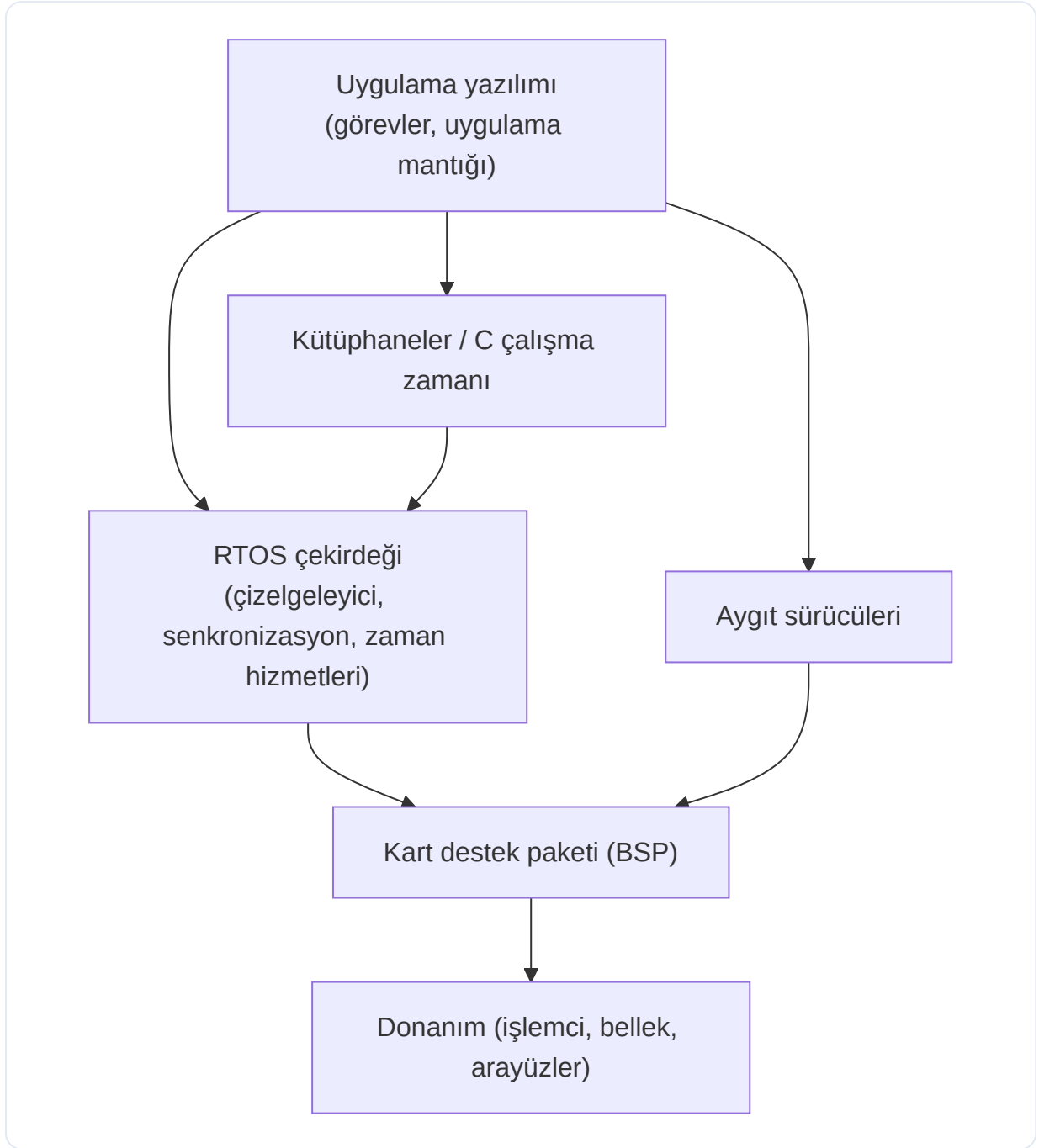
RTOS çekirdeği ve destek yazılımları

Günlük dilde "RTOS" tek bir üründen söz eder gibi kullanılır; oysa hedef karta yüklenen yazılım yığını birkaç ayrı katmandan oluşur ve her katmanın güvence açısından konumu farklıdır. Projede sınırları en baştan netleştirmek gerekir; çünkü sertifikasyon kanıtının kimin tarafından, hangi kapsamla üretileceği bu sınırlara göre belirlenir.

Tipik katmanlar şunlardır:

- **Çekirdek (kernel):** Çizelgeleyici (scheduler), görev (task) yönetimi, kesme dağıtımı, senkronizasyon nesneleri (semafor, kuyruk, muteks) ve zaman hizmetlerini sağlayan ana bileşen. RTOS tedarikçisinin ürünüdür ve çoğu zaman sertifikasyon paketiyle birlikte gelir.
- **Kart destek paketi (board support package, BSP):** Çekirdeği belirli bir işlemci kartına bağlayan başlatma kodu, saat ve kesme denetleyicisi yapılandırması, bellek haritası tanımları. Tedarikçi bir referans BSP verse bile, hedef karta uyarlanması genellikle proje ekibine düşer.
- **Aygıt sürücüler (device driver):** ARINC 429, MIL-STD-1553, Ethernet, ayrık giriş/çıkış gibi arayüzleri işleten kod. Kimi sürücüler RTOS ile gelir, kimileri projede yazılır; kaynağı ne olursa olsun uçuşta çalışan koddur.

- **Kütüphaneler ve çalışma zamanı desteği:** C çalışma zamanı, matematik kütüphaneleri, derleyicinin eklediği yardımcı rutinler. Görünmez oldukları için en sık ihmal edilen katmandır; bağlanan her fonksiyon güvence kapsamına girer.



Güvence açısından temel ilke basittir: **çalıştırılabilir nesne koduna bağlanan her şey, üzerinde çalışan uygulamanın yazılım seviyesiyle (software level) uyumlu biçimde doğrulanmalıdır.** Katman "hazır alındı" diye kapsam dışında kalmaz. Pratikte bu şu anlama gelir:

Katman	Tipik kaynak	Güvence kanıtının tipik sahibi
Çekirdek	RTOS tedarikçisi	Tedarikçinin sertifikasyon paketi + projenin entegrasyon doğrulaması
BSP	Tedarikçi şablonu + proje uyarlaması	Büyük ölçüde proje ekibi
Sürücüler	Karışık	Kaynağına göre; uyarlanan kısım proje ekibinde
Kütüphaneler	Derleyici/tedarikçi	Kullanılan alt küme için proje ekibi

Deneyimle sabittir: sertifikasyon paketli bir çekirdek satın almak, işin yalnızca bir bölümünü kapatır. BSP uyarlaması, projeye özgü sürücüler ve kullanılan kütüphane alt kümesi çoğu projede tedarikçi paketinin dışında kalır ve planlama aşamasında ayrı iş kalemleri olarak görünmelidir; aksi hâlde bu boşluk genellikle katılım aşaması (Stage of Involvement, SOI) denetimlerinde, yani en pahalı anda fark edilir.

Emniyet-kritik RTOS'un nitelikleri

Masaüstü veya genel amaçlı gömülü sistemlerde "iyi" sayılan bir işletim sistemi, emniyet-kritik bir aviyonik projede kullanılamayabilir. Fark, ortalama performansta değil, **en kötü durumun öngörülebilirliğindedir**. Emniyet-kritik bir RTOS'ta aranan başlıca nitelikler şunlardır:

Belirlenimci çizelgeleme (deterministic scheduling). Her çekirdek hizmetinin en kötü durum yürütme süresi (worst-case execution time, WCET) bilinmeli ve belgelenmiş olmalıdır. "Genellikle 5 mikrosaniyede döner" ifadesi yeterli değildir; zamanlama analizi ancak üst sınırlar üzerine kurulabilir. Öncelik tabanlı, kesintiye izin veren (preemptive) çizelgeleyiciler bu alanda yaygındır; bazı mimariler ise görevleri önceden tanımlı zaman pencerelerine yerleştiren zaman tetiklemeli (time-triggered) çizelgeleme kullanır.

Belirlenimci bellek yönetimi. Çalışma sırasında serbest dinamik bellek ayırma (dynamic memory allocation), parçalanma ve tükenme riski nedeniyle emniyet-kritik yazılımda genellikle yasaklanır veya başlatma aşamasıyla sınırlanır. İyi bir RTOS, sabit boyutlu blok havuzları gibi belirlenimci mekanizmalar sunar ve bellek koruma birimini (memory protection unit, MPU) kullanarak bir görevin başka bir görevin alanına yazmasını engelleyebilir. Bu koruma yalnızca bellek içindir; bir görevin işlemciyi gereğinden uzun tutmasını MPU engellemez.

Öngörülebilir kesme işleme. Kesme gecikmesinin (interrupt latency) üst sınırı bilinmeli, kesme servis rutinlerinin (interrupt service routine, ISR) çekirdek hizmetleriyle etkileşimi net kurallara bağlanmış olmalıdır. Kesmelerin kapalı tutulduğu en uzun süre, sistemin zamanlama bütçesine doğrudan girer.

Öncelik terslenmesine (priority inversion) karşı koruma. Düşük öncelikli bir görevin tuttuğu kaynağı bekleyen yüksek öncelikli görev, araya giren orta öncelikli görevler yüzünden süresiz gecikebilir. Olgun bir RTOS bunun için öncelik kalıtımı (priority inheritance) veya öncelik tavanı

(priority ceiling) protokollerini sunar.

Gürbüz bölümlleme (robust partitioning). Aynı işlemci üzerinde farklı yazılım seviyelerinden bileşenler barındırılacaksa, bir bölümdeki (partition) hatanın diğerlerinin ne belleğini ne de zaman bütçesini etkileyemeyeceği gösterilmelidir. Bu alandaki yaygın başvuru ARINC 653'tür: bölümlemeli bir işletim sistemi ile üzerinde çalışan uygulamalar arasındaki arayüzü (application/executive, APEX) ve bölümlerin alan ile zaman bakımından ayrılmasına dayanan çalışma modelini tanımlar. ARINC 653 tarzı işletim sistemleri, her bölüme ayrılmış bellek alanı ve garanti edilmiş zaman pencereleri sağlar; ayrıntı [21. Yazılım Bölümlemesi](#) bölümündedir.

Hata tespiti ve sağlık izleme (health monitoring). Zaman aşımı, izinsiz bellek erişimi, yığın taşması gibi olayların tespit edilmesi ve yapılandırılabilir bir tepkiye (bölümü yeniden başlatma, güvenli duruma geçme, kaydetme) bağlanabilmesi beklenir.

Sertififi edilebilirlik. Teknik nitelikler kadar önemlisi, RTOS'un DO-178C yaşam döngüsü verisiyle birlikte gelmesidir: gereksinimler, tasarım, kaynak kod, doğrulama sonuçları, yapısal kapsam analizi (structural coverage analysis) kanıtı ve izlenebilirlik (traceability). Kanıtı olmayan "hızlı ve küçük" bir çekirdek, sertifikasyon açısından sıfırdan geliştirilen koddan farksızdır.

Nitelik	Yokluğunda tipik belirti
Belirlenimci çizelgeleme	Yük altında ara sıra kaçırılan zaman sınırları
Belirlenimci bellek yönetimi	Uzun çalışmada parçalanma, öngörülemeyen ayırma hataları
Öngörülebilir kesme işleme	Kesme yükü arttığında geciken ya da kaçırılan olaylar
Öncelik terslenmesi koruması	Nadir, yeniden üretilmesi zor gecikme olayları
Gürbüz bölümlleme	Düşük seviyeli bir bileşen hatasının kritik işlevi etkilemesi
Sağlık izleme	Fark edilmeden süren hata; tepkisi tanımsız arıza
Sertifikasyon kanıtı	SOI denetimlerinde kapatılamayan bulgular

RTOS seçimi

RTOS seçimi teknik bir karşılaştırma gibi görünse de aslında bir **risk yönetimi** kararıdır: proje, çekirdeğin davranışına ilişkin kanıt üretme yükünün ne kadarını tedarikçiye devredebilecek, ne kadarını kendi üzerine alacaktır? Seçim üç ekseninde değerlendirilmelidir.

Teknik riskler. Önceki başlıkta sayılan nitelikler burada tek bir soruya iner: tedarikçi en kötü durum davranışını *hedef işlemci ailesi için* sayıyla ve kanıtıyla verebiliyor mu? Çekirdek hizmetlerinin WCET değerleri, kesmelerin kapalı kaldığı en uzun süre, kilit bekleme sürelerinin sınırı ve öncelik terslenmesine karşı uygulanan protokol belgelenmemişse, zamanlama analizinin girdileri eksik demektir ve bu boşluğu proje ölçümle ve analizle kendisi kapatmak zorunda kalır.

Sertifikasyon paketi. DO-178C açısından hazır bir RTOS, önceden geliştirilmiş yazılımdır (previously developed software, PDS). Katalogdan satılan ve müşteriye göre uyarlanmayan bir ürün olduğu ölçüde ticari hazır yazılım (commercial off-the-shelf, COTS) tanımına da girer; belirli bir uygulama için sözleşmeyle geliştirilen yazılımı standart COTS saymaz. Hazır olması ona bir ayrıcalık kazandırmaz: çekirdek uçuşa giden çalıştırılabilir nesne kodunun parçasıdır ve üzerinde çalışan en yüksek seviyeli uygulamanın yazılım seviyesine ait hedefler (objective) onun için de karşılanmış olmalıdır. Tedarikçinin sattığı sertifikasyon paketi bu kanıtın hazır üretilmiş kısmıdır; paketin verisi hedefleri karşılamaya yetmiyorsa eksik tamamlanır. Hazır yazılımın kullanılacağı ve uyumun nasıl sağlanacağı yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) açıklanır; çekirdeğin çözülmemiş problem raporlarının projeye etkisi de değerlendirilir. Önceden geliştirilmiş yazılımın genel değerlendirme yöntemi [24. Yazılım Yeniden Kullanımı](#) bölümündedir.

FAA bu tür bileşenler için AC 20-148 (2004) ile yeniden kullanılabilir yazılım bileşeni (reusable software component, RSC) yaklaşımını tanımlamıştır. Bileşenin geliştiricisi, bir sertifikasyon projesi kapsamında bileşenin yaşam döngüsü verisi için otoriteden kabul alır; kabul, hedeflerin bir kısmı için tam, bir kısmı için kısmi kredi anlamına gelebilir. Sonraki projeler bu krediyi, bileşen değişmediği ve belgelenmiş kısıtlar içinde kaldığı sürece yeniden kullanır. Yaklaşımın pratikteki özü, tedarikçinin entegratöre verdiği iki listedir: bileşenin hangi varsayımlar altında doğrulandığı (işlemci, derleyici ve seçenekleri, yapılandırma, kullanım kısıtları) ve hangi işlerin entegratöre kaldığı. Bu kabulün başka otoritelerce nasıl değerlendirileceği programa göre değişir ve otoriteyle erken konuşulmalıdır.

Paketin hedef yazılım seviyesini karşıladığı, hedef işlemciye ve kullanılan derleyici sürümüne uygulanabilir olduğu ve projenin yapılandırmasını (etkin özellik kümesini) kapsadığı ayrı ayrı doğrulanmalıdır. Paket dışında bırakılmış her özellik ya kapatılır ya da proje tarafından doğrulanır. "Kullanmıyoruz ama bağlı" durumu ise yapısal kapsam analizinde karşınıza çıkar: işlev bir gereksinime izlenebiliyor ve tasarım gereği çalıştırılmıyorsa devre dışı bırakılmış koddur (deactivated code). Bu sınıflandırma bedelsiz değildir: kod planlarda beyan edilir, etkin kod gibi standardın hedeflerine uygun geliştirilmiş olması ve istem dışı çalışmasının önleniğinin analiz ile testle gösterilmesi gerekir. Hiçbir gereksinime izlenemeyen işlev ise gereksiz koddur (extraneous code) ve beklenen çözüm kaldırılmasıdır. Ayrımın ölçütleri [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#) bölümünde işlenmiştir.

Tedarikçi değerlendirmesi. RTOS ilişkisi tek seferlik bir satın alma değil, programın ömrü boyunca süren bir bağımlılıktır. Tedarikçinin hata bildirim ve düzeltme süreci, sürüm politikası, daha önce hangi projelerde ve hangi otoritelerle sertifikasyon geçirdiği, araç kalifikasyonu (tool qualification) gereken yardımcı araçlar sunup sunmadığı ve uzun vadeli destek taahhüdü sorgulanmalıdır. Kaynak koda ve problem raporu geçmişine erişim koşulları sözleşmede netleşmelidir; bir tedarikçinin bilinen hata listesini paylaşma biçimi, çoğu zaman teknik broşüründen daha öğreticidir.

Eksen	Anahtar soru	Zayıflığın bedeli
Teknik	En kötü durum davranışı kanıtlanabilir mi?	Zamanlama analizinin çökmesi, geç keşfedilen tasarım değişikliği
Sertifikasyon paketi	Veri, bizim seviye/işlemci/yapılandırmamızı kapsıyor mu?	Boşlukları projenin doldurması; plan dışı doğrulama işi
Tedarikçi	Program ömrü boyunca destek sürecek mi?	Sürüm kilitlenmesi, hata düzeltmelerine erişememe

Pratik bir uyarı: değerlendirmeyi yalnızca veri sayfaları üzerinden yapmayın. Hedef karta yakın bir ortamda küçük bir deneme uygulaması koşturmak, broşürde görünmeyen davranışları erken ortaya çıkarır. Değerlendirmede sorulacak soruların tam listesi ve deneme uygulamasının kapsamı [Ek C](#) içindedir.

Mimari etkiler

RTOS seçildikten sonra asıl iş, onun üzerine analiz edilebilir bir yapı kurmaktır. Çekirdek pek çok hizmet sunar; emniyet-kritik bir tasarım bunların küçük ve öngörülebilir bir alt kümesiyle yetinir. Aşağıdaki kararlar tasarım tanımında yer alır ve tasarım gözden geçirmesinin konusudur ([7. Yazılım Tasarımı](#)).

Görev modeli ve öncelik atama

İşlevleri görevlere ayırmanın ölçütü kod düzeni değil, zaman davranışıdır: aynı periyotla çalışan ve aynı zaman sınırına bağlı işler tek görevde toplanır, farklı hızlar ayrı görevlere dağıtılır. Görev sayısı arttıkça bağlam değiştirme (context switch) yükü ve analiz edilecek etkileşim sayısı da artar; bu yüzden görev sayısı gerektiği kadar tutulur. Her görev için periyot, zaman sınırı, öncelik, yürütme süresi bütçesi ve yığın (stack) boyutu tasarım verisinde tablo hâlinde verilir; zamanlama analizi bu tabloyu girdi alır.

Sabit öncelikli çizelgelemede yaygın kural hız-monoton (rate-monotonic) atamadır: periyodu kısa olan görev daha yüksek öncelik alır. Sık yapılan hata, önceliği işlevin önemine göre vermektir; öncelik bir önem sıralaması değil, zaman sınırlarını tutturma aracıdır. Kritik ama yavaş bir görevin emniyeti yüksek öncelikte değil, zaman sınırının analize gösterilmesi ve gerekiyorsa bölümlenmeyle sağlanır. Periyodik olmayan olaylar, ardışık iki olay arasındaki en kısa süre sınırlandırılarak modele katılır; sınırı bilinmeyen bir olay kaynağı analiz edilemez. Görevler, kuyruklar ve muteksler başlatma sırasında bir kez yaratılır; çalışma sırasında görev yaratmak ya da öncelik değiştirmek analizin dayandığı modeli bozar.

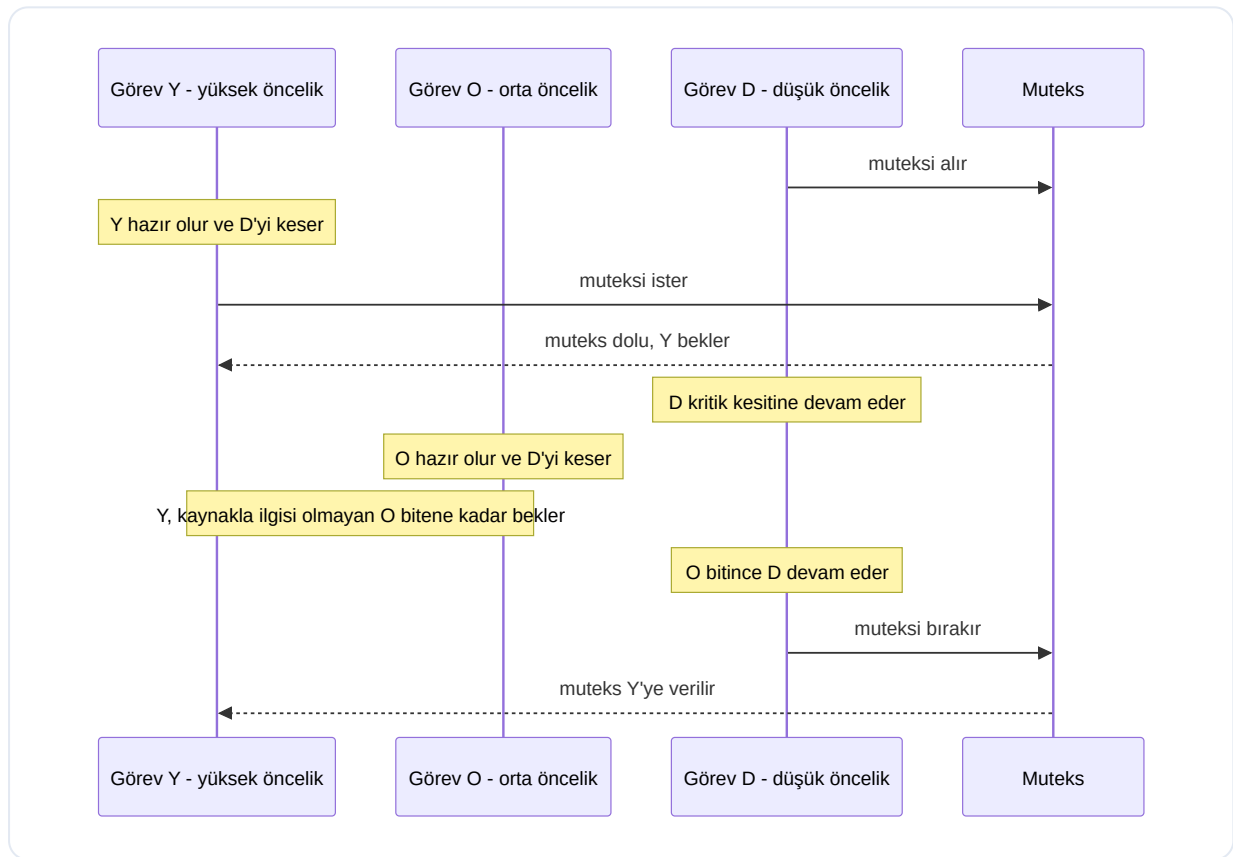
Kesme ile görev arasındaki iş bölümü

Kesme servis rutini yalnızca ertelenemeyecek işi yapar: veriyi alır, kesme kaynağını temizler ve işlemeyi yapacak görevi uyandırır. Uzun süren işlem görev düzeyine taşındığında hem çizelgeleyicinin öncelik kurallarına tabi olur hem de olağan yöntemlerle test edilebilir. Kesme bağlamında yalnızca beklemeyen (non-blocking) çekirdek çağrılarına izin verilir; hangi çağrıların kesme içinden yapılabileceği RTOS belgesinden alınıp tasarım standardına yazılır. Kesme ile görev arasında paylaşılan veri muteksle korunamaz, çünkü kesme rutini bekleyemez; bunun yerine kuyruk ya da kesmelerin çok kısa süreyle kapatıldığı bir kritik kesit (critical section) kullanılır ve bu süre zamanlama bütçesine eklenir.

Paylaşılan verinin korunması

Görevler arasında paylaşılan çok alanlı bir veri korunmazsa, okuyan görev yarısı eski yarısı yeni bir değer görebilir. Koruma için muteks kullanıldığında ise öncelik terslenmesi gündeme gelir.

Aşağıdaki akışta D düşük, O orta, Y yüksek öncelikli görevdir:



Y'nin gecikmesi D'nin kısa kritik kesitiyle değil, O'nun ne kadar çalışacağıyla belirlenir; yani üst sınırı yoktur. Öncelik kalıtımında D, Y beklemeye başladığı anda Y'nin önceliğine yükseltilir; O araya giremez ve Y'nin beklemesi D'nin kritik kesitinin uzunluğuyla sınırlanır. Bu sınır, zamanlama analizinde engellenme süresi (blocking time) olarak hesaba girer.

Aşağıdaki örnek, iki görevin paylaştığı hava verisinin öncelik kalıtlımlı bir muteksle okunmasını gösterir. `os_` önekli çağrılar örnek için uydurulmuş bir çekirdek arayüzüdür:

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>

#define HAVA_VERISI_KILIT_SURESI_MS (20)

typedef struct {
    int32_t irtifa_ft;
    int32_t dikey_hiz_fpm;
    uint32_t zaman_damgasi_ms;
} hava_verisi_t;

static hava_verisi_t g_hava_verisi;          /* iki görev paylaşır */
static os_muteks_t g_hava_verisi_muteksi;    /* öncelik kalıtımı etkin */

bool hava_verisi_oku(hava_verisi_t *hedef)
{
    bool basarili = false;

    if (hedef != NULL) {
        if (os_muteks_al(&g_hava_verisi_muteksi,
                        HAVA_VERISI_KILIT_SURESI_MS) == OS_TAMAM) {
            *hedef = g_hava_verisi; /* üç alan tek tutarlı kopya olarak alınır */
            if (os_muteks_birak(&g_hava_verisi_muteksi) == OS_TAMAM) {
                basarili = true;
            } else {
                saglik_izleme_bildir(HATA_MUTEKS_BIRAKMA);
            }
        } else {
            saglik_izleme_bildir(HATA_MUTEKS_ZAMAN_ASIMI);
        }
    }

    return basarili;
}
```

Örnekte üç tasarım kuralı görülür. Bekleme süresiz değildir; zaman aşımı, muteksın tutulabileceği en uzun kritik kesitten türetilmiş adlandırılmış bir sabittir. Çekirdek çağrılarının dönüş değeri her seferinde denetlenir ve başarısızlık sessizce yutulmaz, sağlık izlemeye bildirilir. Kritik kesit yalnızca kopyalamayı içerir; hesaplama, giriş/çıkış ya da başka bir çekirdek çağrısı kilidin dışında kalır. Birden çok muteksın iç içe alınması gerekiyorsa, kilitlenmeyi (deadlock) dışlamak için bütün görevlerin uyduğu tek bir alma sırası tanımlanır.

İzin verilen çekirdek hizmetleri ve soyutlama katmanı

Tasarım standardı, RTOS'un hangi hizmetlerinin hangi koşulla kullanılabileceğini açıkça listelemelidir: izin verilen senkronizasyon nesneleri, süresiz beklemenin yasak olması, başlatma sonrasında nesne yaratılmaması, kesme içinden yapılabilecek çağrılar. Bu liste iki işe yarar:

gözden geçirmede denetlenebilir bir ölçüt verir ve kullanılan hizmetlerin sertifikasyon paketinin kapsadığı yapılandırmanın içinde kalmasını sağlar.

Uygulamanın çekirdek arayüzünü doğrudan çağırması yerine ince bir soyutlama katmanı üzerinden çağırması, hem bu kısıtları tek noktada zorlar hem de RTOS değiştiğinde etkilenen kodu sınırlar; örneği, uyarlama katmanı (adaptation layer) adıyla [24. Yazılım Yeniden Kullanımı](#) bölümünde verilmiştir. Katmanın kendisi de uçuş yazılımıdır: gereksinimi, tasarımı ve testi vardır. Ayrıca zaman davranışını gizlememelidir; bir sarmalayıcı çağrının bekleyip beklemeyeceği ve en kötü durum süresi arayüzünde yazılı olmalıdır.

Çok çekirdekli işlemciler

Buraya kadar anlatılan zamanlama mantığı, bir anda tek bir yazılım parçasının çalıştığı tek çekirdekli işlemciyi varsayar. Çok çekirdekli işlemcide (multi-core processor) işlemci çekirdekleri önbellek, bellek denetleyicisi ve veri yolu gibi kaynakları paylaşır; bir çekirdekte çalışan yazılım, başka bir çekirdekteki yazılımın yürütme süresini bu kaynaklar üzerinden uzatabilir. Bu etki yollarına girişim kanalları (interference channels) denir. Sonuçları iki yönlüdür: tek çekirdekte ölçülmüş bir WCET değeri çok çekirdekli hedefe taşınamaz ve alan ile zaman bölümlenmesi, yazılımlar gerçekten eşzamanlı çalıştığı için yeniden gösterilmek zorundadır.

Otoritelerin beklentisi FAA AC 20-193 (2024) ve EASA AMC 20-193 (2022) belgelerindedir; bu belgeler daha önce başvurulmuş CAST-32A görüş belgesinin yerini almıştır. Beklentinin özü, girişim kanallarının belirlenmesi, her birinin ya kapatılması ya da etkisinin sınırlandırılması ve zamanlama kanıtının bu etki altında üretilmesidir. RTOS burada belirleyicidir: görevlerin ya da bölümlerin işlemci çekirdeklerine nasıl atandığı, kullanılmayan çekirdeklerin kapalı tutulup tutulmadığı ve paylaşılan kaynak kullanımını sınırlayan mekanizmalar çekirdeğin sunduklarına bağlıdır. Seçim sırasında tedarikçiden, sertifikasyon paketinin hangi çok çekirdekli yapılandırmayı kapsadığını ve hangi girişim analizinin entegratöre kaldığını açıkça söylemesi istenmelidir. Konunun risk başlıkları [Ek B](#) içinde toplanmıştır.

Doğrulama açısından bakış

Bir görev doğru çıktıyı yanlış zamanda üretiyorsa sistem davranışı hatalıdır; bu yüzden RTOS kullanan bir projede zaman davranışı işlev kadar doğrulanır. Zamanlama hataları nadir yük kombinasyonlarında ortaya çıktığı için testin yakalama olasılığı düşüktür; kanıtın omurgasını analiz oluşturur, test ve ölçüm onu destekler.

Çizelgelenebilirlik analizi (schedulability analysis). Görev tablosundaki periyot, zaman sınırı ve öncelik bilgisi; her görevin WCET değeri, en uzun kritik kesitten gelen engellenme süresi, çekirdek hizmetlerinin ve bağlam değiştirmenin süresi ve kesme yükü ile birleştirilerek her

görevin en kötü tepki süresi hesaplanır. Sonuç, zaman sınırıyla karşılaştırılır ve kalan pay raporlanır. Tedarikçinin verdiği çekirdek süreleri yalnızca paketin kapsadığı işlemci ve yapılandırma için geçerlidir.

WCET ve yığın analizi. İkisi de yalnız testle gösterilemez; analiz gerekir ve ölçümle desteklenir. DO-178C ikisini de kaynak kodun doğruluğu ve tutarlılığı incelenirken bakılan konular arasında sayar, derleyici ve bağlayıcı seçenekleri ile kimi donanım özelliklerinin en kötü durum süresine etkisinin değerlendirilmesini ister ve gözden geçirme, analiz ve testin birlikte kullanılabileceğini belirtir. Ulaşılan zamanlama ve bellek payları yazılım başarı özetinde (Software Accomplishment Summary, SAS) bildirilir. RTOS'a özgü nokta yığının görev başına ayrılmasıdır: her görevin yığını kendi en derin çağrı zincirine göre boyutlandırılır ve kesmelerin hangi yığını kullandığı (ayrı bir kesme yığını mı, kesilen görevin yığını mı) hesaba katılır. Yöntemler [9. Yazılım Doğrulama](#) bölümünde anlatılmıştır.

Hedefte ölçüm. Tepki süreleri, titreşim (jitter), kesmelerin kapalı kaldığı en uzun süre ve yığın kullanımının ulaştığı en yüksek düzey hedef donanımda ölçülür. Ölçüm analizin yerine geçmez; analizin varsayımlarını sınar ve analizle ölçüm arasındaki açıklanamayan fark bir bulgu olarak ele alınır.

Gürbüzlük (robustness) testleri. Uygulamanın çekirdek hata dönüşlerine tepkisi gereksinim tabanlı testlerle gösterilir: zaman aşımına uğrayan muteks, dolu kuyruk, süresini aşan görev. Sağlık izleme için de aynısı geçerlidir; yığın taşması, izinsiz bellek erişimi ve zaman sınırı aşımı bilinçli olarak tetiklenir ve yapılandırılmış tepkinin gerçekten verildiği gözlenir. Tepkisi hiç denenmemiş bir sağlık izleme tablosu, kâğıt üzerindeki bir emniyet mekanizmasıdır.

Kanıtın hangi kısmının tedarikçi paketinden geldiği, hangisinin projede üretildiği doğrulama planında açıkça ayrılmalıdır:

Konu	Tedarikçi paketinden gelen	Projenin ürettiği
Çekirdek hizmetlerinin işlevi	Çekirdek gereksinimlerine dayalı test ve yapısal kapsam sonuçları	Paketin geçerlilik koşullarının (işlemci, derleyici, yapılandırma) sağlandığının gösterimi; entegrasyon testleri
Zamanlama	Çekirdek hizmetlerinin ve kesme kapalı süresinin üst sınırları	Görev WCET değerleri, çizelgelenebilirlik analizi, hedefte ölçüm
Bellek	Çekirdeğin kendi bellek ve yığın ihtiyacı	Görev yığınlarının analizi, bellek haritası, MPU yapılandırmasının doğrulanması
Hata davranışı	Çekirdek hata dönüşlerinin tanımı ve testi	Uygulamanın bu dönüşlere tepkisinin ve sağlık izleme tepkilerinin testi
BSP, sürücüler, kütüphane alt kümesi	Varsa referans uygulama	Gereksinimden yapısal kapsama kadar bütün yaşam döngüsü verisi

Kullanım sırasında izlenecek risk alanları ve her biri için sorulacak sorular [Ek B](#) içindedir.

Bu bölümden akılda kalması gerekenler

- RTOS zamanında çalışmayı güvence altına almaz; öngörülebilir mekanizmalar sunar. Zaman sınırlarının tutulduğunu gösteren, bu mekanizmalar üzerine kurulan tasarım ve analizdir.
- Çekirdek, BSP, sürücüler ve kütüphaneler ayrı katmanlardır; bağlanan her katman güvence kapsamına girer ve kanıt sahipliği en baştan netleşmelidir.
- Emniyet-kritik bir RTOS'un ölçüsü ortalama performans değil, en kötü durumun öngörülebilirliği: belirlenimci çizelgeleme, belirlenimci bellek yönetimi, sınırlı kesme gecikmesi, öncelik terslenmesine karşı koruma ve gürbüz bölümlendirme.
- RTOS seçimi bir risk yönetimi kararıdır: teknik nitelik, sertifikasyon paketinin kapsamı ve tedarikçinin uzun vadeli desteği birlikte değerlendirilir.
- Hazır RTOS önceden geliştirilmiş yazılımdır; sertifikasyon paketi ancak projenin seviyesini, işlemcisini, derleyicisini ve yapılandırmasını kapsadığı ölçüde kredi sağlar, gerisi entegratörün işidir.
- Mimari, çekirdek hizmetlerinin kısıtlı bir alt kümesine dayanır: başlatmada yaratılan görevler, işlevin önemine göre değil periyoda göre atanan öncelikler, kısa kesme rutinleri, zaman aşım ve öncelik kalıtsal kilitler. Bu kurallar tasarım standardına yazılır.
- Zaman davranışının kanıtı analize dayanır, ölçüm onu destekler: çizelgelenebilirlik, WCET ve görev başına yığın analizi projenin işidir. Çok çekirdekli hedefte bu analizler girişim kanalları belirlenip sınırlandırılmadan savunulamaz.

21. Yazılım Bölümlemesi

Yazılım bölümlemesi (software partitioning), aynı donanımı paylaşan yazılım bileşenlerini birbirinden yalıtarak birindeki hatanın diğerlerine yayılmasını önleyen mimari tekniktir. Bu bölüm alan ve zaman bölümlemesini, DO-178C'nin bölümlemeden beklediklerini, yalıtımın etrafından dolaşan donanım ve yazılım yollarını ve bölümleme iddiasının nasıl doğrulanacağını ele alır.

Amaç yalnızca bir sınır çizmek değil, o sınırın her koşulda korunduğunu göstermektir; bu yüzden bölümleme kararları, doğrulama yöntemiyle birlikte düşünülür. Yalıtılan her bileşene bölüm (partition) denir. Bu sayfada "bölüm" sözcüğü bu anlamda kullanılır; kitabın bölümleri numarası ve başlığıyla anılır.

Bölümleme neden gerekir?

Aynı işlemci üzerinde farklı yazılım seviyelerinden (software level) işlevler birlikte çalışabilir. Sayfa boyunca şu örneği kullanacağız: tek bir işlemci kartında uçuş kontrol işlevi (Seviye A), onun durum verisini kaydedip bozulma eğilimini bildiren bir durum izleme işlevi (Seviye C) ve yalnızca yerde kullanılan bir bakım ekranı (Seviye D) çalışmaktadır. Bakım yazılımındaki bir hata — bozuk bir işaretçi, sonsuz bir döngü, veri yolunu dolduran bir ileti seli — kontrol döngüsünü etkilememelidir. Bu gösterilemezse üç bileşen tek bir bileşen gibi ele alınır ve hepsi aralarındaki en yüksek seviyeyle, yani Seviye A olarak geliştirilmek zorundadır; bölümlemenin seviye atamasına etkisi [3. Sistem Emniyet Değerlendirmesi Bağlamında Yazılım](#) bölümünde anlatılmıştır.

Örnekteki izleme işlevi, DO-178C'nin ayrı bir mimari yaklaşım olarak andığı emniyet izleme (safety monitoring) değildir. İzlediği işlevin arıza durumuna karşı koruma sağlayan bir izleyici, o işlevin en ağır arıza durumunun gerektirdiği yazılım seviyesini alır ve izlediği işlevi bozan arızayla birlikte devre dışı kalmayacak kadar ondan bağımsız olmak zorundadır. Uçuş kontrolü koruyan bir izleyici bu yüzden Seviye C olamazdı; aynı kartı paylaşması da ayrıca savunulmak zorunda kalırdı.

Aynı sonuca fiziksel ayrımla, yani her işlevi kendi işlemcisine ve donanımına yerleştirerek de ulaşılabilir; o zaman yalıtımı donanım sınırı sağlar. DO-178C bu yolu da bölümleme sayar ve aşağıdaki beklentileri yöntem hangisi olursa olsun arar. Bu bölümün konusu, ayrımın paylaşılan donanım üzerinde işlemci mekanizmaları ve yazılımla kurulmasıdır. Kazancı donanımın paylaşılması, bedeli ise yalıtımın kanıtlanması gereken bir iddiaya dönüşmesidir.

Bölümlerden biri nasıl davranırsa davransın — tasarım hatası içersin ya da kendisine özgü donanım arızalansın — diğerlerinin etkilenmediği gösterilebilen bölümlemeye gürbüz bölümleme (robust partitioning) denir. Terim DO-178C'de geçmez; aşağıda anılan ARINC 653 ve DO-297 çevresinde yerleşmiştir. Sertifikasyonda dayanak yapılabilen budur: "normal çalışmada birbirlerine karışmıyorlar" gözlemi bölümleme kanıtı sayılmaz.

Bölümleme yalnızca seviyeleri ayırmak için kullanılmaz; değiştirilemeyen bileşeni kullanıcının değiştirebildiği yazılımdan korumanın da yaygın yollarından biridir (bkz. [19. Kullanıcı Tarafından Değiştirilebilir Yazılım](#)).

DO-178C bölümlemeden ne bekler?

DO-178C bölümlemeyi, sistemle ilişkiyi anlatırken andığı üç mimari yaklaşımdan biri olarak ele alır; diğer ikisi çok sürümlü benzemez yazılım (multiple-version dissimilar software) ve emniyet izlemedir. Belirli bir mekanizma dayatmaz, bu yaklaşımları tercih edilen ya da zorunlu çözüm olarak da sunmaz; neyin gösterilmesi gerektiğini söyler. Bu beklentileri, neyin korunduğuna göre dört soruda toplamak mümkündür:

Soru	Gösterilmesi gereken	Tipik kanıt
Bir bölüm komşusunun belleğine ya da giriş/çıkışına dokunabilir mi?	Her bölümün kodu, veri alanları ve giriş/çıkışı komşularının bozamayacağı biçimde korunur	Bellek koruma ayarlarının gözden geçirilmesi; izinsiz erişimin reddedildiğini gösteren testler
Bir bölüm komşusunun işlemci zamanını yiyebilir mi?	Ortak işlemci kaynakları bir bölümce ancak çizelgede ona ayrılan yürütme süresi içinde tüketilir	Çizelge tablosunun analizi; süresini aşan bölümün kesildiğini gösteren testler
Bir bölüme özgü donanım bozulursa etki yayılır mı?	Yalnızca bir bölümün kullandığı donanım arızalandığında olumsuz etki o bölümle sınırlı kalır	Emniyet analizi; arıza enjeksiyonu testleri
Korumayı sağlayan mekanizmaya ne kadar güvenilebilir?	Koruma yazılımla sağlanıyorsa o yazılım en az korunan bölümlerin en yüksek yazılım seviyesindedir; donanımla sağlanıyorsa donanımın emniyete etkisi sistem emniyet değerlendirme sürecinde incelenir	Çekirdeğin ve ortak sürücülerin yaşam döngüsü verisi; sistem tarafındaki donanım analizi

Son satır pratikte en çok maliyet doğuran beklentidir. Örneğimizde bölümleme çekirdeği, bellek koruma tablolarını kuran başlatma kodu, pencere geçişini yapan çizelgeleyici (scheduler) ve bölümlerin ortak kullandığı sürücüler, korudukları bölümlerden biri Seviye A olduğu için Seviye A olarak geliştirilir. Bölümleme, düşük seviyeli uygulamaların maliyetini düşürür; platformun maliyetini düşürmez.

Üç noktanın daha bilinmesi gerekir. Birincisi, seviyeleri yazılım ekibi seçmez: bölümleme iddiası sistem emniyet değerlendirme sürecine girdidir ve bileşenlerin seviyesi oradan atanır; ayrı ayrı seviye alabilenler yalnızca bölümlenmiş bileşenlerdir. İkincisi, bölümleme yaklaşımı yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) yazılıma genel bakış içinde bildirilir (bkz. [5. Yazılım Planlama](#)); otorite bu iddiayı ilk denetimde sorgular ([SW SOI-1](#)). İzi başka verilerde de aranır: bütünlüğün hangi yöntemle doğrulanacağı yazılım doğrulama planında (Software Verification Plan, SVP), yazılıma tahsis edilen bölümleme gereksinimleri ile bölümlerin etkileşimi ve seviyeleri gereksinim verisinde, bölümleme yöntemi ve ihlalin nasıl önlendiği tasarım tanımında yer alır; yazılım başarı özeti (Software Accomplishment Summary,

SAS) kullanılan yaklaşımı plandan farklarıyla yeniden anlatır. Bölümler arasında hangi etkileşime ne ölçüde izin verildiği ve korumanın yalnız donanımla mı, donanım ile yazılımın birlikte çalışmasıyla mı sağlandığı bu verilerde açıkça yanıtlanmış olmalıdır. Üçüncüsü, bölümleme bütünlüğünün doğrulanması tasarım sürecinin çıktılarına ilişkin doğrulama hedefleri (objective) arasındadır ve Seviye A'dan Seviye D'ye dört seviyede de aranır; Seviye A'da bağımsızlıkla karşılanır. Bu gruptaki on üç hedeften Seviye D'de geçerli olan yalnızca budur. Bölümleme bir tasarım tercihi olarak kalamaz; kanıtı üretilmesi gereken bir hedeftir.

Alan ve zaman bölümlemesi

Uygulamada bölümleme iki temel eksenle düşünülür: alan bölümlemesi (spatial partitioning) ve zaman bölümlemesi (temporal partitioning). İlki "bir bölüm, diğerinin belleğini bozamaz" güvencesini, ikincisi "bir bölüm, diğerinin işlemci zamanını çalamaz" güvencesini hedefler. Giriş/çıkış aygıtları ve diğer paylaşılan kaynaklar ayrı bir eksen oluşturmaz; her biri hem alan (kim erişebilir) hem zaman (ne zaman ve ne kadar süreyle) sorusuyla incelenir. Gerçek sistemlerde iki eksen birlikte uygulanır; yalnızca birinin sağlanması, paylaşılan platformda yeterli yalıtım vermez. Bellek koruması kusursuz çalışan bir sistemde bile sonsuz döngüye giren bölüm, zaman bölümlemesi yoksa diğerlerini işlemcisiz bırakır.

Alan bölümlemesi

Alan bölümlemesi, paylaşılan bellek üzerinde her bölüme kendi adres alanını ayırır ve bu sınırın donanım tarafından zorlanmasını sağlar. Tipik mekanizmalar şunlardır:

- **Bellek yönetim birimi (memory management unit, MMU) veya bellek koruma birimi (memory protection unit, MPU):** Her bölüm için erişilebilir adres aralıklarını ve erişim haklarını (oku/yaz/çalıştır) tanımlar. Sınır dışı bir erişim, donanım istisnası (exception) üretir ve çekirdek tarafından yakalanır.
- **Ayrıcalık seviyeleri:** Uygulama bölümleri kullanıcı kipinde (user mode) çalışır; MMU/MPU yapılandırmasını yalnızca çekirdek kipindeki (kernel mode) bölümleme çekirdeği değiştirebilir. Aksi hâlde bir bölüm kendi sınırlarını genişletebilirdi.
- **Yazma korumalı paylaşım:** İki bölümün aynı veriyi okuması gerekiyorsa, alan salt okunur olarak her ikisine eşlenir; yazma hakkı yalnızca üretici bölümde kalır.

Örneğimizdeki üç bölüm için erişim hakları aşağıdaki gibi kurulabilir. Tablonun asıl değeri, boş bırakılmış hücrelerinin olmamasıdır: her bölge için her bölümün hakkı açıkça yazılmıştır.

Bellek bölgesi	Uçuş kontrol	İzleme	Bakım
Uçuş kontrol kodu ve sabit verisi	oku, çalıştır	erişim yok	erişim yok
Uçuş kontrol verisi ve yığını	oku, yaz	erişim yok	erişim yok
İzleme kodu ve sabit verisi	erişim yok	oku, çalıştır	erişim yok

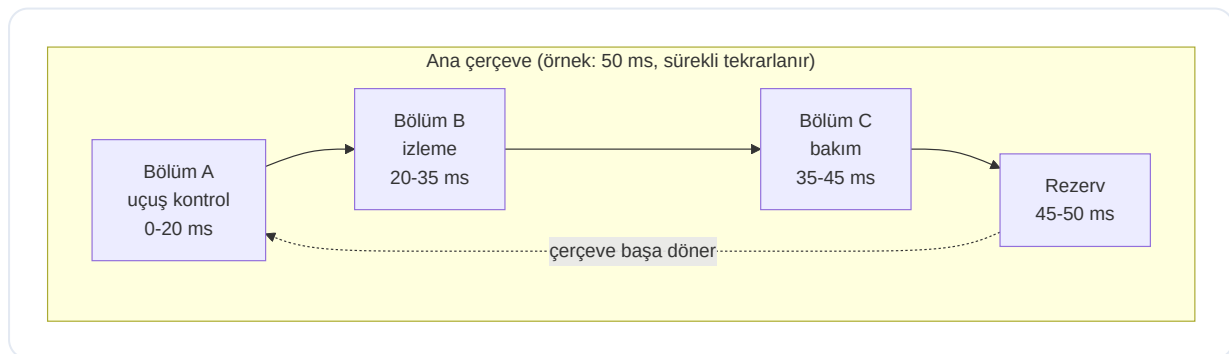
Bellek bölgesi	Uçuş kontrol	İzleme	Bakım
İzleme verisi ve yığını	erişim yok	oku, yaz	erişim yok
Bakım kodu ve sabit verisi	erişim yok	erişim yok	oku, çalıştır
Bakım verisi ve yığını	erişim yok	erişim yok	oku, yaz
Durum tamponu (uçuş kontrolden izlemeye)	oku, yaz	oku	erişim yok
Eyleyici arayüzü yazmaçları	oku, yaz	erişim yok	erişim yok
Bakım veri yolu yazmaçları	erişim yok	erişim yok	oku, yaz
Çekirdek kodu, verisi ve koruma tabloları	erişim yok	erişim yok	erişim yok

İki ayrıntı böyle bir tabloda sık atlanır. Hiçbir bölge aynı anda hem yazılabilir hem çalıştırılabilir olmamalıdır; aksi hâlde bozulan veri kod olarak yürütülebilir. Her yığının bitimine de erişimsiz bir koruma bölgesi konmalıdır ki taşma, komşu veriyi sessizce bozmak yerine istisna üretsinsin.

Alan bölümlemesi yalnızca RAM ile sınırlı değildir; kalıcı bellek (non-volatile memory) bölgeleri, bellek eşlemlili (memory-mapped) çevre birimi yazmaçları ve yığın/öbek (stack/heap) taşmaları da aynı analiz kapsamına girer. Bir bölümün kendi yığınını taşıyıp komşu bölgeye yazması, klasik bir alan bölümlemesi ihlalidir.

Zaman bölümlemesi

Zaman bölümlemesi, paylaşılan işlemciyi bölümlere öngörülebilir dilimler hâlinde tahsis eder. En yaygın yaklaşım, sabit çevrimli çizelgelemedir (scheduling): ana çerçeve (major frame) sabit uzunluktadır ve içindeki zaman pencereleri bölümlere statik olarak atanır. Çizelge entegrasyon sırasında tanımlanan bir konfigürasyon verisidir; bölümler çalışma zamanında onu değiştiremez ve yeni çizelge üretemez. Bazı platformlar önceden tanımlanmış birkaç çizelge arasında yetkili bir bölümün isteğiyle geçişe izin verir; bu durumda her çizelge ve her geçiş ayrıca doğrulanır.



Zaman bölümlemesinin sağlaması gereken güvenceler şunlardır:

- Bir bölüm kendi penceresini aşarsa (örneğin sonsuz döngüye girerse) çekirdek, donanım zamanlayıcısıyla o bölümü keser ve sıradaki pencereye geçer; taşma diğer bölümlerin dilimini kısaltamaz.

- Pencere geçişindeki bağlam değiştirme (context switch) süresi çizelgede hesaba katılır; aksi hâlde kâğıt üzerindeki dilimler pratikte erir.
- Kritik bölümün penceresi, o bölümün en kötü durum yürütme süresi (worst-case execution time, WCET) analiziyle boyutlandırılır.
- Bir bölümün penceresinin başlama anı, önceki bölümlerin ne yaptığına bağlı olarak kaymaz; erken biten bölümün artan süresi sıradakine devredilmez.

Çizelge tablosu uygulama kodunda tek satır değişmeden bütün sistemin zamanlamasını değiştirebildiği için kod kadar ciddiye alınır; bu tabloların parametre verisi ögesi olarak nasıl yönetildiği [22. Konfigürasyon Verisi](#) bölümünde anlatılmıştır.

Paylaşılan giriş/çıkışın yönetimi

Bellek ve işlemci bölünse bile giriş/çıkış (I/O) kaynakları — veri yolları, UART'lar, ayırık hatlar — çoğu zaman ortaktır ve bölümleme analizinin en çok ihmal edilen parçasıdır. Yaygın çözümler:

Yaklaşım	Açıklama	Dikkat edilecek nokta
I/O sahipliği	Her çevre birimi tek bir bölüme atanır	Diğer bölümlerin yazmaçlara eşlenmiş adreslere erişimi MMU ile kapatılmalı
Sunucu bölüm	Ortak I/O'yu tek bir sürücü bölümü yönetir, diğerleri iletiler	Sunucu bölümün yazılım seviyesi, hizmet verdiği en kritik bölüme göre belirlenir
Zaman dilimli erişim	Veri yoluna erişim, bölüm pencereleriyle hizalanır	Pencere dışına sarkan doğrudan bellek erişimi (DMA) aktarımları ayrıca analiz edilmeli

Deneyimde en sık görülen hata, çevre birimi yazmaçlarının "kimse oraya yazmaz nasılsa" varsayımıyla korumasız bırakılmasıdır. Bölümleme kanıtı varsayımı değil, donanımın zorladığı sınıra dayanmalıdır.

ARINC 653 ve IMA bağlamı

Bölümlemeyle çalışan bir mühendisin en sık karşılaşacağı çerçeve ARINC 653'tür: bölümlemeli bir işletim sistemi ile üzerinde çalışan uygulamalar arasındaki arayüzü (application/executive, APEX) ve bu arayüzün arkasındaki çalışma modelini tanımlar. Modelin bu bölüm açısından önemli dört ögesi vardır.

İki seviyeli çizelgeleme. Üst seviyede çekirdek, bölümleri yukarıda anlatılan statik ve döngüsel çizelgeyle çalıştırır. Alt seviyede her bölüm kendi penceresi içinde kendi süreçlerini (process; başka işletim sistemlerindeki görevin karşılığı) öncelik tabanlı olarak çizelgeler. Bir bölümün içindeki öncelik hatası ya da kaçan süre sınırı o bölümün sorunudur; pencere sınırını geçemez.

Portlar. Bölümler birbirleriyle yalnızca konfigürasyonda tanımlı kanallara bağlı portlar üzerinden konuşur. Örnekleme portu (sampling port) yalnızca son iletiyi tutar, yeni ileti eskisinin üzerine yazar ve okuyana iletinin hâlâ taze olup olmadığı bildirilir; periyodik durum verisi için uygundur. Kuyruk portu (queuing port) iletileri sabit kapasiteli bir kuyrukta sırayla tutar; komut ve olay gibi hiçbir atlanmaması gereken iletiler için kullanılır.

Sağlık izleme (health monitoring). Bellek ihlali, kaçan süre sınırı ya da uygulamanın bildirdiği hata gibi olaylar süreç, bölüm ya da modül düzeyinde sınıflandırılır ve konfigürasyon tablosunda önceden tanımlanmış tepkiye bağlanır: kaydetme, sürecin durdurulması, bölümün yeniden başlatılması ya da kapatılması gibi. Bölümleme açısından önemli olan, ihlalin algılanması kadar tepkinin de ihlal eden bölümle sınırlı kalmasıdır.

Konfigürasyon tabloları. Bellek bölgeleri, zaman pencereleri, portlar, kanallar ve sağlık izleme tepkileri uygulama kodunda değil, entegratörün hazırladığı tablolarda durur. Bölümleme mekanizması çekirdekte, bölümleme kararı ise bu tablolardadır.

ARINC 653 çoğunlukla tümleşik modüler aviyonik (integrated modular avionics, IMA) platformlarında kullanılır. IMA'da işler başlıca üç role dağılır: platformu ve çekirdeği sağlayan, platformda barınan uygulamayı geliştiren ve hepsini tek bir konfigürasyonda birleştiren entegratör. Bu rollerin sorumluluklarını ve platformun, tek tek uygulamaların ve bütünleşik sistemin kademeli kabulünü DO-297 / ED-124 ele alır; otorite tarafındaki karşılıkları FAA AC 20-170 ve EASA AMC 20-170'tir.

Bu iş bölümünün bölümleme kanıtına doğrudan etkisi vardır. Platform sağlayıcının gürbüz bölümleme kanıtı, belirli kullanım koşulları altında geçerlidir: izin verilen konfigürasyon aralıkları, kullanılabilecek hizmetler, uygulamaların uyması gereken kısıtlar. Entegratör, kendi konfigürasyonunun bu koşulların içinde kaldığını göstermek zorundadır; "sertifikalı çekirdek kullanıyoruz" cümlesi tek başına bölümleme kanıtı olmaz. İşletim sistemi seçimi ve hazır sertifikasyon paketinin kapsamı [20. Gerçek Zamanlı İşletim Sistemleri](#) bölümünün konusudur.

Bölümlemeyi zorlaştıran konular

MMU ve zaman çizelgesi kurulduğunda bölümleme bitmiş görünür; oysa asıl zorluk, bu iki mekanizmanın etrafından dolaşabilen donanım ve yazılım yollarındadır. Bölümleme analizi (partitioning analysis) tam da bu yolları tek tek bulup kapatıldığını göstermek zorundadır.

Doğrudan bellek erişimi (DMA)

Doğrudan bellek erişimi (direct memory access, DMA) denetleyicileri, işlemciyi atlayarak belleğe yazar; dolayısıyla MMU'nun bölüm sınırları çoğu donanımda DMA aktarımlarına uygulanmaz. Yanlış programlanmış tek bir DMA tanımlayıcısı, kritik bölümün belleğinin üzerine yazabilir. Alınabilecek önlemler:

- DMA'yı yalnızca çekirdeğin veya tek bir güvenilir sürücü bölümünün programlayabilmesi; uygulama bölümlerinin DMA yazmaçlarına erişiminin MMU ile kapatılması,
- varsa I/O MMU (IOMMU) benzeri bir donanımla DMA hedef adreslerinin de sınırlandırılması,
- DMA hedef aralıklarının, bölüm bellek haritasına karşı çekirdek tarafından çalışma zamanında doğrulanması,
- DMA aktarım süresinin veri yolu üzerinde yarattığı gecikmenin (bus contention) WCET analizine dâhil edilmesi — DMA yalnızca alan değil, zaman bölümlemesini de etkiler.

Önbellek

Önbellek (cache) iki yönden sorun çıkarır. Birincisi zamanlamadır: bir bölüm çalışırken paylaşılan önbelleği kendi verisiyle doldurur; sıradaki bölüm, soğuk önbellekle başladığı için yürütme süresi bölüm geçişlerine bağlı hâle gelir. Bu, "bir bölümün davranışı diğerinin zamanlamasını etkileyemez" iddiasını doğrudan zedeler. İkincisi tutarlılıktır. Aynı çekirdekte çalışan bölümler belleğe aynı önbellek üzerinden eriştiği için önbellek kendi başına bayat veri üretmez; sorun, önbelleği atlayan erişimlerde ortaya çıkar. DMA denetleyicisi ya da bir çevre birimi belleğe doğrudan yazdığında önbellekteki kopya eskir; önbellekteki değişiklik henüz belleğe yazılmamışken başlatılan bir DMA aktarımı ise eski veriyi gönderir. Aynı fiziksel alanın farklı sanal adreslerle eşlendiği durumlar da bazı önbellek mimarilerinde benzer bir tutarsızlık doğurabilir. Pratik yaklaşımlar:

- bölüm geçişinde önbelleğin boşaltılması (flush/invalidate) — belirlenimci (deterministic) ama performans maliyeti çizelgeye eklenmelidir,
- önbelleğin yollara/kümelere bölünerek (cache partitioning) bölümlere ayrılması,
- WCET analizinde önbelleğin tamamen soğuk kabul edilmesi gibi kötümser ama savunulabilir varsayımlar,
- DMA ve aygıt tamponlarının önbelleksiz (non-cacheable) bölgelere yerleştirilmesi ya da aktarımdan önce ve sonra önbelleğin açıkça eşitlenmesi.

Soğuk önbellek varsayımının gerçekten en kötü durum olduğu, hedef işlemci için gerekçelendirilmelidir: karmaşık boru hattına sahip işlemcilerde yerel olarak en kötü görünen başlangıç durumu, her zaman en uzun toplam süreyi vermez.

Kesmeler

Kesmeler (interrupt) zaman bölümlemesinin doğal düşmanıdır: hangi bölüm çalışırsa çalışsın, kesme geldiğinde işlemci kesme servis rutinine (interrupt service routine, ISR) dallanır ve o bölümün zaman dilimini tüketir. Sık gelen ya da beklenmedik bir kesme fırtınası, kritik bölümün penceresini eritebilir. Tipik önlemler; bölüm penceresi başına kesme kaynaklarının maskelenmesi, kesme servis rutinlerinin çekirdekte kısa tutulup işin bölüm bağlamına ertelenmesi ve kesme sıklığına donanımsal ya da yazılımsal üst sınır konmasıdır. Analizde her

kesme kaynağının en kötü durum sıklığı ve hizmet süresi, etkilediği bölümlerin bütçesine eklenmelidir. Örneğimizde bakım veri yolunun kesmesi, uçuş kontrol penceresi boyunca maskeli tutulur; böylece bakım cihazının davranışı kontrol döngüsünün süresine giremez.

Çok çekirdekli işlemciler

Buraya kadar anlatılan zaman bölümlemesi, bir anda yalnızca bir bölümün çalıştığı varsayımına dayanır. Çok çekirdekli işlemcide (multi-core processor) bölümler gerçekten eşzamanlı çalışır ve çekirdekler son seviye önbelleği, bellek denetleyicisini ve veri yolunu paylaşır. Bir çekirdekteki bölüm, başka bir çekirdekte kendi penceresi içinde çalışan bölümün yürütme süresini bu kaynaklar üzerinden uzatabilir; bu etki yollarına girişim kanalları (interference channels) denir. MMU ve pencere çizelgesi yerinde dursa bile zaman bölümlemesi böylece delinmiş olur.

Önlemler girişi ya ortadan kaldırmaya ya da sınırlandırmaya dayanır: kritik bölümün penceresinde diğer çekirdeklerin boşta tutulması, bölümlerin çekirdeklere statik atanması, paylaşılan önbelleğin bölümlenmesi, çekirdek başına bellek erişim bütçesi uygulanması ve WCET'in diğer çekirdekler en olumsuz yükü üretirken belirlenmesi. Otoritelerin beklentisi FAA AC 20-193 ve EASA AMC 20-193 belgelerindedir: girişim kanalları belirlenir, her biri için azaltma önlemi ya da sınır gösterilir ve bölümleme iddiası bu etki altında doğrulanır. Kullanılmayan çekirdeklerin gerçekten kapalı olduğu da varsayılmaz, gösterilir. Risk başlıkları [Ek B: Gerçek Zamanlı İşletim Sistemlerinde Endişe Alanları](#) sayfasında toplanmıştır.

Bölümler arası haberleşme

Bölümler tamamen yalıtılırsa sistem işlevini yerine getiremez; bir biçimde veri alışverişi gerekir. Haberleşme kanalı ise, tanımlı gereği yalıtım duvarında açılan kontrollü bir kapıdır ve yanlış tasarlanırsa hata yayılım yolu olur:

- Kanallar ve portlar konfigürasyonla **statik** tanımlanmalı; çalışma zamanında yeni kanal açılmamalıdır.
- Veri, alan bölümlemesini korumak için çekirdek denetiminde kopyalanmalı ya da tek yönlü, salt okunur eşlenmiş tamponlar kullanılmalıdır.
- Alıcı bölüm, gelen veriyi **güvenilmez girdi** gibi ele almalıdır: düşük yazılım seviyesindeki bir bölümden gelen bozuk değer, kritik bölümde aralık ve tutarlılık kontrolünden geçmeden kullanılmamalıdır.
- Bloklayan alma/gönderme çağrıları, bir bölümün diğerini bekleterek zaman bölümlemesini dolaylı yoldan bozmasına izin vermemelidir (zaman aşımı zorunlu).
- Kuyruk portunun dolması ve örnekleme portundaki verinin bayatlaması tanımlı durumlardır; alıcının her ikisine tepkisi gereksinimlerde yazılı olmalıdır.

Örneğimizde bakım bölümü, yerdeki testler için uçuş kontrol bölümüne yüzey sapması komutu gönderebilsin. Seviye A alıcı, Seviye D gönderenin hiçbir beyanına güvenmez:

```

#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>

#define TEST_CMD_MIN_DEG      (-5.0f)
#define TEST_CMD_MAX_DEG      (5.0f)
#define TEST_CMD_MAX_AGE_MS   (200u)

typedef struct {
    float surface_deg; /* bakım bölümünün istediği yüzey sapması */
} MaintTestCmd;

/* Bakım bölümünden gelen test komutunu denetler. Komut kullanılabilirse true
 * döner ve *surface_deg güncellenir; aksi hâlde çıkış değişmez.
 * maint_port_read() ve PORT_OK platformun port arayüzünden gelir. */
bool maint_cmd_accept(bool on_ground, float *surface_deg)
{
    MaintTestCmd cmd;
    uint32_t age_ms = 0u; /* iletinin yaşını çekirdek ölçer, gönderen değil
 */
    bool accepted = false;

    if ((surface_deg != NULL)
        && (maint_port_read(&cmd, &age_ms) == PORT_OK)) {
        /* NaN her iki aralık karşılaştırmasında da elenir. */
        if (on_ground
            && (age_ms <= TEST_CMD_MAX_AGE_MS)
            && (cmd.surface_deg >= TEST_CMD_MIN_DEG)
            && (cmd.surface_deg <= TEST_CMD_MAX_DEG)) {
            *surface_deg = cmd.surface_deg;
            accepted = true;
        }
    }

    return accepted;
}

```

Dört denetim de ayrı bir hata yolunu kapatır: port çağrısının dönüş değeri boş ya da hatalı okumayı, yaş sınırı bayat komutu, aralık denetimi bozuk değeri, yerde olma koşulu ise komutun yanlış uçuş evresinde kabul edilmesini eler. Tazelik bilgisinin iletinin içinden değil çekirdekten alınması bilinçli bir tercihtir; gönderenin yazdığı zaman damgası da gönderen kadar güvenilir değildir. Bu denetimlerin her biri alıcı bölümün gereksinimlerinde yer alır ve gürbüzlük testleriyle doğrulanır. Bu yalnızca iyi uygulama değildir: DO-178C, yazılım mimarisinin tutarlılığı gözden geçirilirken, daha düşük seviyeli bir bileşenle arayüzü olan yüksek seviyeli bileşenin olası hatalı girdilere karşı kendini koruduğunun da doğrulanmasını bekler.

Unsur	Tehdit ettiği eksen	Tipik önlem
DMA	Alan + zaman	DMA erişiminin çekirdekte toplanması, hedef adres doğrulama
Önbellek	Zaman (ve tutarlılık)	Geçişte boşaltma, önbellek bölümleme, kötümser WCET

Unsur	Tehdit ettiği eksen	Tipik önlem
Kesmeler	Zaman	Pencere bazlı maskeleye, kısa kesme servis rutini, sıklık sınırı
Çok çekirdek	Zaman	Girişim kanallarının belirlenmesi, çekirdek ataması, girişim altında WCET
Bölümler arası haberleşme	Alan + zaman	Statik kanallar, çekirdek denetimli kopya, zaman aşımı, girdi doğrulama

Bu unsurların ortak özelliği, tek tek bakıldığında zararsız görünmeleridir; bölümleme ihlalleri genellikle iki mekanizmanın kesişiminde (örneğin pencere sınırını aşan bir DMA aktarımı) ortaya çıkar. Bu yüzden bölümleme analizi, mekanizma listesi değil, etkileşim matrisi üzerinden yürütülmelidir.

Bölümlemenin doğrulanması

Bölümleme yalnızca tasarım metniyle değil, analiz ve testle gösterilir; ikisi birbirinin yerini tutmaz. Analiz hangi yayılma yollarının var olduğunu ve her birinin nasıl kapatıldığını söyler; test, kapatma mekanizmasının hedef donanımda gerçekten çalıştığını gösterir.

Bölümleme analizi

Analizin çıkış noktası bölümlerin listesi değil, paylaştıkları kaynakların listesidir. Paylaşılan her kaynak için şu sorular yanıtlanır: bir bölüm bu kaynak üzerinden diğerini nasıl etkileyebilir, bunu hangi mekanizma önler ve mekanizmanın çalıştığının kanıtı nerededir? Örneğimiz için matrisin bir kesiti şöyledir:

Paylaşılan kaynak	Yayılma yolu	Önlem	Kanıt
RAM	Bakım bölümünün uçuş kontrol verisine yazması	MMU eşlemesi, kullanıcı kipi	Bellek haritası analizi, sınır dışı yazma testi
İşlemci zamanı	Bakım bölümünün penceresini aşması	Pencere sonunda zamanlayıcı kesmesi	WCET analizi, pencere aşımı testi
Veri yolu ve DMA	Pencere dışına sarkan aktarım	DMA'nın çekirdekte toplanması	Tasarım gözden geçirmesi, zamanlama ölçümü
Kesme hattı	Bakım veri yolundan kesme fırtınası	Pencere bazlı maskeleye	Kesme fırtınası testi
Haberleşme portu	Bozuk ya da bayat komut	Alicıda denetim, zaman aşımı	Bozuk ileti testi

Kanıt sütunu boş kalan satır, kapatılmamış bir yayılma yoludur. Analiz kısmında dayanan teknikler [9. Yazılım Doğrulama](#) bölümünde anlatılanlardır: bellek haritası analizi, bağlayıcı çıktısındaki yerleşimin koruma tablolarıyla örtüştüğünü; WCET analizi her pencerenin yeterli olduğunu; yığın analizi hiçbir bölümün kendi yığını aşmadığını gösterir.

İhlal testleri

Olumlu testler ("her bölüm kendi işini yapıyor") bölümleme hakkında neredeyse hiçbir şey söylemez. Bölümlemeyi sınanan, kasıtlı olarak kurala uymayan bir test bölümüdür. Hedef donanımda, gerçek konfigürasyonla çalıştırılan tipik senaryolar şunlardır:

- başka bir bölümün verisine, çekirdek alanına ve korunan yazmaçlara okuma, yazma ve dallanma denemesi; ayrıcalıklı komut çalıştırma denemesi,
- sonsuz döngü ya da uzun hesaplama penceresinin aşılması,
- yığın taşıma,
- bölümün sahip olduğu aygıttan en yüksek sıklıkta kesme üretilmesi,
- portların doldurulması, bozuk ve aralık dışı iletilerin gönderilmesi, bloklayan çağrılarda süresiz bekleme denemesi.

Her senaryoda üç şey gözlenir: ihlal algılanmış mıdır, tepki sağlık izleme tablosunda tanımlı olan mıdır ve diğer bölümlerin verisi ile zamanlaması değişmeden kalmış mıdır? Sonuncusu ölçümle gösterilir; örneğin uçuş kontrol penceresinin başlama anı ve süresi, ihlal senaryosu çalışırken kaydedilir. DO-178C, donanım/yazılım entegrasyon testlerinin ortaya çıkarması beklenen hata türleri arasında bölümleme ihlallerini de sayar; bellek yönetim donanımının hatalı denetlenmesi, yığın taşması ve yürütme süresi gereksinimlerinin karşılanamaması da aynı gruptadır. Standardın test ortamını belirli kıldığı tek test yöntemi budur: yazılım hedef bilgisayarda çalışırken sınanır.

Konfigürasyon tablolarının doğrulanması

Bölümleme çekirdeği kusursuz olsa bile yanlış bir tablo, yanlış sınırı kusursuzca uygular. Tablolar bu yüzden bölümleme mekanizmasının parçası sayılır ve platformun yazılım seviyesinde doğrulanır. Tipik denetimler: bellek bölgeleri örtüşüyor mu, her bölge yalnızca amaçlanan bölümlere ve amaçlanan haklarla mı eşlenmiş, pencerelerin toplamı ana çerçeveye eşit mi, her pencere ilgili bölümün WCET değerini ve geçiş maliyetini karşılıyor mu, her kanalın tek bir kaynağı var mı, her sağlık izleme olayına bir tepki atanmış mı? Tabloları bir araç üretiyor ya da denetliyor ve çıktısı ayrıca doğrulanmıyorsa araç kalifikasyonu gündeme gelir (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)).

Bölümleme kanıtı, doğrulandığı konfigürasyon için geçerlidir. Bir pencere uzatıldığında, yeni bir port eklendiğinde ya da bir bölge büyütüldüğünde değişiklik etki analizi (change impact analysis) yalnızca değişen bölümü değil, matrisin ilgili satırlarını da yeniden ele alır. Denetimde sorulan soru genellikle yalındır: "Şu bölüm sonsuz döngüye girerse ya da bozuk bir işaretçiyle yazarsa ne olur ve bunu hangi kayıt gösteriyor?"

Sık yapılan hatalar

- **Doğrulanmamış konfigürasyon.** Çekirdek özenle doğrulanır, tablolar ise entegrasyonun son haftasında elle düzenlenir; oysa sınırı çizen tablodur ("Konfigürasyon tablolarının doğrulanması").
- **Korumasız bırakılan paylaşılan kaynak.** RAM bölünür, ama çevre birimi yazmaçları, DMA denetleyicisi ya da kalıcı bellek herkese açık kalır; hata bu açıklıktan sızar ("Paylaşılan giriş/çıkışın yönetimi", "Doğrudan bellek erişimi").
- **Kâğıt üzerinde kalan zaman bütçesi.** Bağlam değiştirme, kesme yükü, önbellek ve çok çekirdek girişimi pencere hesabına katılmaz; çizelge ölçümde tutmaz ("Zaman bölümlemesi", "Bölümlemeyi zorlaştıran konular").
- **Yalnızca olumlu test.** Sınırın içinde kalan davranış test edilir, sınırı zorlayan davranış edilmez; bölümleme iddiası böylece sınanmamış kalır ("İhlal testleri").
- **Platform kanıtına koşulsuz güvenmek.** Tedarikçinin bölümleme kanıtı, kullanım kısıtlarına uyulduğu sürece geçerlidir; bu kısıtların projede karşılandığı ayrıca gösterilmelidir ("ARINC 653 ve IMA bağlamı").

Bu bölümden akılda kalması gerekenler

- Bölümleme, etkileşimi sınırlayan mimari önlemdir; alan ve zaman eksenleri birlikte sağlanmadıkça paylaşılan platformda yalıtım eksik kalır.
- Yalıtım gösterilemezse bütün bileşenler en yüksek seviyeyle geliştirilir; bölümlemeyi sağlayan yazılımın kendisi de en az koruduğu en yüksek yazılım seviyesindedir, bölümlemeyi sağlayan donanım ise sistem emniyet değerlendirmesinde ele alınır.
- Alan bölümlemesi MMU/MPU gibi donanım zorlamasına, zaman bölümlemesi öngörülebilir ve statik bir çizelgeye dayanmalıdır; varsayım kanıt değildir.
- Paylaşılan giriş/çıkış, DMA, önbellek, kesmeler ve çok çekirdekli işlemcilerdeki girişim kanalları bölümleme sınırlarının etrafından dolaşabilen yollardır; analizde tek tek ele alınmalıdır.
- Bölümler arası haberleşme kanalları statik tanımlanmalı, gelen veri güvenilir girdi gibi doğrulanmalı ve bekleme zaman aşımıyla sınırlanmalıdır.
- Bölümleme bütünlüğü Seviye A'dan Seviye D'ye dört seviyede de aranan bir doğrulama hedefidir: paylaşılan kaynaklar üzerinden yürütülen analizle ve hedef donanımda yapılan ihlal testleriyle gösterilir.
- Konfigürasyon tabloları bölümleme mekanizmasının parçasıdır; platformun yazılım seviyesinde doğrulanır ve her değişiklikte analiz yeniden ele alınır.

22. Konfigürasyon Verisi

Konfigürasyon verisi (configuration data), kod değişmeden yazılımın davranışını belirleyen parametre ve tablolardır. DO-178C, koddan ayrı yönetilen bu tür veriyi parametre verisi ögesi (parameter data item, PDI) adıyla ele alır ve kod kadar sıkı doğrulanmasını bekler.

Bu bölüm, hangi verinin hangi kurallarla yönetildiğini, PDI'nin koddan ayrı doğrulanabilmesinin koşullarını ve verinin sürüm, uyumluluk ve değer hatalarına karşı nasıl korunacağını anlatır.

Konfigürasyon verisi nedir ve neden kod kadar önemlidir?

Aviyonik yazılımın davranışı yalnızca koddan okunmaz. Aynı çalıştırılabilir nesne kodu (executable object code), okuduğu veriye göre farklı bir motor tipinin limitlerini uygular, farklı bir bölgenin frekans tablosunu kullanır ya da opsiyonel bir işlevi açıp kapatır. Kod sabit kalırken davranışı değiştiren bu veriye konfigürasyon verisi denir. Uygulamada dört türü öne çıkar:

- **Eşik ve limit değerleri:** uyarı eşikleri, zaman aşımı süreleri, motor ya da uçak tipine göre değişen limitler.
- **Tablolar:** kalibrasyon eğrileri, frekans ve kanal tabloları, arama tabloları.
- **Seçenek verisi:** opsiyonel donanımın takılı olup olmadığı, müşteri seçenekleri, ölçü birimi tercihleri.
- **Platform tanımları:** giriş/çıkış eşlemeleri, veri yolu adresleri, bölümlere ayrılan bellek ve zaman bütçeleri.

Bu verinin kod kadar önemli olmasının nedeni basittir: yanlış bir değer, hatasız bir kodu yanlış çalıştırır. Üstelik veri hatası kod hatasından daha sessizdir. Derleyici uyarı vermez, statik analiz bir şey görmez, kod gözden geçirmesi eşiğin 950 yerine 590 yazıldığını fark edemez; gereksinim tabanlı testler de çoğu zaman uçağa yüklenecek dosyayla değil, bir test veri kümesiyle koşulmuştur.

Verinin yokluğu, yanlışlığı kadar tehlikelidir. 2015'te Sevilla'da bir A400M, teslim öncesi ilk test uçuşunda düştü. Üretici, üç motorun kalkıştan kısa süre sonra güç kumandasına yanıt vermediğini ve sorunun son montajdaki yazılım yüklemesinden kaynaklandığını açıkladı; soruşturmadan kamuoyuna yansıyan bilgiye göre bu yükleme sırasında üç motorun kontrol birimindeki tork kalibrasyon parametreleri silinmişti. Kodun tek satırı değişmemişti; eksik olan veriydi.

Adlandırmayla ilgili iki not, sonraki karışıklıkları önler. Birincisi, konfigürasyon verisi ile konfigürasyon yönetimi (configuration management) ayrı şeylerdir: ilki yazılımın davranışını ayarlayan veridir, ikincisi bütün yaşam döngüsü verisini kimliklendirip kontrol altında tutan

süreçtir (bkz. [10. Yazılım Konfigürasyon Yönetimi](#)). Konfigürasyon verisi, o sürecin yönettiği öğelerden yalnızca biridir. İkincisi, "konfigürasyon verisi" gündelik bir şemsiye terimdir; DO-178C'nin tanımladığı kavram PDI'dir ve davranışı etkileyen her veri PDI değildir.

Hangi veri hangi rejime girer?

Davranışı etkileyen verinin hepsi aynı kurallarla yönetilmez. Rejimi üç soru belirler: veri çalıştırılabilir nesne kodunun içinde mi dışında mı, onu kim değiştirebilir ve yanlış bir değer emniyeti etkileyebilir mi?

Veri türü	Tipik örnek	Kim değiştirir?	Nasıl güvence altına alınır?	İlgili bölüm
Koda gömülü sabit ve tablo	Derleme zamanı sabiti, kaynak koddaki sabit dizi	Geliştirici; her değişiklik yeni derlemedir	Çalıştırılabilir nesne kodunun parçasıdır; kodla birlikte gözden geçirilir ve test edilir. PDI değildir	8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon
Parametre verisi ögesi	Motor limit tablosu, kalibrasyon dosyası, frekans tablosu	Geliştirici ya da entegratör; onaylı süreçle	Onaylı konfigürasyonun parçasıdır; her dosya doğrulanır, koşulları sağlanırsa koddan ayrı	Bu bölüm
Seçenek verisi	Yazılımla programlanan seçenek, pin programlama girişi	Uçak üreticisi ya da kurulumu yapan kuruluş; onaylı seçenekler arasından	Seçilmeyen işlevin kodu devre dışı bırakılmış kod sayılır; onaylanmamış konfigürasyonun istem dışı seçilemeyeceği gösterilir	17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar
Platform konfigürasyon tabloları	ARINC 653 zaman çizelgesi, bellek ve port tabloları	Platform entegratörü	Genellikle PDI olarak yönetilir; bölümlene analizinin girdisidir	21. Yazılım Bölümlenmesi
Kullanıcı tarafından değiştirilebilir veri	Emniyeti etkilemediği gösterilmiş operatör tercihleri	Operatör; belirlenmiş kısıtlar içinde	Onaylanan, verinin kendisi değil koruma mekanizması ve değişiklik kısıtlarıdır	19. Kullanıcı Tarafından Değiştirilebilir Yazılım
Havacılık veri tabanı	Seyrüsefer, arazi ve engel verisi	Veri tedarikçisi; dönemsel güncellemeyle	Veri zinciri DO-200B / ED-76A ile güvence altına alınır; veriyi okuyan kod DO-178C kapsamındadır	23. Havacılık Verileri

Tabloda "sahada yüklenen veri" diye bir satır yoktur, çünkü sahada yükleme bir veri türü değil, bir teslim yoludur. PDI dosyası da seçenek dosyası da uçakta, ekipman sökülmeden yüklenebilir; o zaman bu bölümdeki kurallara [18. Sahada Yüklenebilir Yazılım](#) bölümündeki yükleme, bütünlük ve parça kimliği koşulları eklenir.

PDI ile kullanıcı tarafından değiştirilebilir veri. En sık karıştırılan iki satır bunlardır, çünkü ikisi de "parametre tablosu" biçiminde görünür. Ayrım verinin türünde değil, kimin hangi güvenceyle değiştirdiğindedir. PDI onaylı yazılım konfigürasyonunun parçasıdır: her dosyanın doğru değerleri taşıdığı gösterilir ve değişen dosya, yeni bir onaylı konfigürasyon demektir. Kullanıcı tarafından değiştirilebilir yazılımda (user-modifiable software, UMS) ise veri, ilk sertifikasyonda belirlenen kısıtlar içinde kaldığı sürece otoritenin ya da üreticinin gözden geçirmesi olmadan operatörce değiştirilir. Bu ancak kısıtlar içindeki hiçbir değişikliğin emniyeti olumsuz etkileyemeyeceği sistem düzeyinde gösterilmişse mümkündür. Pratik ölçüt şudur: veri için "doğru değer" diye bir şey varsa ve yanlış değer emniyeti etkileyebiliyorsa o veri PDI olarak (ya da kodun içinde) yönetilir; koruma mekanizmasının izin verdiği her değer emniyetliyse UMS adaydır. Biçim açısından ikisi çakışabilir: operatörün değiştireceği veri de ayrı bir konfigürasyon ögesi olarak tutulabilir ve DO-178C, kullanım biçimi bunu gerektirdiğinde PDI'ye UMS rehberliğinin de uygulanmasını ister.

Seçenek verisi. DO-178C, opsiyonel işlevlerin donanım pinleri yerine yazılımla programlanan seçeneklerle seçildiği durumu seçenekle seçilebilir yazılım (option-selectable software) adıyla anar ve onaylanmamış bir konfigürasyonun istem dışı seçilememesini bekler. Seçenek kapalıyken çalışmayan kod, devre dışı bırakılmış kod (deactivated code) olarak ele alınır. Seçenekler koddan ayrı bir dosyada tutuluyorsa o dosya çoğu zaman PDI dosyası olarak da yönetilir; iki rejim birbirini dışlamaz, üst üste biner.

Platform tabloları. Bölümlemeli bir işletim sisteminde hangi bölümün (partition) ne kadar bellek ve hangi zaman penceresini alacağını söyleyen tablolar, uygulama kodunda tek satır değişmeden bütün sistemin zamanlamasını değiştirir. Bu tablolardaki bir hata tek bir işlevi değil, bölümlemenin kendisini bozar; bu yüzden bölümlemeyi sağlayan platform yazılımının seviyesinde doğrulanırlar (çizelgeleme tarafı için bkz. [20. Gerçek Zamanlı İşletim Sistemleri](#)).

Parametre verisi öğeleri ve DO-178C

DO-178C, önceki sürümde muğlak kalan bir soruya doğrudan cevap verir: kodun dışında tutulan ama davranışı belirleyen veri nasıl sertifikaya edilir? Bunun için **parametre verisi ögesi** kavramı tanımlanmıştır. PDI, çalıştırılabilir nesne kodunu değiştirmeden yazılımın davranışını etkileyen ve ayrı bir konfigürasyon ögesi (configuration item) olarak yönetilen veri kümesidir. Standart iki adı ayırır: PDI, öğelerinin yapısı ve öznitelikleriyle tanımlanan kümedir; PDI dosyası ise bu kümenin her ögesine değer atanmış, hedef bilgisayarın doğrudan kullanabildiği bir örneğidir. Aynı PDI'nin birden çok dosyası olabilir.

Bu kavramın asıl kazandırdığı şey **ayrık yaşam döngüsüdür**: çalıştırılabilir nesne kodu bir kez üretilip doğrulandıktan sonra, yalnızca PDI dosyası değişerek yeni bir uçak konfigürasyonu tanımlanabilir. Kod yeniden derlenmez, kodun test kampanyası tekrarlanmaz; yalnızca yeni veri, kendi doğrulama sürecinden geçirilir. Bu, bölge frekans tabloları, motor tipine göre limitler veya müşteri seçenekleri gibi sık değişen veriler için ciddi bir verimlilik sağlar.

Ayrı doğrulamanın koşulları

Bunun bedava olmadığını da baştan söylemek gerekir. Zemin, PDI'nin **yapısının ve özniteliklerinin** (tip, birim, sınır, çözünürlük) açıkça tanımlanmasıdır. Bu tanım, kodun veriyi nasıl kullandığıyla birlikte yüksek seviyeli gereksinim (high-level requirement) olarak yazılır ve izlenebilirlik (traceability) kurulur; veri kodla birlikte doğrulanacak olsa da bu beklenti değişmez. Ayrık yaşam döngüsü ise ancak DO-178C'nin PDI doğrulamasına ayırdığı maddede (6.6) aranan dört koşulun tümü sağlandığında savunulabilir:

- Çalıştırılabilir nesne kodunun, tanımlı yapıya uyan **her geçerli PDI değeriyle** doğru çalıştığı normal aralık testleriyle gösterilir. Kodun doğrulaması tek bir veri kümesine göre değil, veri zarfının (data envelope) tamamına göre yapılır; aksi hâlde her yeni PDI dosyası kodun yeniden testini gerektirir.
- Kod, tanımlı yapı ve özniteliklere **uymayan** PDI dosyasına karşı gürbüzdür (robust): bozuk, eksik, yanlış sürümlü ya da aralık dışı değer taşıyan dosyayla karşılaştığında gereksinimlerde öngörülen tepkiyi verir. Bu, gürbüzlük testleriyle gösterilir.
- PDI içeriğinin kodda yol açabileceği **bütün davranış doğrulanabilir** olmalıdır. Veri yalnızca eşik ve tablo değeri taşıyorsa bu kolaydır; adres, ofset ya da yorumlanan komut dizisi taşımaya başladığında davranış uzayı sınırlanamaz hâle gelir ve koşul zorlanır.
- Yaşam döngüsü verisinin yapısı PDI'nin **ayrı yönetilmesine** izin verir: gereksinimi, izlenebilirliği, doğrulama kaydı ve konfigürasyon kimliği koddan ayrı tutulabilir.

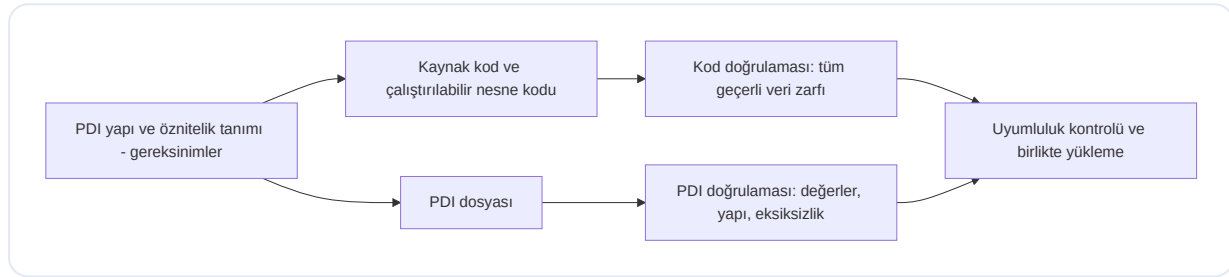
Bu koşullardan biri sağlanamıyorsa PDI, çalıştırılabilir nesne koduyla **birlikte** doğrulanır; yani her yeni dosya, kodun ilgili testlerinin o dosyayla yeniden koşulması demektir. Koşullar sağlandığında da her PDI dosyasının kendi doğrulaması kalır. DO-178C bunu, kodlama ve entegrasyon çıktılarının doğrulandığı Tablo A-5'teki iki hedefle ister; kanıt test, analiz ve gözden geçirmenin bir bileşimiyle üretilebilir:

Hedef (Tablo A-5)	Dosya için sorulan soru	Hangi seviyelerde?
8: PDI dosyası doğru ve eksiksizdir	Dosya, yüksek seviyeli gereksinimlerdeki yapıya uyuyor ve orada tanımlanmamış öge taşıyor mu? Her ögenin değeri doğru, özniteliklerine uygun ve diğer öğelerle tutarlı mı (örneğin uyarı eşiği limitin altında mı)?	Seviye A–D; A ve B'de bağımsızlıkla
9: PDI dosyasının doğrulaması tamamlanmıştır	Dosyadaki öğelerin hepsi doğrulama sırasında ele alındı mı, bakılmadan geçilen öge kaldı mı?	Seviye A–C; A ve B'de bağımsızlıkla

Her öğede aynı derinlik gerekmez: kimi öğede özniteliklere uygunluğu göstermek yeter, kimisinde değerin kendisi doğrulanır. Hangi öğenin hangi gruba girdiği doğrulama planında gerekçesiyle yazılmalıdır.

Dosyanın, birlikte çalışacağı çalıştırılabilir nesne kodu sürümüyle **uyumluluğu** bu iki hedefin dışında, ayrıca ele alınır. Standart, yükleme kontrolü ile uyumluluğun planlamada ele alınmasını ister ve sahada yüklenen PDI için bozuk dosyanın ve kodla uyumsuzluğun saptanmasına özellikle dikkat çeker. Bunun için PDI dosyası tipik olarak bir başlık taşır: yapı tanımının sürümü, uyumlu yazılımın parça numarası ve dosyanın bütününe kapsayan bir döngüsel artıklık denetimi (cyclic redundancy check, CRC) değeri. Yazılım bu başlığı yükleme sırasında ve her açılıştan denetler.

PDI dosyası yazılım seviyesini (software level) kod üzerinden miras alır: Seviye A bir yazılımın davranışını belirleyen tablo, Seviye A titizliğiyle doğrulanır. Ayrıca PDI dosyası konfigürasyon yönetimi açısından kendi başına bir konfigürasyon öğesidir — kendi parça numarası, kendi sürüm geçmiş, kendi problem raporları olur.



Veri zarfına göre doğrulama pratikte ne demektir?

Veri zarfı, gereksinimlerdeki öznitelik sınırlarının toplamıdır ve test durumlarına doğrudan çevrilir: her öğe için alt sınır, üst sınır ve tipik değer; birbirine bağlı öğeler için uç bileşimler (en küçük ve en büyük tablo boyutu gibi). Bu testler uçağa gidecek dosyayla değil, zarfın köşelerini bilerek zorlayan **test PDI dosyalarıyla** koşular; bu dosyalar da doğrulama durumlarının parçası olarak konfigürasyon kontrolü altında tutulur.

Yapısal kapsam analizi (structural coverage analysis) aynı mantığı izler. Bir kararın koşulu PDI'den geliyorsa, tek bir dosyayla koşulan testler o kararın yalnızca bir yönünü görür; boşluk, zarfın öbür ucundaki bir test dosyasıyla kapatılır. Yol zarftaki hiçbir dosyayla çalıştırılmıyorsa kapsanmayan kod için bilinen sorular sorulur: gereksinim mi eksiktir, kod mu gereksizdir, yoksa yol tasarım gereği devre dışı mı bırakılmıştır? En kötü durum yürütme süresi (worst-case execution time, WCET) ve bellek kullanımı da zarfın en ağır ucuna (en uzun tablo, en çok kayıt) göre analiz edilir; yoksa "yalnızca veri" olan bir güncelleme zaman bütçesini aşabilir.

Zarf, değişikliklerde karar ölçütü olur. Yeni bir PDI dosyası zarfın içinde kalıyorsa değişiklik etki analizi (change impact analysis) yalnızca dosyanın doğrulanmasını ve uyumluluk kaydının güncellenmesini gerektirir. Dosya zarfın dışına çıkıyorsa — yeni bir alan, genişleyen bir aralık, büyüyen bir tablo — bu bir veri güncellemesi değil, gereksinim değişikliğidir ve kodun etkilenen kısmı yeniden doğrulanır. DO-178C'nin beklentisi de bu yöndedir: PDI'nin yapısı ya da öznitelikleri

değiştğinde kodun değiştirilmesi ve yeniden doğrulanması gerekip gerekmediği analiz edilir. Zarfı gereğinden dar tanımlamak ileride her yeni konfigürasyonda kod doğrulamasını yeniden açar; gereğinden geniş tanımlamak bugünkü test yükünü büyütür. Bu denge, proje başında bilerek kurulmalıdır.

PDI dosyasının üretimi ve değerlerin kaynağı

Pratikte en sık görülen hata, PDI'yi "sadece veri" sayıp doğrulamasını üretim betiklerine gömülü birkaç otomatik kontrole indirgemektir. Değerin *kaynağının* doğrulanması — örneğin bir frekans tablosundaki her satırın onaylı bir sistem verisine dayandığının gösterilmesi — otomatik biçim kontrolünün yapamayacağı, gözden geçirme gerektiren bir iştir. Denetimde sorulan soru da genellikle budur: "Bu dosyadaki şu değer nereden geldi, kim neye karşı doğruladı?"

PDI dosyaları çoğunlukla elle yazılmaz; bir tablodan ya da veri tabanından ikili dosya üreten bir araçla oluşturulur. Bu aracın çıktısı uçağa yüklenen yazılımın parçasıdır. Çıktı ayrıca doğrulanıyorsa — örneğin ikili dosya bağımsız bir yolla geri okunup kaynak değerlerle karşılaştırılıyorsa — araca güvenmek gerekmez; doğrulanmıyorsa araç kalifikasyonu gündeme gelir (bkz. [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)).

Konfigürasyon verisi örneği: motor limit tablosu

Bir motor izleme işlevinin, motor tipine göre değişen limitleri bir PDI dosyasından okuduğunu düşünelim. Yapı tanımlı gereksinimlerde yazılıdır; koddaki karşılığı ve yazılımın dosyayı kullanmadan önce yaptığı denetim şöyle olabilir:

```

#define PDI_SEMA_SURUMU      (3u)          /* kodun beklediği yapı sürümü */
#define PDI_UYUMLU_YAZILIM  (0x00A71503u) /* çekirdek yazılımın parça kimliği */
#define EGT_MIN_C            (600)         /* gereksinimlerdeki alt sınır */
#define EGT_MAX_C            (1100)        /* gereksinimlerdeki üst sınır */
#define N1_MIN_BINDE         (900u)        /* %90,0 */
#define N1_MAX_BINDE         (1100u)       /* %110,0 */

typedef struct {
    uint32_t sema_surumu; /* yapı tanımının sürümü */
    uint32_t uyumlu_yazilim; /* birlikte onaylanan çekirdek yazılım kimliği */
    int16_t egt_uyari_c; /* egzoz gazı sıcaklığı uyarı eşiği, °C */
    int16_t egt_limit_c; /* egzoz gazı sıcaklığı limiti, °C */
    uint16_t n1_limit_binde; /* fan devri limiti, binde (%0,1) */
    uint16_t ayrilmis; /* kullanılmıyor; sıfır olmalı */
    uint32_t crc32; /* önceki tüm alanların CRC değeri */
} motor_pdi_t;

typedef enum {
    PDI_GECERLI = 0,
    PDI_BOZUK,
    PDI_UYUMSUZ,
    PDI_ARALIK_DISI
} pdi_durum_t;

/* CRC-32 hesabı; başka bir birimde tanımlıdır. */
uint32_t crc32_hesapla(const uint8_t *veri, size_t boyut);

pdi_durum_t motor_pdi_denetle(const motor_pdi_t *p)
{
    pdi_durum_t durum;

    if (p == NULL) {
        durum = PDI_BOZUK; /* dosya yok */
    } else if (crc32_hesapla((const uint8_t *)p,
        offsetof(motor_pdi_t, crc32)) != p->crc32) {
        durum = PDI_BOZUK; /* bütünlük */
    } else if ((p->sema_surumu != PDI_SEMA_SURUMU) ||
        (p->uyumlu_yazilim != PDI_UYUMLU_YAZILIM)) {
        durum = PDI_UYUMSUZ; /* yapı ve yazılım uyumluluğu */
    } else if ((p->egt_limit_c < EGT_MIN_C) || (p->egt_limit_c > EGT_MAX_C) ||
        (p->egt_uyari_c < EGT_MIN_C) ||
        (p->egt_uyari_c >= p->egt_limit_c) ||
        (p->n1_limit_binde < N1_MIN_BINDE) ||
        (p->n1_limit_binde > N1_MAX_BINDE) ||
        (p->ayrilmis != 0u)) {
        durum = PDI_ARALIK_DISI; /* öznitelikler ve iç tutarlılık */
    } else {
        durum = PDI_GECERLI;
    }
    return durum;
}

```

Örnek, dosyanın bellekte bu yapıyla aynı yerleşimde bulunduğunu varsayar: alanların sırası, hizalaması ve bayt sırası da yapı tanımının parçasıdır ve PDI dosyasını üreten araç bu tanıma göre yazar. Yerleşimin derleyiciye ve işlemciye bağlı kalması istenmiyorsa alanlar, 23. bölümdeki veri tabanı örneğinde olduğu gibi bayt bayt okunur.

Denetimin sırası rastgele değildir: bütünlüğü doğrulanmamış bir dosyanın sürüm alanına, sürümü doğrulanmamış bir dosyanın değerlerine güvenilmez. Dosya geçersizse yazılımın ne yapacağı da bir gereksinimdir ve sistem emniyet değerlendirme süreciyle birlikte belirlenir: en korumacı limitlerle çalışıp arızayı bildirmek, işlevi başlatmamak ya da ekipmanı arızalı ilan etmek. Kabul edilemeyecek tek seçenek, geçersiz dosyayı sessizce kullanmaktır.

Aynı çalıştırılabilir nesne kodu, bu yapıya uyan iki ayrı dosyayla iki ayrı uçak konfigürasyonunu tanımlar (değerler kurgusaldır):

Alan	PDI dosyası 1: motor tipi A	PDI dosyası 2: motor tipi B
sema_surumu	3	3
uyumlu_yazilim	0x00A71503	0x00A71503
egt_uyari_c	900	950
egt_limit_c	950	1010
n1_limit_binde	1000	1040

İki dosyanın ayrı parça numarası, ayrı sürüm geçmişi ve ayrı doğrulama kaydı vardır; kod ise tektir ve zarfın tamamıyla (600–1100 °C, %90,0–%110,0) bir kez doğrulanmıştır. Örnek, yükleme anındaki denetimin sınırını da gösterir: denetim, B tipi motora yanlılıkla 1. dosyanın yüklendiğini yakalayamaz, çünkü o dosyanın bütün değerleri geçerli aralıktadır. Değerin *bu uçak için* doğru olduğunu yazılımın iç denetimi değil, PDI dosyasının doğrulanması ve yükleme sonrasındaki parça numarası kontrolü gösterir.

PDI yaşam döngüsü verisinde nerede görünür?

PDI yalnızca bir dosya değildir; planlardan başarı özetine kadar bütün veride iz bırakır. Tipik dağılım şöyledir:

Yaşam döngüsü verisi	PDI ile ilgili içerik
Yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC) ve diğer planlar	PDI'nin ne için kullanıldığı ve yazılım seviyesi; ayrı doğrulama yolunun seçilip seçilmediği; dosyanın hangi süreç ve araçlarla üretilip doğrulanacağı ve değiştirileceği, araç kalifikasyonu ihtiyacı; yükleme kontrolü ve uyumluluk yaklaşımı
Yazılım gereksinim verisi	PDI yapı tanımı: öğelerin tipi, birimi, sınırları, çözünürlüğü; kodun veriyi nasıl kullandığı; gerektiğinde değerlerin kendisi; izlenebilirlik
PDI dosyası	Belirli bir konfigürasyona ait değerlerin yüklenebilir hâli; kendi parça numarası ve sürümüyle ayrı bir konfigürasyon ögesi. Entegrasyon sürecinin çıktısıdır ve Seviye A–D'nin hepsinde en sıkı kontrol kategorisindedir (CC1). Her dosya için yaşam döngüsü verisi üretilir; bu veri ayrı paketleniyorsa ilgili çalıştırılabilir nesne kodunun SAS'ına atıf yapar

Yaşam döngüsü verisi	PDI ile ilgili içerik
Doğrulama durumları, prosedürleri ve sonuçları	Kodun veri zarfına göre normal aralık ve gürbüzlük testleri; her PDI dosyası için değer doğruluğu, yapı uyumu ve eksiksizlik kanıtları
Yazılım konfigürasyon indeksi (Software Configuration Index, SCI)	PDI dosyalarının kimliği ve sürümü, dosyanın nasıl üretildiği, uyumluluk kaydı: hangi PDI sürümünün hangi çalıştırılabilir nesne kodu sürümüyle geçerli olduğu
Yazılım başarı özeti (Software Accomplishment Summary, SAS)	Onaya sunulan konfigürasyondaki PDI dosyaları, planlanan yaklaşımdan sapmalar ve PDI ile ilgili açık problem raporları

Bu kararların çoğu planlama aşamasında verilir. Ayrı doğrulama yolu seçilecekse bunun PSAC'ta beyan edilmesi ve otoriteyle erken konuşulması gerekir; proje sonunda "bu tablo aslında PDI idi" demek, eksik gereksinim ve eksik gürbüzlük testi olarak geri döner. Planlama tarafı [5. Yazılım Planlama](#), doğrulama hedefleri [9. Yazılım Doğrulama](#), otoriteye sunulan veri ise [12. Sertifikasyon İrtibatı](#) bölümünde anlatılır.

Tipik riskler ve önlemleri

Risk	Belirti	Önem
Yanlış tablonun yüklenmesi	Değerler geçerli aralıktadır, yazılım arıza bildirmez; davranış başka bir konfigürasyona aittir	Dosya başlığında konfigürasyon kimliği; yükleme sonrasında parça numarasının kurulum kaydıyla karşılaştırılması
Uyumsuz sürüm	Yeni yazılım eski yapıdaki dosyayı okur; alanlar kayar, değerler anlamsızlaşır	Başlıkta yapı sürümü ve uyumlu yazılım kimliği; SCI'da uyumluluk kaydı; uyumsuz dosyanın açılışta reddedilmesi
Kaynağı belirsiz değer	Gözden geçirmede "bu sayı nereden geldi?" sorusu cevapsız kalır	Her değer onaylı bir kaynağa (sistem gereksinimi, kalibrasyon kaydı) izlenebilmesi; değerlerin yazarı dışında biri tarafından gözden geçirilmesi
Birim ya da ölçek hatası	Değer makul görünür ama on ya da bin kat sapmıştır; birimler karışmıştır	Yapı tanımında birim ve çözünürlük; alan adında birim; dar tutulmuş aralık ve öğeler arası tutarlılık denetimleri
Kısmi ya da kesilmiş yükleme	Dosyanın bir kısmı yeni, bir kısmı eskidir ya da dosya hiç yoktur	Dosyanın bütünü kapsayan CRC; dosya yokluğunun geçersiz dosya sayılması; yükleme tamamlanmadan dosyanın geçerli işaretlenmemesi
Zarfın dışına çıkan değer	Yeni konfigürasyon için bir sınır "biraz" genişletilir; kod o aralıkta hiç test edilmemiştir	Zarf değişikliğinin gereksinim değişikliği sayılması; değişiklik etki analizi; kodun etkilenen kısmının yeniden doğrulanması
Kayıtsız değişiklik	Uçaktaki dosyanın kimliği SCI'daki kimlikle uyuşmaz	Değişiklik yetkisinin tanımlanması; her PDI değişikliğinin değişiklik kontrolünden geçmesi

Önlem sütunu, kod için zaten bilinen beş ilkeye çıkar: sürüm kontrolü, gözden geçirme, doğrulama, değişiklik yetkisi ve izlenebilirlik. Konfigürasyon verisinde yeni olan ilkeler değil, bunların kodun dışındaki bir dosyaya da aynı ciddiyetle uygulanmasıdır.

Bu bölümden akılda kalması gerekenler

- Konfigürasyon verisi, kod değişmeden davranışı değiştirir; yanlış ya da eksik veri, hatasız kodu yanlış çalıştırır ve kod odaklı denetimlerin hiçbirine takılmaz.
- Davranışı etkileyen her veri aynı rejime girmez: koda gömülü sabit kodla birlikte doğrulanır, PDI onaylı konfigürasyonun ayrı bir ögesidir, kullanıcı tarafından değiştirilebilir veri ancak emniyeti etkileyemediği gösterilmişse serbest bırakılır, havacılık veri tabanları DO-200B ile yönetilir.
- Parametre verisi ögesi, verinin koddan ayrı sürümlenip doğrulanmasına imkân verir. Bunun dört koşulu vardır: kod tüm geçerli veri zarfıyla doğrulanmıştır, geçersiz dosyaya karşı gürbüzdür, veriden doğan bütün davranışı doğrulanabilir ve yaşam döngüsü verisi ayrı yönetime elverişlidir. Koşullar sağlanmıyorsa veri kodla birlikte doğrulanır.
- Her PDI dosyası kendi başına doğrulanır: yapı, değer doğruluğu, ögeler arası tutarlılık ve eksiksizlik. Tablo A-5'in 8. ve 9. hedefleri bunu ister; Seviye A ve B'de bağımsızlık aranır. Dosya, yazılım seviyesini kullandığı koddan alır.
- PDI dosyası, yapı sürümü, uyumlu yazılım kimliği ve bütünlük değeri taşır; yazılım bunları yükleme sırasında ve her açılışta denetler, uyumluluk SCI'da kayıt altına alınır.
- Yazılımın iç denetimi yapı ve aralık hatasını yakalar, "aralıkta ama yanlış" değeri yakalayamaz; değer kaynağı ancak izlenebilirlik ve gözden geçirmeyle doğrulanır.
- Zarfın dışına çıkan yeni veri, veri güncellemesi değil gereksinim değişikliğidir.

23. Havacılık Verileri

Havacılık verisi; seyrüsefer, arazi, engel ve havalimanı haritalama veri tabanları gibi, uçaktaki yazılımdan ayrı üretilen ve dönemsel olarak yenilenen veri kümelerini kapsar. Bu bölüm, verinin kaynağından uçağa uzanan veri zincirini, bu zinciri güvence altına alan DO-200B sürecini ve otorite onayını, DO-178C ile sınırı ve uçaktaki yazılımın veriyi kullanmadan önce yapması gereken denetimleri anlatır.

Havacılık verisi nedir ve neden kritiktir?

Bir uçuş yönetim sistemi (flight management system, FMS), yaklaşma prosedürünü kusursuz bir algoritmayla uçarabilir; prosedürdeki bir noktanın koordinatı yanlışsa uçak yine yanlış yere gider. Kod gereksinimlerine uygundur, test edilmiş ve onaylanmıştır, ama sonuç yanlıştır. Bu yüzden veri pasif bir girdi değil, emniyet zincirinin etkin bir halkasıdır.

Havacılık verisini yazılımdan ayıran üç özellik vardır. Yazılımı geliştiren kuruluşun dışında doğar: kaynağı, devletlerin havacılık bilgi yayını (Aeronautical Information Publication, AIP) gibi resmî yayınları ile ölçüm ve harita kurumlarıdır. Yazılımın onayı yenilenmeden, belirli bir takvimle değişir. Hacmi de insan gözüyle denetlenemeyecek kadar büyüktür: tek bir veri tabanı on binlerce kayıt taşır. En sık karşılaşılan türler şunlardır:

Veri türü	Beslediği işlev	Güncelleme
Seyrüsefer (navigation) veri tabanı: havalimanları, pistler, seyrüsefer yardımcıları, hava yolları, terminal prosedürleri	FMS'te rota ile kalkış, geliş ve yaklaşma prosedürleri	28 günlük sabit dönemlerle
Arazi ve engel veri tabanı	Arazi farkındalık ve uyarı sistemi (terrain awareness and warning system, TAWS), sentetik görüş	Tedarikçinin yayın takvimine göre; engel verisi araziden daha sık değişir
Havalimanı haritalama veri tabanı (aerodrome mapping database)	Yerdeki hareketli harita ekranı; pist ve taksi yolu farkındalığı	Tedarikçinin yayın takvimine göre

Uçağa ya da motora özgü performans tabloları ve kalibrasyon dosyaları bu anlamda havacılık verisi değildir; onları yazılımı geliştiren ya da entegre eden kuruluş üretir ve [22. Konfigürasyon Verisi](#) bölümündeki kurallarla yönetilirler.

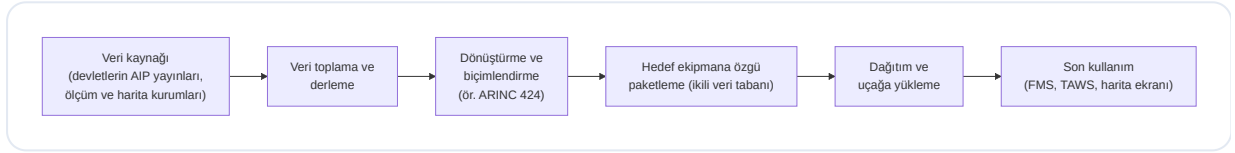
Her veri türü için aynı dört soru sorulur: Veri kimden geliyor ve bize ulaşana kadar kimlerin elinden geçiyor? Güncelleme döngüsü nasıl işliyor? Bütünlük yol boyunca nasıl korunuyor? Yanlış ya da süresi geçmiş veri fark edildiğinde sistem ne yapacak? İlk üç soru veri zincirinin, sonuncusu uçaktaki yazılımın ve sistem gereksinimlerinin konusudur; bölüm de bu sırayı izler.

Veri zinciri ve DO-200B

Yazılım için DO-178C neyse, havacılık verisi için de **DO-200B** (Avrupa'daki karşılığı ED-76A, 2015) benzer bir rolü üstlenir: verinin kendisinin doğruluğunu değil, veriyi işleyen sürecin disiplinini güvence altına alır. Temel fikir şudur: veri, kaynağından uçaktaki ekipmana ulaşana kadar uzun bir **veri zincirinden (data chain)** geçer ve bu zincirin her halkası hata ekleyebilir ya da mevcut bir hatayı fark etmeden iletebilir. DO-200B, zincirdeki her katılımcının kendi payına düşen işleme adımlarını tanımlı, denetlenebilir ve tekrarlanabilir bir süreçle yürütmesini bekler.

Önceki sürüm DO-200A / ED-76'dır; eski kabul belgelerinde ve eski proje dokümanlarında hâlâ karşınıza çıkar. 2024'te DO-200C / ED-76B yayımlanmıştır. Aşağıda anlatılan otorite dokümanları DO-200B / ED-76A'yı esas aldığı için bu bölüm o sürümü izler; yeni bir projede hangi sürümün taahhüt edileceği otoriteyle teyit edilmelidir.

Tipik bir veri zinciri şöyle görünür:



DO-200B'nin süreci, zincirin başındaki resmî yayından sonrasını kapsar: yayındaki bir hata, kusursuz işleyen bir zincir boyunca sadakatle taşınır. Bu yüzden veri tedarikçileri kaynak veriyi makullük denetimlerinden geçirir ve şüpheli gördüklerini kaynağına bildirir.

Veri kalite gereksinimleri

Zincirin her aşamasında verinin sağlaması gereken nitelikler, **veri kalite gereksinimleri (data quality requirements, DQR)** olarak tanımlanır. Bu gereksinimleri veriyi kullanacak uygulamanın sahibi belirler; zincirdeki her halka da bir önceki halkayla bunlar üzerinde yazılı olarak anlaşır. Yazılımdaki karşılığıyla düşünürseniz veri kalite gereksinimleri, veri sürecinin "gereksinim dokümanıdır": süreç onlara göre doğrulanır. Kalite tek boyut değildir:

Kalite boyutu	Anlamı	Örnek soru
Doğruluk (accuracy)	Değerin gerçek değere yakınlığı	Pist eşiği koordinatı kaç metre hatalı olabilir?
Çözünürlük (resolution)	Değerin ifade edildiği hassasiyet	Koordinat kaç ondalık basamakla saklanıyor?
Bütünlük (integrity)	Verinin üretiminden kullanımına kadar bozulmamış ya da yanlışlıkla değiştirilmemiş olma güvencesi	Dönüştürme sırasında bir alan sessizce kırıldı mı?
Güncellik (timeliness)	Verinin kullanıldığı dönemde geçerli olması	Veri tabanı hangi yayın dönemini kapsıyor?

Kalite boyutu	Anlamı	Örnek soru
Tamlık (completeness)	Gerekli tüm kayıtların mevcut olması	Bölgedeki tüm engeller veri tabanında var mı?
İzlenebilirlik (traceability)	Her kaydın kaynağına geri götürülebilmesi	Bu yükseklik değeri hangi resmî yayından geldi?
Biçim (format)	Verinin, alıcının beklediği yapı ve kodlamayla teslim edilmesi	Paketlenen dosya, ekipmanın beklediği biçim sürümünde mi?

DO-200B bütünlük boyutunu doğrudan bir sayı olarak değil, sürecin ne kadar sıkı yürütüleceğini söyleyen bir **güvence seviyesi** üzerinden ifade eder.

Veri süreci güvence seviyesi

Verinin ne kadar sıkı işleneceği, o verinin beslediği işlevin emniyet etkisine bağlıdır. DO-200B bu amaçla üç **veri süreci güvence seviyesi (data process assurance level, DPAL)** tanımlar. Mantık DO-178C'deki yazılım seviyelerine (software level) benzer: hatalı verinin katkıda bulunabileceği arıza durumu (failure condition) ne kadar ağırsa, işleme sürecine uygulanan doğrulama, geçerleme ve kayıt tutma yükümlülükleri o kadar artar.

Seviye	Hatalı verinin katkıda bulunabileceği arıza durumu	ICAO bütünlük sınıfı
DPAL 1	Katastrofik ya da Tehlikeli	Kritik (critical)
DPAL 2	Majör ya da Minör	Temel (essential)
DPAL 3	Emniyet etkisi yok	Rutin (routine)

Tabloyu okurken üç noktaya dikkat etmek gerekir:

- **Seviye verinin türüne değil, kullanımına bağlıdır.** Aynı pist eşiği koordinatı, onu yalnızca durum farkındalığı için arka plan haritasında gösteren uygulamada ve yaklaşma kılavuzluğunda kullanan uygulamada farklı seviye gerektirebilir. Seviye, verinin beslediği işlevin sistem emniyet değerlendirme sürecinden çıkar ve veri kalite gereksinimlerine yazılır.
- **Yazılım seviyeleriyle birebir eşleşme yoktur.** Beş yazılım seviyesine karşılık üç DPAL vardır ve iki ölçek farklı şeyleri ölçer: yazılım seviyesi veriyi okuyan kodun geliştirme titizliğini, DPAL veriyi üreten sürecin titizliğini belirler. Yazılım seviyesi yüksek bir uygulama, veri hatasının etkisi mimariyle sınırlanmışsa daha düşük bir DPAL ile yetinebilir; gerekçesi sistem emniyet değerlendirme sürecinde kayda geçirilir.
- **DO-200C ölçeği genişletmiştir.** Yeni sürüm, 2 ile 3 arasına DPAL 2B adıyla bir ara seviye ekler. Eski ve yeni sürümle üretilmiş verinin aynı zincirde bulunduğu durumda seviyelerin nasıl eşleneceği veri kalite gereksinimlerinde açıkça yazılmalıdır.

Güncellik ve AIRAC döngüsü

Güncellik boyutunun kendine özgü bir mekanizması vardır: seyrüsefer verisi dünya genelinde **AIRAC (Aeronautical Information Regulation and Control)** adı verilen 28 günlük sabit yayın döngüsüyle güncellenir. Değişiklikler önceden duyurulur ve herkes için aynı gün yürürlüğe girer; dönemler de yıl ve sıra numarasıyla anılır (2607, 2026'nın yedinci dönemidir). Bu, veri zincirine sıkı bir takvim baskısı getirir: her döngüde toplama, dönüştürme, doğrulama ve dağıtım adımlarının eksiksiz tamamlanması gerekir. Süreç tanımlı ve otomatik değilse takvim baskısı doğrulama adımlarının atlanmasına yol açar; sahada görülen veri kalite sorunlarının kökeninde çoğu zaman bu vardır.

Otorite onayı ve zincirdeki roller

DO-200B bir endüstri standardıdır; ona ağırlığını veren, otoritelerin veri sürecini kabul ederken onu ölçü almasıdır. Yazılımda onay, belirli bir yazılım sürümünü taşıyan ürüne verilir. Veri tarafında ise her dönem yeni bir ürün çıkar; otoritenin her birini ayrı ayrı onaylaması mümkün değildir. Bu yüzden kabul edilen ürün değil, ürünü üreten **kuruluş ve süreçtir**.

Zincirde dört rol ayırt edilir:

Rol	Tipik olarak kim?	Ne yapar?	Güvencenin dayanağı
Veri kaynağı (data originator)	Devletlerin havacılık bilgi hizmetleri, ölçüm ve harita kurumları	Veriyi üretir ve resmî yayınlarla duyurur	Devletin havacılık bilgi hizmeti kuralları (ICAO Ek 15 çerçevesi); DO-200B zinciri bu yayından sonra başlar
Veri tedarikçisi (data supplier)	Resmî yayınları toplayıp standart biçimde veri tabanı üreten kuruluş	Toplar, derler, ARINC 424 gibi bir değişim biçiminde yayımlar	Tip 1 kabul
Uygulama entegratörü (application integrator)	Çoğunlukla ekipman üreticisi	Veri kalite gereksinimlerini tanımlar, veriyi ekipmanın ikili biçimine paketler, ekipmanla uyumluluğunu gösterir	Tip 2 kabul
Son kullanıcı (end user)	Operatör	Veri tabanını edinir, uçağa yükler ve güncel tutar	İşletme kuralları: verinin kabul görmüş bir tedarikçiden geldiğini ve geçerli döneme ait olduğunu güvence altına almak

İki büyük otorite aynı standarda dayanır ama farklı hukuki araç kullanır:

FAA		EASA
Dayanak	AC 20-153B (2016)	(AB) 2017/373 sayılı Uygulama Tüzüğü; veri hizmetleri sağlayıcılarına ilişkin Part-DAT
Kabulün biçimi	Kabul mektubu (Letter of Acceptance, LOA)	Veri hizmetleri (DAT) sağlayıcı sertifikası; 1 Ocak 2019'dan itibaren gönüllü kabul mektubu uygulamasının yerini almıştır
Tip 1	Veri sürecinin DO-200B hedeflerini karşıladığını tanır; belirli bir uçak sistemi ya da ekipmanla uyumluluk içermez	Veri kalite gereksinimlerini karşılayan veri tabanı üretir; uçaktaki uygulamayla uyumluluk belirlenmemiştir
Tip 2	Sürece ek olarak, teslim edilen verinin belirli aviyonik sistemlerle ve onların amaçlanan işleviyle uyumluluğunu tanır	Sertifikalı uygulama ya da ekipmanla uyumluluğu gösterilmiş veri tabanı üretir; son kullanıcıya doğrudan teslim edebilir

Kabul, bir kez alınıp rafa konan bir belge değildir. Sahibinden, emniyeti etkileyebilecek veri kusurlarını otoriteye bildirmesi, süreç ve veri kalite gereksinimi değişikliklerini kabule sunması, süreci dönemsel iç denetimlerle gözden geçirmesi ve her dağıtımla birlikte kabul durumunu, bilinen sapmaları ve yapılan değişiklikleri bildiren bir yayın beyanı (release statement) vermesi beklenir. Operatör, kendi sorumluluğunu bu beyana dayanarak yerine getirir. EASA tarafında operatörden, sertifikalı sistemlerde kullandığı veri tabanını Tip 2 sertifikalı (ya da eşdeğeri kabul edilen) bir sağlayıcıdan alması beklenir.

Aviyonik üreticisi çoğu zaman Tip 2 rolündedir ve DO-178C projesiyle veri süreci tam burada buluşur. Veri kalite gereksinimlerini üretici yazar; bunların kaynağı, yazılım seviyesini de belirleyen sistem emniyet değerlendirme sürecidir. Veri tabanının ikili biçimi üreticinin tasarım kararıdır ve onu okuyan yazılımın gereksinimlerinde tanımlanır. Tip 2 kabul tipik olarak, hangi veri tabanının hangi ekipman parça numarasıyla uyumlu olduğunu gösteren bir listeye bağlanır; ikili biçimi değiştiren bir yazılım sürümü bu listeyi, paketleme aracını ve veri kalite gereksinimlerini de etkiler. Değişiklik etki analizinde (change impact analysis) en kolay unutulmuş kalem budur.

Uçaktaki yazılımın payı: DO-178C ile sınır

İki standardın sınırı, verinin içeriği ile veriyi kullanan kod arasından geçer. Veri tabanının içeriğinin üretimi DO-200B'ye tabidir ve her dönemde, yazılım onayına dokunmadan yenilenir. Veri tabanını okuyan, ayrıştıran ve kullanan yazılım ise DO-178C'ye tabidir: ikili biçimin tanımı yazılım gereksinimlerinin parçasıdır ve yazılım belirli bir dönemin içeriğiyle değil, biçimin izin verdiği bütün içerikle doğru çalışacak biçimde doğrulanır. Veri tabanı bozuk, eksik ya da süresi geçmiş olduğunda yazılımın ne yapacağı da DO-178C tarafında kalır; bunlar gereksinim olarak yazılır ve gürbüzlük (robustness) testleriyle doğrulanır. Sınırı DO-178C kendisi de çizer: parametre verisi öğelerini tanımlarken konfigürasyon tablolarını ve veri tabanlarını örnek verir, havacılık verisini ise kapsamı dışında bıraktığını açıkça söyler.

Havacılık veri tabanı, davranışı etkileyen diğer veri türleriyle sık karıştırılır. Ayrım, verinin biçiminde değil; kimin ürettiğinde ve güvencenin neye dayandığındadır:

	Parametre verisi ögesi (parameter data item, PDI)	Havacılık veri tabanı	Kullanıcı tarafından değiştirilebilir veri
Kim üretir?	Yazılımı geliştiren ya da entegre eden kuruluş	Veri tedarikçisi ve uygulama entegratörü; kaynağı resmî yayınlardır	Operatör
Ne zaman değişir?	Konfigürasyon değiştiğinde; her yeni dosya yeni bir onaylı konfigürasyondur	Dönemsel olarak; yazılım onayı yenilenmeden	Operatörün ihtiyacına göre; önceden belirlenmiş kısıtlar içinde
Güvencenin dayanağı	DO-178C: her dosya doğrulanır	DO-200B süreci ve bu sürecin otoritece kabulü	Koruma mekanizmasının ve değişiklik kısıtlarının DO-178C kapsamında onaylanması
Uçaktaki yazılımdan beklenen	Yapı, uyumluluk ve aralık denetimi	Kimlik, bütünlük, tamlık ve geçerlilik dönemi denetimi; geçersiz veriye karşı gürbüzlük	Kısıtların dışına çıkılmasını engelleyen koruma

Parametre verisi öğeleri 22. bölümde, kullanıcı tarafından değiştirilebilir veri [19. Kullanıcı Tarafından Değiştirilebilir Yazılım](#) bölümünde anlatılır. Veri tabanı uçağa çoğunlukla sahada, yazılımla aynı veri yükleyiciyle yüklenir; yine de DO-178C'deki anlamıyla sahada yüklenebilir yazılım (field-loadable software) gibi ele alınmaz: içeriği o belgenin kapsamı dışındadır ve kendi güvence zinciriyle yönetilir. Ortak olan yükleme mekanizmasıdır: doğru dosyanın yüklendiğinin ve aktarımda bozulmadığının teyidi [18. Sahada Yüklenebilir Yazılım](#) bölümünde anlatılır. Hangi verinin hangi rejime girdiği planlama aşamasında kararlaştırılır ve yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) belirtilir.

Yükleme zamanı denetimi

Uçaktaki yazılım açısından pratik sonuç şudur: yazılım, yüklenen veri tabanının kimliğini, biçim sürümünü, bütünlüğünü, tamlığını ve geçerlilik dönemini **kullanmadan önce** denetlemelidir. Sürüm ve bütünlük bilgisinin dosyada nasıl taşındığı, 22. bölümdeki PDI örneğiyle büyük ölçüde ortaktır; fark, geçerlilik döneminin de işin içine girmesidir. Basit bir denetim C dilinde şöyle görünebilir. Örnekte dosya 24 baytlık bir başlık, sabit uzunluklu kayıtlardan oluşan bir gövde ve en sonda döngüsel artıklık denetimi (cyclic redundancy check, CRC) değerinden oluşur:

```

#include <stddef.h>
#include <stdint.h>

#define NAV_DB_IMZA          (0x4E415644u) /* dosya türü imzası */
#define NAV_DB_BICIM_SURUMU (4u)          /* kodun beklediği biçim sürümü */
#define NAV_DB_BASLIK_BOYU  (24u)         /* altı adet 32 bit alan */
#define NAV_DB_KAYIT_BOYU   (32u)         /* sabit uzunluklu kayıt */
#define NAV_DB_CRC_BOYU     (4u)          /* dosyanın son dört baytı */

typedef enum {
    NAV_DB_GECERLI = 0,
    NAV_DB_DONEM_DISI, /* sağlam, ama geçerlilik dönemi dışında */
    NAV_DB_BOZUK       /* bütünlük, kimlik ya da tamlık hatası */
} nav_db_durum_t;

typedef struct {
    uint32_t airac_donemi; /* ör. 2607 */
    uint32_t gecerlilik_basi; /* ilk geçerli gün (gün sayacı) */
    uint32_t gecerlilik_sonu; /* son geçerli gün (gün sayacı) */
    uint32_t kayit_sayisi; /* gövdede bulunması gereken kayıt */
} nav_db_kimlik_t;

/* CRC-32 hesabı; başka bir birimde tanımlıdır. */
uint32_t crc32_hesapla(const uint8_t *veri, size_t boyut);

/* Alanlar bayt bayt okunur: dosya doğrudan bir yapıya bindirilirse sonuç
 * derleyicinin yapı dolgusuna ve işlemcinin bayt sırasına bağlı kalır. */
static uint32_t be32_oku(const uint8_t *p)
{
    return ((uint32_t)p[0] << 24) | ((uint32_t)p[1] << 16) |
           ((uint32_t)p[2] << 8) | (uint32_t)p[3];
}

nav_db_durum_t nav_db_denetle(const uint8_t *dosya, size_t boyut,
                             uint32_t bugun, nav_db_kimlik_t *kimlik)
{
    nav_db_durum_t durum = NAV_DB_BOZUK;

    if ((dosya != NULL) && (kimlik != NULL) &&
        (boyut >= ((size_t)NAV_DB_BASLIK_BOYU + NAV_DB_CRC_BOYU))) {
        const size_t korunan = boyut - NAV_DB_CRC_BOYU; /* başlık + gövde */
        const size_t govde = korunan - NAV_DB_BASLIK_BOYU;

        if (crc32_hesapla(dosya, korunan) == be32_oku(&dosya[korunan])) {
            /* Bütünlük doğrulandı; başlık alanları artık okunabilir. */
            kimlik->airac_donemi = be32_oku(&dosya[8]);
            kimlik->gecerlilik_basi = be32_oku(&dosya[12]);
            kimlik->gecerlilik_sonu = be32_oku(&dosya[16]);
            kimlik->kayit_sayisi = be32_oku(&dosya[20]);

            if ((be32_oku(&dosya[0]) == NAV_DB_IMZA) &&
                (be32_oku(&dosya[4]) == NAV_DB_BICIM_SURUMU) &&
                ((govde % NAV_DB_KAYIT_BOYU) == 0u) &&
                ((govde / NAV_DB_KAYIT_BOYU) == kimlik->kayit_sayisi)) {
                if ((bugun >= kimlik->gecerlilik_basi) &&
                    (bugun <= kimlik->gecerlilik_sonu)) {
                    durum = NAV_DB_GECERLI;
                } else {
                    durum = NAV_DB_DONEM_DISI;
                }
            }
        }
    }
}

```

```

    }
  }
}
return durum;
}

```

Örnekte dikkat edilecek dört nokta vardır:

- **CRC başlığı da korur.** Yalnızca gövdeyi kapsayan bir CRC, bozulmuş bir dönem ya da kayıt sayısı alanını sessizce geçirir. Denetim sırası da buna göredir: bütünlüğü doğrulanmamış dosyanın hiçbir alanına güvenilmez.
- **Tamlık ayrıca denetlenir.** Başlıktaki kayıt sayısı gövdenin gerçek boyutuyla karşılaştırılır. Paketleme aracının yarıda kestiği bir dosyanın CRC'si doğru hesaplanmış olabilir; eksik kaydı bu karşılaştırma yakalar.
- **"Bozuk" ile "dönem dışı" aynı şey değildir.** Bütünlük, kimlik ya da tamlık hatası kesin rettir: veri tabanı kullanılmaz ve ona dayanan işlevlerin kullanılamadığı mürettebata (flight crew) bildirilir. Geçerlilik dönemi dışındaki veri tabanı ise bozuk değildir; yalnızca eskidir ya da henüz yürürlüğe girmemiştir. Birçok uçuş yönetim sistemi bu yüzden ardışık iki dönemi birlikte taşır; dönem değiştiğinde etkin veri tabanı da değiştirilir. Dönem dışı veri tabanında yaygın davranış, mürettebatı açıkça uyarmaktır; hangi koşullarda kullanılabileceğini işletme kuralları belirler. Hangi davranışın seçileceği bir sistem gereksinimidir ve iki durumu tek bir "geçersiz" sonucunda birleştiren tasarım bu ayrımı yapamaz.
- **Tarihin kaynağı da gereksinimdir.** Dönem denetimi, güvenilir bir tarih olduğunu varsayar. Tarih geçerli değilse yazılım dönemi kendi başına doğrulayamaz; o durumda veri tabanının dönemi mürettebata gösterilir ve doğrulama ona bırakılır.

Bu denetim veri zincirindeki süreç güvencesinin yerine geçmez. Yalnızca son halkada, paketlemeden sonra dağıtım ve yükleme sırasında oluşabilecek bozulmaları ve yanlış dosya yüklenmesini yakalar; kaynakta ya da dönüştürmede yanlış üretilmiş bir koordinat bütün bu denetimlerden geçer.

Veri işleme araçları

Veri zincirindeki işin büyük bölümü elle değil, araçlarla yapılır: kaynak yayınlardan kayıt ayıklayan çözümleyiciler, koordinat sistemlerini dönüştüren kütüphaneler, ARINC 424 metnini hedef ekipmanın ikili biçimine derleyen paketleyiciler, iki dönem arasındaki farkları çıkaran karşılaştırma araçları. Bu araçların ortak riski, yazılım geliştirme araçlarıyla aynıdır: **araç hatası, çıktıya sessizce hata ekleyebilir** ve bu hata binlerce kaydın arasında insan gözüyle fark edilmez.

Bu yüzden veri işleme araçlarına, yazılım dünyasındaki araç kalifikasyonu (tool qualification) mantığının aynısı uygulanır. Karar iki soruya dayanır:

1. Araç, çıktı veriye hata **ekleyebilir mi** ya da mevcut bir hatayı **gözden kaçırabilir mi**?

2. Aracın çıktısı, araçtan bağımsız bir adımla **doğruluyor mu?**

Birinci soruya "evet", ikinciye "hayır" yanıtı veriliyorsa araca güvence gerekir. Doğrulama yükünü nereye koyduğunuza göre iki temel strateji vardır:

Strateji	Yaklaşım	Bedeli
Çıktıyı doğrula	Aracın her üretimi bağımsız bir kontrolle (ikinci araç, örneklem denetimi, geri dönüştürüp karşılaştırma) doğrulanır	Her AIRAC döngüsünde tekrarlanan işletim maliyeti
Aracı güvence altına al	Aracın gereksinimleri yazılır, doğrulanır, konfigürasyon yönetimine alınır; çıktısına güvenilir	Bir kerelik yüksek geliştirme ve kalifikasyon maliyeti, sonrasında hızlı döngü

AIRAC takvimi 28 günde bir döndüğü için sık görülen yaklaşım ikinci stratejidir: her döngüde on binlerce kaydı elle örneklemek sürdürülebilir değildir. Yazılım tarafındaki karşılığıyla bu, DO-178C'nin araç sınıflandırmasındaki Ölçüt 1 durumudur: çıktısı ürünün parçası olan ve bu yüzden ona hata ekleyebilen araç. DO-178C'de kalifikasyonu gerekli kılan, böyle bir aracın çıktısının ayrıca doğrulanmamasıdır; kalifikasyonun titizliğini yazılım seviyesi belirler. Veri tarafında aynı rolü, aracın ürettiği verinin güvence seviyesi oynar. DO-200B de araç kalifikasyonu için DO-330'a başvurur. DO-330 baştan başka alanlarda da kullanılabilecek biçimde yazılmıştır ve havacılık veri tabanlarını bu alanlar arasında sayar; hangi ölçütle hangi araç kalifikasyon seviyesinin uygulanacağını ise alanın kendi belgesine bırakır. Ölçütler ve araç kalifikasyon seviyeleri [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#) bölümünde ayrıntılı anlatılmıştır ve oradaki mantık buraya doğrudan taşınabilir.

Sahada en çok gözden kaçan iki araç sınıfı şunlardır:

- **Dönüştürücüler ve biçim çevirileri.** Kayıp, çoğu zaman açık bir hata mesajıyla değil, sessiz kırpma ya da yuvarlama ile olur: kaynakta yedi ondalık basamaklı bir koordinatın hedef biçimde beş basamağa inmesi kimseyi uymaz, ama yaklaşma prosedüründe fark yaratabilir. Dönüşümün kalite boyutlarını (özellikle çözünürlük ve tamlık) koruduğu ayrıca gösterilmelidir.
- **Elektronik tablolar ve betikler.** Bir mühendisin "geçici olarak" yazdığı dönüştürme betiği ya da elle düzenlenen tablo da veri zincirinin parçasıdır ve aynı iki soruya tabidir. Konfigürasyon yönetimi dışında yaşayan bu tür araçlar, denetimlerde bulgu üretmeye en yatkın noktalardır.

Son olarak, araç güvencesi ile yükleme zamanı denetimi birbirinin yerine geçmez: CRC, dağıtım sırasındaki bozulmayı yakalar ama aracın baştan yanlış ürettiği (ve CRC'si "doğru" hesaplanmış) veriyi yakalayamaz. İkisi birlikte gerekir.

İlgili endüstri dokümanları

DO-200B süreç disiplinini tanımlar ama verinin **içeriğini ve biçimini** tanımlamaz; o iş, veri türüne göre uzmanlaşmış bir endüstri dokümanı ailesine dağılmıştır. Bu dokümanları tanımak önemlidir, çünkü bir veri projesinde "hangi alan, hangi çözünürlükte, hangi kodlamayla" sorularının yanıtı DO-200B'de değil bu dokümanlarda bulunur.

En sık karşılaşılanlar şunlardır:

Doküman	Yazım tarihindeki güncel sürüm	Alanı	Kısa açıklama
ARINC 424	Güncel eki (supplement) kullanılır	Seyrüsefer veri tabanı	Havalimanları, pistler, seyrüsefer yardımcıları, hava yolları ve terminal prosedürlerinin satır tabanlı kayıt biçimi; FMS veri tabanlarının fiilî kaynak biçimi
DO-201 / ED-77	DO-201C / ED-77B (2025)	Seyrüsefer verisi	Seyrüsefer verisi için içerik ve kalite gereksinimlerini (doğruluk, çözünürlük vb.) uygulama alanına göre tanımlar
DO-276 / ED-98	DO-276C / ED-98C (2015)	Arazi ve engel verisi	Arazi yükseklik modeli ve engel veri kümeleri için kullanıcı gereksinimlerini ve kalite boyutlarını belirler
DO-272 / ED-99	DO-272D / ED-99D (2015)	Havalimanı haritalama	Havalimanı hareket alanı (pist, taksi yolu, apron) veri tabanlarının içerik ve kalite gereksinimleri
DO-291 / ED-119	DO-291C / ED-119C (2015)	Veri değişimi	Arazi, engel ve havalimanı haritalama verisinin taraflar arasında aktarımı için değişim biçimi
ARINC 816	Güncel eki kullanılır	Havalimanı harita ekranı	Havalimanı haritalama verisinin uçtaki ekran uygulamalarına yönelik gömülü biçimi

Bu dokümanların iş bölümünü şöyle okumak yararlıdır: **DO-201, DO-276 ve DO-272** "veride ne olmalı ve ne kalitede olmalı" sorusuna, **ARINC 424, ARINC 816 ve DO-291** "veri hangi biçimde taşınmalı" sorusuna, **DO-200B** ise "bu veri hangi süreçle işlenmeli" sorusuna yanıt verir. Üçü birlikte zinciri kapatır.

Pratik iki not:

- ARINC 424 bir **değişim biçimidir**, uçuşta kullanılan biçim değildir. Her FMS üreticisi 424 kaynağını kendi ikili veri tabanına derler; bu derleme adımı, bir önceki kısımda anlatılan araç güvencesi sorusunun tam merkezindedir.
- Bu dokümanlar da yaşar: kayıt tipleri eklenir, kalite gereksinimleri güncellenir ve sürümler birbirine göre hizalanır. Örneğin DO-201C, DO-200C'deki yeni güvence seviyesini kullanacak biçimde güncellenmiştir; tablodaki sürümler de bu kitabın yazıldığı tarihe aittir. Bir projede hangi dokümanın **hangi sürümüyle** taahhüt edildiği, veri kalite gereksinimlerinde ve sertifikasyon planlarında açıkça yazılmalı, konfigürasyon yönetimi altında izlenmelidir.

Bu bölümden akılda kalması gerekenler

- Yanlış veri, doğru kodu bile yanlış sonuca götürür. Havacılık verisi yazılımın dışında üretilir ve yazılım onayı yenilenmeden dönemsel olarak değişir; güvencesi bu yüzden ayrı bir standarda, DO-200B / ED-76A'ya dayanır.
- DO-200B verinin değil sürecin güvencesidir: veri kalite gereksinimleri neyin bekleneceğini, veri süreci güvence seviyesi (DPAL 1, 2, 3) sürecin ne kadar sıkı yürütüleceğini söyler. Seviye, verinin beslediği işlevin arıza durumu sınıfından çıkar.
- Otorite veri tabanını değil, onu üreten kuruluşu ve süreci kabul eder: FAA'da kabul mektubu, EASA'da veri hizmetleri sağlayıcı sertifikası. Tip 1 yalnız süreci, Tip 2 ayrıca belirli ekipmanla uyumluluğu kapsar; aviyonik üreticisi çoğunlukla Tip 2'dir.
- Veri tabanının içeriği DO-200B'ye, onu okuyan yazılım ve ikili biçimin tanımı DO-178C'ye tabidir. Havacılık veri tabanı ne parametre verisi ögesidir ne de kullanıcı tarafından değiştirilebilir veri.
- Yazılım veri tabanını kullanmadan önce kimliğini, bütünlüğünü, tamlığını ve geçerlilik dönemini denetler; bozuk veri ile dönem dışı veri ayrı sonuçlardır ve ayrı sistem davranışı gerektirir. Bu denetim zincirdeki süreç güvencesinin yerine geçmez.
- Veri işleme araçları yazılım araçlarıyla aynı soruya tabidir: hata ekleyebilen ve çıktısı bağımsız doğrulanmayan araç güvence gerektirir.
- İçerik ve kalite gereksinimleri DO-201, DO-276 ve DO-272'de, taşıma biçimleri ARINC 424, ARINC 816 ve DO-291'de tanımlıdır; hangi sürümün taahhüt edildiği yazılı olmalıdır.

24. Yazılım Yeniden Kullanımı

Yazılım yeniden kullanımı, başka bir proje için geliştirilmiş ya da hazır alınmış bir bileşeni kodu ve kanıtıyla birlikte yeni bir sisteme taşımaktır; ancak bir yerde doğrulanmış olmak, yeni bağlamda kendiliğinden sertifikasyon kredisi sağlamaz. Bu bölüm bağlamın kanıtı neden geçersiz kılabilirdiğini, yeniden kullanılabilir bileşenin nasıl tasarlandığını, önceden geliştirilmiş ve ticari hazır yazılımın boşluk analiziyle nasıl değerlendirildiğini ve ürün servis geçmişinin ne zaman kredi sağladığını anlatır.

Bağlam neden belirleyicidir?

Doğrulama kanıtı koda değil, kodun belirli bir bağlamdaki davranışına aittir: belirli bir işlemcide, belirli bir derleyici ve gerçek zamanlı işletim sistemi (real-time operating system, RTOS) ile, belirli arayüzlerin arkasında, belirli bir girdi aralığı ve yazılım seviyesi (software level) için üretilmiştir. Bileşen başka bir projede kusursuz çalışmış olabilir; ama bu koşullardan biri değiştiğinde kanıtın o koşula dayanan kısmı da geçerliliğini yitirir. Yeniden kullanım kararı bu yüzden üç soruyla başlar: bileşen hangi bağlamda ve hangi varsayımlarla doğrulanmıştı, yeni sistemde bunlardan hangileri değişiyor ve değişenin dokunduğu kanıt nasıl yeniden üretilecek?

Aşağıdaki tablo en sık değişen bağlam boyutlarını ve her birinin sarstığı kanıtı özetler.

Bağlam boyutu	Neden önemlidir?	Yeniden ele alınan kanıt
İşlemci, derleyici ve derleme seçenekleri	Çalıştırılabilir nesne kodu değişir; kayan nokta, hizalama ve eniyileme davranışı farklılaşabilir	Testlerin yeni hedefte yeniden koşulması; yığın (stack) ve en kötü durum yürütme süresi (worst-case execution time, WCET) analizleri; Seviye A'da derleyicinin ürettiği ve kaynak koda doğrudan izlenemeyen ek kodun doğrulanması
RTOS ve sürücüler	Çizelgeleme, kesme ve bellek hizmetlerinin davranışı ve süresi değişir	Uyarlama katmanının doğrulanması; donanım-yazılım entegrasyon testleri
Arayüzler	Birim, ölçek, değer aralığı, güncellenme hızı ve geçersizlik işareti farklı olabilir	Arayüz gereksinimleri; veri ve kontrol bağlaşımı (data and control coupling) analizi; entegrasyon testleri
Zamanlama	Çağrı sıklığı, işlemci yükü ve süre sınırları değişir	WCET ve çizelgelenebilirlik (schedulability) analizi; zamana bağlı test durumları (test case)
Girdi aralığı ve çalışma modları	Yeni sistem, bileşeni daha önce hiç görmediği değerlere ya da modlara itebilir	Normal aralık ve gürbüzlük (robustness) testlerinin kapsadığı aralık; gereksinimlerdeki aralık varsayımları
Yazılım seviyesi	Daha yüksek seviye ek hedef ve daha fazla hedefte bağımsızlık getirir	Eksik hedeflerin kanıtı, ör. karar kapsama ya da değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC); bağımsızlık kayıtları

Bağlam boyutu	Neden önemlidir?	Yeniden ele alınan kanıt
Konfigürasyon	Etkin özellik kümesi ve parametre verisi farklıdır; eski projede çalışan kod yeni projede kullanılmayabilir	Devre dışı bırakılmış kod (deactivated code) analizi; konfigürasyon ve parametre verisinin doğrulanması

Satırların ortak noktası şudur: değişen şey çoğu zaman kod değildir. Kaynak kod aynı kaldığı için "hiçbir şey değişmedi" demek kolaydır; oysa kanıtın dayandığı varsayım değişmiştir ve bu varsayım hiçbir yerde yazılı değilse farkı kimse görmez.

i ARIANE 5, 1996

Ariane 5'in ilk uçuşu, kalkıştan yaklaşık 40 saniye sonra roketin rotasından sapıp parçalanmasıyla sonuçlandı. Soruşturma kurulunun kamuya açık raporuna göre ataletsel referans sisteminin yazılımı büyük ölçüde Ariane 4'ten devralınmıştı. Yeni roketin uçuş profili daha yüksek yatay hız değerleri üretti; bu hıza bağlı bir büyüklük 64 bit kayan noktadan 16 bit işaretli tam sayıya dönüştürülürken taşıdı. İşlenmeyen istisna önce yedek birimi, ardından asıl birimi durdurdu, çünkü ikisinde de aynı yazılım koşuyordu. Taşın hesabın ait olduğu işlev Ariane 5'te kalkıştan sonra gerekli bile değildi. Bu bir havacılık projesi değildir, ama ders aynıdır: kod aynıydı, bağlam (uçuş profili) farklıydı ve "bu değer taşmaz" varsayımı yeni bağlam için yeniden sorgulanmamıştı.

Yeniden kullanılabilir bileşen tasarlamak

Yeniden kullanım çoğu zaman sonradan akla gelir: proje biter, bir sonraki projede "aynı kodu alalım" denir ve o noktada kodun ne kadar projeye özgü varsayımla dolu olduğu ortaya çıkar. Gerçek yeniden kullanılabilirlik, bileşen daha ilk projede yazılırken verilen tasarım kararlarıyla kazanılır. Üç temel ilke öne çıkar: dar ve belgeli arayüzler, platform bağımlılıklarının yalıtılması ve kanıt paketinin bileşenle birlikte taşınabilir olması.

Dar ve belgeli arayüzler

Bir bileşenin arayüzü ne kadar geniş ve örtükse, yeni bir bağlama taşındığında kırılma olasılığı o kadar yüksektir. Yeniden kullanılabilir bir bileşende:

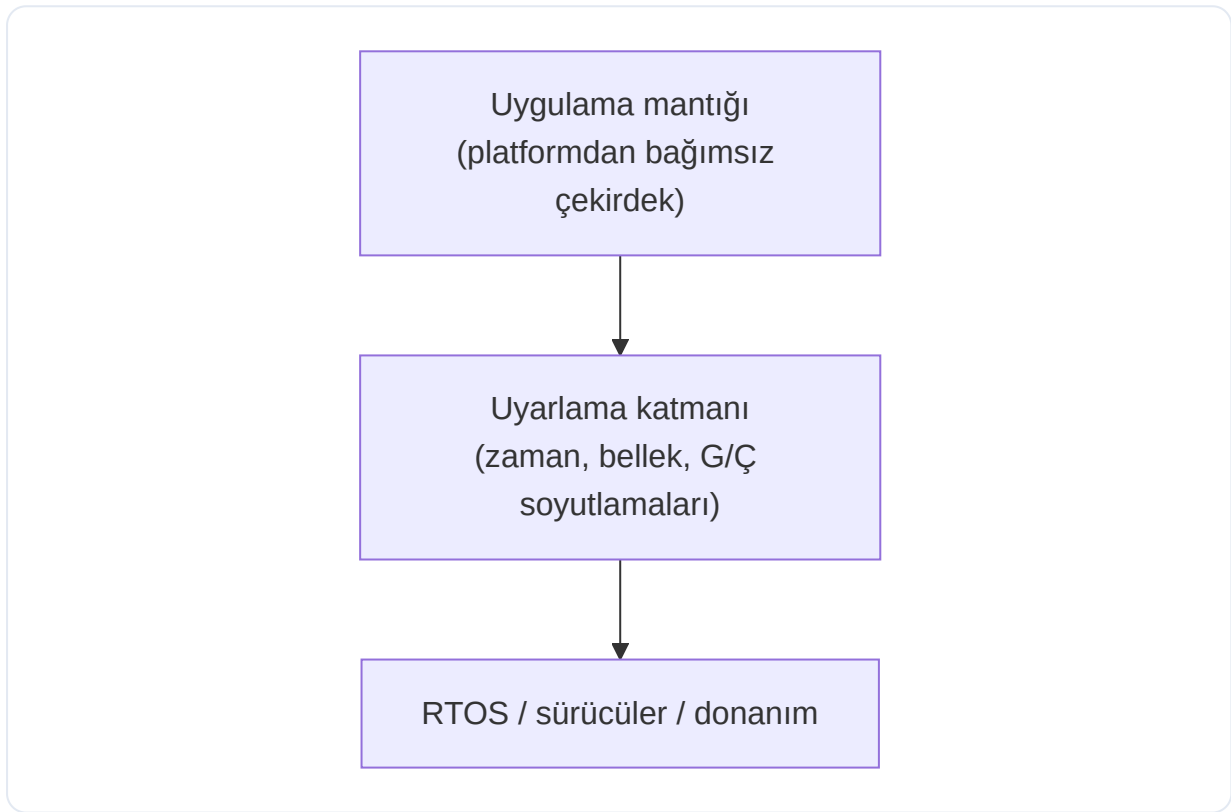
- Arayüz, az sayıda ve açıkça tanımlı fonksiyondan oluşur; bileşenin iç veri yapılarına doğrudan erişim verilmez.
- Global değişken üzerinden örtük veri alışverişi yapılmaz; tüm girdi ve çıktılar fonksiyon parametreleri veya tanımlı mesaj yapıları üzerinden akar.
- Arayüzün davranışı düşük seviyeli gereksinimlerde tanımlıdır: değer aralıkları, birimler, hata durumlarında dönen kodlar, çağrı sıklığı ve zamanlama beklentileri yazılıdır.

- Arayüz sözleşmesine girmeyen davranışlar (örneğin iç tamponların boyutu) belgede "garanti edilmez" olarak işaretlenir; yeni proje bunlara dayanamaz.

Bu disiplinin sertifikasyon açısından karşılığı doğrudan izlenebilirliktir: arayüz gereksinim olarak yazılmışsa, yeni projede "bu bileşenden ne bekliyoruz?" sorusunun cevabı belgededir ve doğrulama kanıtı bu gereksinimlere bağlıdır.

Platform bağımlılıklarının yalıtılması

Bileşenin işlevsel çekirdeği ile donanıma, RTOS'a veya derleyiciye bağımlı kısımları ayrı katmanlarda tutulmalıdır. Yaygın yaklaşım, platforma dokunan her şeyi ince bir uyarlama katmanının (adaptation layer) arkasına almaktır:



C dilinde bu ayırım, çekirdeğin yalnızca soyut bir arayüz başlığına bağımlı olmasıyla sağlanır. Başlık dosyası arayüz sözleşmesini taşır:

```

/* platform_zaman.h - çekirdeğin gördüğü tek zaman arayüzü */
#ifndef PLATFORM_ZAMAN_H
#define PLATFORM_ZAMAN_H

#include <stdint.h>

/* Serbest koşan 32 bit sayaç: en büyük değerden sonra sıfırdan devam eder. */
typedef uint32_t pz_tik_t;

pz_tik_t pz_simdiki_tik(void); /* monotonik sayacın anlık değeri */
pz_tik_t pz_ms_den_tik(uint32_t sure_ms); /* milisaniye -> sayaç birimi */

#endif /* PLATFORM_ZAMAN_H */

```

Çekirdek kod RTOS'un zaman hizmetini değil, bu arayüzü çağırır:

```

/* filtre.c - çekirdek: RTOS API'sini değil, soyut arayüzü çağırır */
#include <stdbool.h>
#include <stdint.h>
#include "platform_zaman.h"

#define FILTRE_PERIYODU_MS (25U)

void filtre_guncelle(int32_t yeni_deger); /* filtre denklemi; burada gösterilmiyor */

static pz_tik_t son_ornek_tik; /* son güncellemenin zaman damgası */

bool filtre_adimi(int32_t yeni_deger)
{
    const pz_tik_t simdi = pz_simdiki_tik();
    bool guncellendi = false;

    /* İşaretsiz çıkarma 2^32'ye göre modülerdir: sayaç taşıp sıfırdan
    devam etse de geçen süre doğru hesaplanır. */
    if ((pz_tik_t)(simdi - son_ornek_tik) >= pz_ms_den_tik(FILTRE_PERIYODU_MS)) {
        filtre_guncelle(yeni_deger);
        son_ornek_tik = simdi;
        guncellendi = true;
    }

    return guncellendi;
}

```

Sayaç tipi bilerek sabit genişlikli seçilmiştir. Genişliği platforma göre değişen bir tipte sayacın taşma noktası, dolayısıyla süre hesabının doğruluğu derleyiciye bağlı kalırdı. "Sayaç 32 bitin tamamını kullanır" koşulu artık arayüz sözleşmesinin parçasıdır ve her yeni uyarlama katmanı bunu sağlamak zorundadır.

Yeni platforma taşımada `platform_zaman.c` gibi uyarlama dosyaları yeniden yazılır ve kendi gereksinimlerine göre doğrulanır. Çekirdeğin gereksinimleri, tasarımı, test durumları ve beklenen sonuçları büyük ölçüde olduğu gibi kalır; **test sonuçları ise kalmaz**. İşlemci ya da derleyici değiştiğinde çekirdeğin çalıştırılabilir nesne kodu da değişir: gereksinim tabanlı testler yeni

hedefte yeniden koşulur, yığın kullanımı ve en kötü durum yürütme süresi gibi derleyiciye ve işlemciye bağlı analizler yenilenir. Yalıtımın kazancı işi sıfırlamak değil, öngörülebilir kılmaktır: neyin yeniden yazılacağı, neyin yalnızca yeniden koşulacağı baştan bellidir.

Kanıt paketinin taşınabilirliği

Aviyonikte yeniden kullanılan şey yalnızca kod değil, koda eşlik eden yaşam döngüsü verisidir. Bileşenle birlikte taşınabilir bir kanıt paketi şunları içerir:

- bileşene ait gereksinimler ve tasarım verisi (sistem belgelerinden ayrıştırılmış, kendi başına anlamlı),
- test durumları, test prosedürleri ve beklenen sonuçlar; mümkünse hedef donanımdan bağımsız çalıştırılabilir biçimde,
- yapısal kapsam analizi sonuçları ve kapsam boşluklarının gerekçeleri,
- bileşenin hangi varsayımlar altında doğrulandığını listeleyen bir kullanım bildirimi: hedef işlemci, derleyici ve seçenekleri, yığın/bellek bütçesi, çağrı bağlamı, yazılım seviyesi,
- açık problem raporları (problem report) ve bilinen sınırlamalar.

Kanıt proje belgelerinin içine dağılmış hâldeyse, bileşen teknik olarak taşınsa bile sertifikasyon kredisi taşınmaz ve doğrulama büyük ölçüde tekrarlanır. Bu nedenle "bileşen sınırında paketlenmiş kanıt", yeniden kullanılabilir tasarımın kod kadar önemli bir parçasıdır.

Bu fikrin otorite rehberindeki karşılığı, FAA'nın AC 20-148 ile tanımladığı yeniden kullanılabilir yazılım bileşeni (reusable software component, RSC) yaklaşımıdır. Rehber üç rol ayırır: bileşeni geliştiren, onu kendi yazılımına katan entegratör (integrator) ve sertifikasyon başvurusunu yapan başvuru sahibi (applicant). Geliştirici, bileşenin kullanıldığı gerçek bir sertifikasyon projesi kapsamında otoriteden kabul alır ve entegratöre şunu açıkça bildirir: hangi hedefler (objective) tümüyle karşılanmıştır, hangileri kısmen karşılanmıştır, hangileri entegratöre kalmıştır ve bileşen hangi varsayım ve kısıtlarla doğrulanmıştır. Sonraki projeler bu krediyi, bileşen değişmediği ve bildirilen kısıtların içinde kalındığı ölçüde kullanır; entegrasyon, hedef donanımdaki doğrulama ve sistem düzeyindeki uyum her projede yeniden gösterilir. Kabul bir ürün onayı değildir; rehber DO-178B döneminde yazılmıştır ve başka otoritelerin bu kabulü nasıl değerlendireceği programa göre değişir. Yaklaşımın hazır RTOS paketlerindeki uygulaması [20. Gerçek Zamanlı İşletim Sistemleri](#) bölümünde anlatılır.

Önceden geliştirilmiş yazılımın kullanımı

Önceden geliştirilmiş yazılım (previously developed software, PDS), yeni projeden önce var olan ve olduğu gibi ya da değiştirilerek alınmak istenen yazılımı tanımlar. Daha önce sertifikaya edilmiş bir uçaktaki uygulama, şirket içi bir kütüphane, başka bir sektör için yazılmış bir protokol yığını

veya ticari hazır yazılım (commercial off-the-shelf, COTS) — hepsi bu şemsiyenin altındadır. Değerlendirmenin özü tek sorudur: **mevcut kanıt, yeni sistemin gerektirdiği güvenceyi hangi ölçüde karşılıyor?**

DO-178C konuyu ek hususlar arasında (12.1) ele alır ve yeniden kullanımın biçimine göre farklı durumlar ayırır: yazılımın değiştirilmesi, yazılımın başka bir hava aracına ya da kurulumuna alınması, uygulama ya da geliştirme ortamının değişmesi (yeni işlemci, derleyici ya da birlikte entegre edilen başka yazılım) ve eldeki yaşam döngüsü verisi yeni kullanımın hedeflerini karşılamadığında geliştirme temel çizgisinin (baseline) yükseltilmesi. Yeni kurulum için ölçüt açıktır: sistem emniyet değerlendirme süreci kurulumu değerlendirir; yazılım seviyesi ve sertifikasyon temeli (certification basis) öncekiyle aynıysa, işlev ve ortam da değişmiyorsa standart ek çaba istemez. Aynı kısım, konfigürasyon yönetimi ve kalite güvencesinin yeniden kullanılan yazılımı da kapsamasını ister. Konfigürasyon yönetimi, bileşenin önceki uygulamadaki ürün ve verisinden yenisine izlenebilirliği kurar ve birden çok uygulamada kullanılan bileşenin problem raporlarıyla değişikliklerinin ortak izlenmesini sağlar. Kalite güvencesi, bileşenin yeni kullanımın yazılım seviyesinin gereklerini karşıladığına ve süreçlerde yapılan değişikliklerin planlara yazıldığına güvence verir. Hangi durum söz konusu olursa olsun, önceden geliştirilmiş yazılımı kullanma niyeti ve boşlukları kapatma stratejisi yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) bildirilir; [SW SOI-1](#) kontrol listesi bunu planlama denetiminin soruları arasında sayar. Yazılıma ait çözülmemiş problem raporlarının etkisi de değerlendirilir.

Yeniden kullanım öteki veride de iz bırakır: tasarım tanımı devralınan bileşenin alındığı temel çizgiyi belirtir, yazılım konfigürasyon indeksi (Software Configuration Index, SCI) ürünündeki önceden geliştirilmiş yazılımı tanımlar, yazılım uygunluk gözden geçirmesi de güncel temel çizginin önceki temel çizgiye ve onaylı değişikliklere izlenebildiğini arar.

Değerlendirmenin adımları

Pratikte PDS değerlendirmesi bir boşluk analizi (gap analysis) olarak yürütülür:

- 1. Yeni bağlamdaki yazılım seviyesi belirlenir.** Sistem emniyet değerlendirme süreci (system safety assessment process), yazılımın yeni sistemdeki yazılım seviyesini belirler; bu seviye eski projedekinden yüksek olabilir.
- 2. Eldeki kanıt envanteri çıkarılır.** Planlar, gereksinimler, tasarım, testler, kapsam sonuçları, konfigürasyon kayıtları ve problem raporları toplanır.
- 3. Boşluklar listelenir.** Hangi hedefler mevcut kanıtlarla karşılanıyor, hangileri eksik? Örneğin eski proje Seviye C ise ve yeni kullanım Seviye B gerektiriyorsa, karar kapsama kanıtı büyük olasılıkla eksiktir. Hedef boşluklarına bağlam boşlukları eklenir: bölümün başındaki tablonun her satırı için eski ve yeni değer yan yana yazılır.
- 4. Kullanılmayan işlevler ele alınır.** Bileşenin yeni projede kullanılmayan işlevleri imajda kalacaksa devre dışı bırakılmış kod olarak yönetilir: gereksinime izlenebilir tutulur, planlarda beyan edilir ve istem dışı etkinleşemeyeceği gösterilir. Ayrıntısı [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#) bölümündedir.

5. **Devralınan problem raporları yeniden değerlendirilir.** Önceki projede açık bırakılmış raporlar yeni sistemin arıza durumlarına göre yeniden sınıflandırılır; eski bağlamda zararsız sayılan bir kısıt yeni bağlamda emniyeti etkileyebilir. Sınıflar [9. Yazılım Doğrulama](#) bölümünde tanımlıdır.
6. **Boşluk kapatma stratejisi seçilir ve PSAC'a yazılır.** Eksik analiz ve testler tamamlanır, gerekiyorsa [25. Tersine Mühendislik](#) bölümünde anlatılan yolla gereksinim ve tasarım verisi üretilir ya da ürün servis geçmişi gibi alternatif güvence yolları önerilir. Strateji otoriteyle erkenden konuşulur; bu görüşmenin nasıl yürütüldüğü [12. Sertifikasyon İrtibatı](#) bölümündedir.
7. **Değişiklik etkisi ayrıca ele alınır.** PDS yeni projede değiştirilecekse, yeniden doğrulama kapsamı değişiklik etki analiziyle (change impact analysis) belirlenir; analizin taradığı eksenler [10. Yazılım Konfigürasyon Yönetimi](#) bölümündedir.

Tipik senaryolar

Senaryo	Ana zorluk	Tipik yaklaşım
Aynı seviyede, değişmeden taşıma	Bağlam farkı (işlemci, derleyici, RTOS, arayüzler)	Değişen bağlam boyutuna göre: testlerin yeni hedefte yeniden koşulması, entegrasyon testleri, zamanlama ve yığın analizinin yenilenmesi
Değiştirilerek taşıma	Değişikliğin değişmeyen kısma etkisi	Değişiklik etki analizi + etkilenen kısmın yeniden doğrulanması
Daha yüksek seviyeye taşıma	Eksik hedefler (ör. MC/DC, bağımsızlık)	Boşluk analizi + eksik doğrulamanın tamamlanması
Önceki DO-178 sürümüyle (ör. DO-178B) onaylanmış yazılım	İlk onayın yeni kurulum için yeterliliği	Servis kaydının ve açık problem raporlarının değerlendirilmesi; seviye yeterliyse ilk onaya dayanma, değilse temel çizginin yükseltilmesi
DO-178 ailesi dışında geliştirilmiş yazılım	Yaşam döngüsü verisi hiç yok ya da farklı biçimde	Tersine mühendislik, yeniden belgeleme, kapsamlı yeniden doğrulama
COTS yazılım	Kaynak koda ve geliştirme sürecine sınırlı erişim	Tedarikçi verisi + ek doğrulama + koruyucu mimari (sarmalayıcı, izleme, bölümlendirme)

Önceki DO-178 sürümleriyle onaylanmış yazılım

Sahada sık karşılaşılan bir yeniden kullanım biçimi, DO-178B ya da daha eski bir sürümle onaylanmış ve yıllardır uçan bir yazılımın bugün değiştirilmesi ya da yeni bir kurulumla alınmasıdır. AC 20-115D ve AMC 20-115D bu yazılımı eski yazılım (legacy software) adıyla ayrıca ele alır. Yaklaşımın özü üç sorudur:

- **Servis kaydı temiz mi?** Yazılımın önceki kurulumlardaki kullanım geçmişi değerlendirilir. Emniyetle ilgili bir servis sorunu, yazılıma bağlanan bir uçuşa elverişlilik direktifi (airworthiness directive) ya da yeni kurulumda emniyeti etkileyebilecek açık problem raporu varsa, bilinen yazılım ve süreç eksiklikleri değişiklikten ya da yeniden kullanımdan önce giderilir.
- **Yazılım seviyesi yeni kurulum için yeterli mi?** Yeterliyse, servis kaydı temizse ve yazılım değişmiyorsa ilk onay yeni kurulumun dayanağı olabilir. Yetersizse geliştirme temel çizgisi yükseltilir; yani eksik hedeflerin kanıtı tamamlanır.
- **Değişiklik hangi süreçle yapılacak?** Rehber, ilk onayda esas alınan DO-178 sürümüyle ve mevcut süreçlerle devam etmeye koşullu olarak izin verir: planlar, süreçler ve yaşam döngüsü ortamı hâlâ sürdürülüyor olmalıdır; model tabanlı geliştirme, nesne yönelimli teknoloji ya da biçimsel yöntemler kullanılacaksa bunları içeren süreçler otoritece önceki bir sertifikasyon projesinde kabul edilmiş olmalıdır. Bu yolda yazılımın DO-178C'yi karşıladığı beyan edilmez; uyum ilk onayın esas aldığı sürüme göre kalır. Koşullar sağlanmıyorsa ya da DO-178C'ye uyum beyan edilecekse geliştirme temel çizgisi DO-178C'ye göre yükseltilir; otoritenin daha önce kabul etmediği bir teknik giriyorsa ilgili teknoloji eki de devreye girer. Hangi yol seçilirse seçilsin yeniden doğrulamanın kapsamını değişiklik etki analizi belirler.

Koşulların tam listesi rehber metnindedir ve projeye nasıl uygulanacağı otoriteyle konuşulur. Akılda tutulacak nokta şudur: ilk onay bir hak değil, bir başlangıç noktasıdır. Belirli bir kurulum ve o günün bilinen problemleri için verilmiştir; aradan geçen yılların servis kaydı da artık değerlendirmenin parçasıdır.

DO-178 dışı ve COTS yazılımın özel durumu

Otomotiv, savunma veya endüstriyel standartlara göre geliştirilmiş yazılım kaliteli olabilir; ancak hedefleri ve kanıt biçimi farklıdır. "ISO 26262 ASIL D'ye göre geliştirildi" gibi bir ifade, DO-178C hedeflerinin otomatik karşılandığı anlamına gelmez; eşleme tek tek hedef düzeyinde yapılmalı ve boşluklar kapatılmalıdır.

DO-178C, COTS yazılıma ayrı bir kolaylık tanımaz (2.5.3): hava aracı sistemine giren COTS yazılım da standardın hedeflerini karşılar, yaşam döngüsü verisindeki eksikler tamamlanır; standart bunun için temel çizginin yükseltilmesine ve ürün servis geçmişine işaret eder. Uygulamada sorun genellikle erişimdir: kaynak kod, geliştirme kayıtları ve problem geçmişi tedarikçinin elindedir. Bu durumda yaygın çözümler şunlardır:

- tedarikçiden sertifikasyon veri paketi temin etmek (bazı RTOS ve kütüphane tedarikçileri bunu ürün olarak sunar; pakete sorulacak sorular [Ek C: Gerçek Zamanlı İşletim Sistemi Seçiminde Sorulacak Sorular](#) içindedir),
- COTS bileşeni emniyet etkisi düşük işlevlerle sınırlamak, yani daha düşük bir yazılım seviyesinde tutmak ve kritik işlevleri kendi geliştirilen kodda bırakmak,
- [21. Yazılım Bölümlemesi](#) bölümünde anlatılan bölümleme ve izleme (monitoring) gibi mimari önlemlerle COTS bileşenin arıza etkisini sınırlamak.

Son iki madde birbirine bağlıdır: bileşene daha düşük bir seviye atanması bir beyanla olmaz; sistem emniyet değerlendirme sürecinin mimariye bakarak verdiği bir karardır ve bileşenin arızasının kritik işlevlere yayılamayacağını gösterilmesine dayanır.

Deneyim şunu gösterir: DO-178 dışı bir yazılımı yüksek seviyeye taşımanın maliyeti, çoğu zaman aynı işlevi baştan geliştirmeye yaklaşıp. Bu nedenle PDS kararı, proje başında dürüst bir boşluk analiziyle verilmeli; "kod hazır, ucuz olur" varsayımıyla plan yapılmamalıdır.

Ürün servis geçmişi

Ürün servis geçmişi (product service history), bir yazılımın gerçek kullanımda geçirdiği sürenin ve bu süre boyunca toplanan problem verisinin, eksik geliştirme kanıtının yerine kısmi güvence olarak öne sürülmesidir. Fikir cazip görünür: "Bu yazılım on yıldır sahada sorunsuz çalışıyor, demek ki güvenilir." Ancak bu argümanın sertifikasyonda kabul görmesi, sanıldığından çok daha sıkı koşullara bağlıdır.

DO-178C servis geçmişini alternatif yöntemler arasında (12.3.4) ele alır: geçmişin yeni kullanımla ilgisi, birikmiş sürenin yeterliliği, servis sırasında bulunan problemlerin toplanıp analiz edilmesi ve PSAC'ta bildirilecek bilgiler ayrı ayrı değerlendirilir. Değerlendirme yalnızca yazılım ekibinin işi de değildir: servis geçmişinin nasıl kullanıldığı ve ne gösterdiği, sistem emniyet değerlendirme süreci dahil sistem süreçlerinin önünden geçer ve otoriteye sunulur. Konunun kamuya açık ayrıntılı bir kaynağı, FAA'nın DO-178B döneminde yayımladığı Software Service History Handbook'tur (DOT/FAA/AR-01/116).

Kredinin dayanması gereken koşullar

Standart, yöntemin kabul edilebilirliğini bir dizi etkene bağlar: servis ortamının ilgililiği, problem raporlamanın etkinliği, yazılımın konfigürasyon yönetimi ile kararlılığı ve olgunluğu, geçmişin uzunluğu ve gerçekleşen hata oranı, değişikliklerin etkisi. Bu bölüm bunları dört koşulda toplar; dördü aynı ağırlıkta değildir. Uygulamada ilk üçü ön koşul gibi işler: biri eksikse geçmişin neyi kanıtladığı söylenemez ve kredi sağlanmaz. Dördüncüsü kredinin ölçüsünü belirler: yetersizse kredi kısmi kalır ve boşluk ek doğrulamayla kapatılır.

- **Benzer kullanım bağlamı.** Geçmişteki kullanım ile yeni kullanım; işlev, arayüzler, çalışma modları ve ortam açısından karşılaştırılabilir olmalıdır. Yer sisteminde çalışmış bir yazılımın geçmişi, uçuş bağlamı için doğrudan kredi sağlamaz. Ortamlar yalnızca kısmen farklıysa standart, farkın hedef ortamda ek doğrulamayla kapatılmasını ister; fark büyüdükçe gereken geçmiş de büyür.
- **Güvenilir problem kayıtları.** Servis dönemi boyunca hataların sistematik olarak raporlandığı, sınıflandırıldığı ve kapatıldığı gösterilebilmelidir. "Hata kaydı yok" ifadesi, kayıt sisteminin hiç işlemediği anlamına da gelebilir; kanıt değeri ancak işleyen bir raporlama süreciyle doğar.

- **Konfigürasyon kararlılığı.** Kredi talep edilen sürüm ile sahada çalışan sürümler arasındaki ilişki konfigürasyon yönetimi kayıtlarıyla izlenebilir olmalıdır. Servis dönemi boyunca sık ve izlenemeyen değişiklik yapılmışsa, hangi geçmişin hangi sürüme ait olduğu belirsizleşir ve argüman çöker.
- **Yeterli ve ilgili süre.** Süre, yazılımın çalışma biçimine uyan bir ölçüyle verilir: uçuş boyunca sürekli çalışan yazılımda uçuş saati, istek üzerine çalışan yazılımda istek sayısı. Bu ölçü filo büyüklüğü ve çalışma profiliyle birlikte okunur. Ne kadarının yeteceğini standart dört şeye bağlar: yeni kullanımın yazılım seviyesi ve sistemin emniyet hedefleri, servis ortamı ile yeni ortam arasındaki farklar, servis geçmişiyle karşılanmak istenen hedefler ve o hedefler için eldeki öteki kanıt; istenen kredi büyüdükçe gösterilmesi gereken geçmiş de büyür. Serviste gözlenen hata oranı (error rate) PSAC'ta öngörülenden yüksek çıkarsa hatalar analiz edilip otoriteyle gözden geçirilir; sürenin uzatılması gerekebilir ya da servis geçmişi bir yol olmaktan çıkabilir. Bu, yazılıma sayısal bir arıza olasılığı atamak demek değildir; olasılık hedefi sistem düzeyindeki arıza durumuna aittir. Nadiren tetiklenen çalışma modları için "toplam saat" tek başına yanıltıcıdır: yazılım yıllarca çalışmış ama kritik mod hiç devreye girmemiş olabilir.

Değerlendirmede yaşanan zorluklar

- Eski kullanım verisi genellikle başka bir kuruluşun elindedir; erişim ve veri kalitesi sözleşmesel engellere takılır.
- Sahadaki hataların hangilerinin yazılım kaynaklı olduğu çoğu zaman net ayrıştırılmamıştır; donanım ve operasyon kaynaklı olaylarla karışır.
- Servis geçmişi yalnızca gözlenen davranış kanıtları; hiç tetiklenmemiş girdi kombinasyonları hakkında bilgi vermez. Bu yüzden yapısal kapsam gibi hedeflerin yerini tam olarak tutamaz.
- Yazılım yeni kullanım için değiştirilecekse, değişen kısım için geçmiş kredi sağlamaz; o kısım olağan yolla doğrulanır.
- Kredi hesabı öznel: "kaç saat yeterlidir?" sorusunun standart bir cevabı yoktur. Yaklaşım PSAC'ta tanımlanır ve sertifikasyon otoritesiyle erken ve açık mutabakat gerekir. PSAC'a yalnızca niyet değil, hesabın kuralları da yazılır: gereken geçmiş miktarı ve gerekçesi, neyin hata sayılacağı, kabul edilebilir hata oranı, geçmişi geçersiz kılacak problem ölçütü ve servis geçmişiyle karşılanacak hedefler.

Bu nedenle pratikte servis geçmişi, tek başına bir güvence yolu olmaktan çok, boşluk analizinde belirli hedefler için **destekleyici** bir argüman olarak kullanılır ve hemen her zaman ek doğrulama faaliyetleriyle birleştirilir. Değerlendirmeye başlarken sorulması gereken soruların derli toplu bir listesi [Ek D: Yazılım Servis Geçmişi Soruları](#) sayfasında verilmiştir; PSAC'ı yazmadan önce bu listeyi dürüst bir ön eleme yapmak, aylar sonra reddedilecek bir argümana yatırım yapmaktan çok daha ucuzdur.

Bu bölümden akılda kalması gerekenler

- Doğrulama kanıtı bağlama aittir: işlemci, derleyici, RTOS, arayüz, zamanlama, girdi aralığı, yazılım seviyesi ya da konfigürasyon değiştiğinde kanıtın o varsayıma dayanan kısmı yeniden üretilir. Daha önce kullanılmış olmak tek başına yeterli değildir.
- Yeniden kullanılabilirlik sonradan eklenmez; dar arayüzler, platform yalıtımı ve bileşen sınırında paketlenmiş kanıtla baştan tasarlanır.
- Platform yalıtımı gereksinimleri, tasarımı ve test durumlarını korur; yeni işlemci ya da derleyicide testler yine yeniden koşulur, yığın ve zamanlama analizleri yenilenir.
- PDS değerlendirmesi bir boşluk analizidir: strateji PSAC'ta bildirilir, kullanılmayan işlevler ve devralınan açık problem raporları analizin parçasıdır.
- Önceki bir DO-178 sürümüyle alınmış onay başlangıç noktasıdır; servis kaydı ve yazılım seviyesinin yeni kurulum için yeterliliği yeniden değerlendirilir.
- DO-178 dışı veya COTS yazılımda boşluğu kapatmanın maliyeti yeniden geliştirmeye yaklaşabilir.
- Servis geçmişinde benzer bağlam, güvenilir problem kayıtları ve konfigürasyon kararlılığı ön koşuldur; süre ve kapsam kredinin ölçüsünü belirler ve hesabın kuralları PSAC'ta otoriteyle kararlaştırılır. Çoğunlukla destekleyici bir argümandır.

25. Tersine Mühendislik

Tersine mühendislik (reverse engineering), mevcut koddan geriye doğru giderek eksik ya da güncelliğini yitirmiş gereksinim ve tasarım verisini yeniden kurma çalışmasıdır. DO-178C hedefleri (objective) bu yolda da aynen geçerlidir; bu bölüm yaklaşımın ne zaman makul olduğunu, nasıl planlanacağını ve koddan çıkarılan gereksinimlerin koddan bağımsız bir referansla nasıl geçerleneceğini anlatır.

Neden yapılır?

Olağan akışta, yani ileri yönde geliştirmede (forward engineering), her veri bir üst soyutlama düzeyindeki veriden üretilir: sistem gereksinimlerinden yüksek seviyeli gereksinimler (high-level requirements), onlardan tasarım ve düşük seviyeli gereksinimler (low-level requirements), onlardan kod. Tersine mühendislikte elde zincirin yalnızca alt halkası vardır ve üst halkalar ondan geriye doğru kurulur. DO-178C'nin sözlüğü terimi geniş tutar: eldeki yazılım verisinden daha üst düzeydeki verinin üretilmesi. Düşük seviyeli gereksinimlerden yüksek seviyeli gereksinim kurmak da, nesne kodundan kaynak kod çıkarmak da bu tanıma girer. Bu bölüm uygulamada en sık karşılaşılan hâli, kaynak koddan gereksinim ve tasarımın kurulmasını ele alır. Buna başvurulacak tipik durumlar şunlardır:

- **Kayıp ya da eskimiş veri.** Uzun ömürlü bir üründe gereksinim ve tasarım dokümanları kaybolmuş, ekip değişmiş ya da yıllar içinde yapılan düzeltmeler dokümanlara işlenmediği için veri kodla uyumsuz hâle gelmiştir.
- **Başka bir süreçle geliştirilmiş yazılım.** Askerî, endüstriyel ya da ticari hazır yazılım (commercial off-the-shelf, COTS) kökenli bir bileşen, DO-178C'nin beklediği yaşam döngüsü verisi olmadan gelir.
- **Daha yüksek yazılım seviyesine taşıma.** Eski projede üretilmemiş ya da yeni yazılım seviyesinin (software level) gerektirdiği ayrıntıda olmayan veri tamamlanmak zorundadır.

Üçü de önceden geliştirilmiş yazılımın (previously developed software, PDS) yeniden kullanımının özel hâlleridir. [24. Yazılım Yeniden Kullanımı](#) bölümünde anlatılan boşluk analizi (gap analysis) hangi verinin eksik olduğunu gösterir; tersine mühendislik bu boşluğu kapatmanın yollarından yalnızca biridir. Bileşeni yeniden geliştirmek, onu koruyucu bir mimariyle çevrelemek ya da ürün servis geçmişi (product service history) kredisi talep etmek ([Ek D: Yazılım Servis Geçmişi Soruları](#)) aynı analizde birlikte tartılır.

Ne zaman makul, ne zaman değil?

Tersine mühendislikte en büyük risk, kodun ne yaptığına bakıp bunun yapılması gereken şey olduğunu varsaymaktır. Bu yüzden çıkarılan bilgi hemen doğru kabul edilmez. Riskin büyüklüğü, elde koddan bağımsız ne kadar bilgi bulunduğuyla bağlıdır:

Eldeki veri	Yeniden kurulacak veri	Ana risk	Değerlendirme
Gereksinim ve tasarım var ama kodla uyumsuz	Güncel gereksinim, tasarım ve iz verisi (trace data)	Her farkta "kod doğrudur" diye karar vermek	Genellikle makul; her fark tek tek problem raporuyla (problem report) çözülür
Kaynak kod ve sistem gereksinimleri var; yazılım gereksinimi ve tasarım yok	Yüksek ve düşük seviyeli gereksinimler, mimari, iz verisi	Gereksinimin kodun yeniden anlatımına dönüşmesi; tasarım gerekçelerinin kaybı	Makul; yazılımın niyetini bilen uzman ve bağımsız referans şarttır
Kaynak kod var; niyeti anlatan belge de bilen kişi de yok	Sistem düzeyindeki niyet dahil bütün zincir	Geçerlemenin (validation) dayanacağı referans yok; koddaki hata gereksinime taşınır	Önce sistem düzeyinde niyet kurulur; kurulamıyorsa yeniden geliştirme daha sağlamdır
Yalnızca çalıştırılabilir nesne kodu (executable object code) var	Kaynak kod dahil her şey	Kod gözden geçirmesi ve kaynak düzeyinde kapsam analizi için güvenilir bir kaynak yoktur	Tanım gereği bu da tersine mühendisliktir, ama uygulamada savunulması en güç hâldir; yeniden geliştirme ya da 24. bölümdeki COTS yaklaşımları düşünülür
Kod bu projede, gereksinim beklenmeden yazılmış	Geriye dönük "tamamlanan" gereksinim ve tasarım	Planlara uyulmadığının üstünün örtülmesi	Bilgi kurtarma değil, süreç sapmasıdır; otoritenin de böyle değerlendirmesi beklenir

Yazılım seviyesi yükseldikçe hem iş hem risk büyür. Seviye A ve B'de bağımsızlık aranan doğrulama (verification) hedefleri vardır; kodu çözümleyip gereksinimi yazan kişi o gereksinimin gözden geçirmesini ve testini de üstlenirse hem bağımsızlık hem de hatayı yakalama şansı kaybolur. Kod, karar kapsama (decision coverage) ya da değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC) gözetilmeden yazılmış olduğundan, bu ölçütleri gereksinim tabanlı testle (requirements-based testing) sağlamak beklenenden zor çıkabilir ve kodda değişiklik gerektirebilir. Seviye D'de yapısal kapsam hedefi bulunmadığı için aynı çalışma çok daha dar kalır. Yaklaşımın en iyi sonuç verdiği yer, küçük, kararlı ve işlevi iyi bilinen bileşenlerdir; büyük, karmaşık ve yorum satırı bile olmayan bir kod tabanında niyeti güvenle kurmak çoğu zaman yeniden yazmaktan pahalıdır.

Dayanak dokümanlar

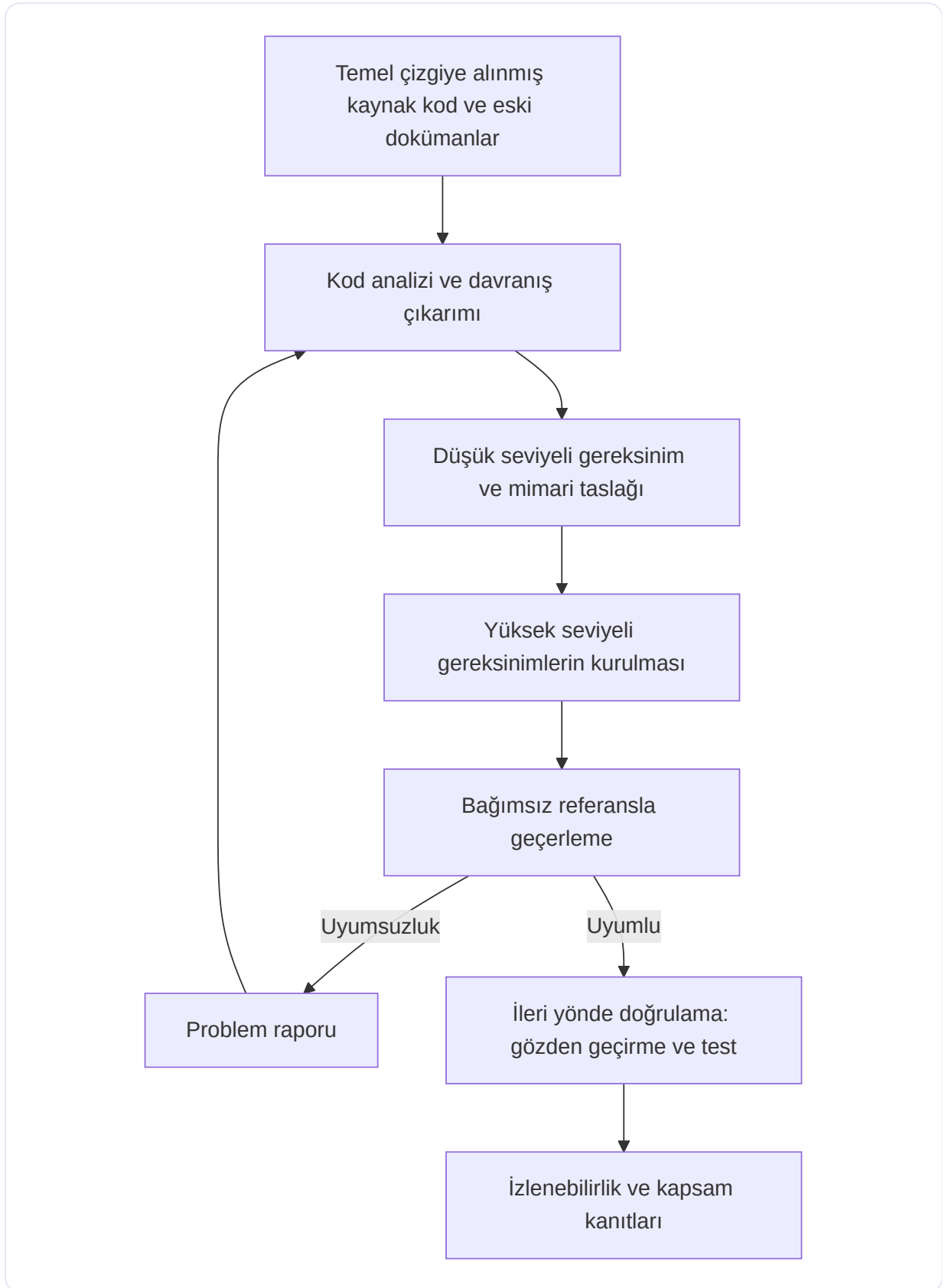
Tersine mühendisliği ayrıntılı biçimde düzenleyen bağlayıcı bir belge yoktur. DO-178C'de terim yalnızca sözlükte ve geliştirme temel çizgisinin (baseline) yükseltilmesini anlatan maddede (12.1.4) geçer. Beklentiler üç kaynaktan izlenebilir:

Doküman	Statüsü	Bu konuda söylediği
DO-178C'nin önceden geliştirilmiş yazılıma ilişkin hükümleri, özellikle madde 12.1.4 (AC 20-115D ve AMC 20-115D ile tanınır)	Kabul edilebilir uyum yöntemi	Bir geliştirme temel çizgisi yükseltilirken eksik ya da yetersiz yaşam döngüsü verisinin tersine mühendislikle yeniden üretilebileceğini kabul eder; veriyi üretmenin tek başına yetmeyebileceğini, doğrulama hedefleri için ek faaliyet gerekebileceğini de belirtir. Amaç yine aynı hedefleri karşılamaktır
CAST-18, "Reverse Engineering in Certification Projects" (2003)	Sertifikasyon otoritelerinin yazılım uzmanlarından oluşan ekibin (Certification Authorities Software Team, CAST) tutum belgesi; resmî politika değildir, DO-178B döneminden kalma tarihî bir başvuru kaynağıdır	Otoritelerin kaygılarını sıralar: yetersiz planlama, yaklaşımın maliyeti kısmak ya da aksayan projeyi kurtarmak için seçilmesi, yazılımın niyetini bilen uzmana erişilememesi, anlaşılması güç kod, soyutlama ve izlenebilirlik (traceability) güçlüğü, geç kalan otorite irtibatı
DOT/FAA/TC-15/27, "Reverse Engineering for Software and Digital Systems" (2016)	FAA'nın desteklediği araştırma raporu; FAA politikası değildir, DO-178B ve DO-254 esas alınarak yazılmıştır	Tersine mühendisliğin ne zaman kabul edilebilir olduğuna dair bir çerçeve önerir ve bunu biri yazılım, biri donanım üzerinde iki vaka çalışmasıyla sınar: veri akışının ve verinin ne zaman konfigürasyon kontrolüne alınacağını plan ve standartlarda açıkça anlatılması; alan uzmanlarının (subject matter expert) gerçekleştirilmiş davranışı amaçlanan kullanım açısından değerlendirmesi ve alan bilgisinin gereksinim ile tasarıma işlenmesi

Üç kaynağın ortak noktası, tersine mühendisliğin ayrı bir uyum yolu olmadığıdır: veri hangi sırayla üretilirse üretilsin, karşılanması gereken hedefler ileri yönde geliştirilmiş bir projedekilerle aynıdır. Ayrıntılı bir rehber bulunmadığı için kabul koşulları proje bazında, sertifikasyon otoritesiyle birlikte belirlenir.

Süreç adım adım

Tersine mühendislik genellikle şu akışla işler:



1. **Girdi dondurulur.** Kaynak kod ve eldeki bütün eski veri tanımlanıp konfigürasyon kontrolüne alınır; analiz bu tanımlı sürüm üzerinde yapılır.

2. **Kod çözümlenir.** Çağrı yapısı, veri ve kontrol akışı, arayüzler ve durumlar çıkarılır. Her çıkarım, dayandığı kod parçasıyla birlikte kaydedilir; bu kayıt ileride iz verisine dönüşür.
3. **Tasarım ve düşük seviyeli gereksinimler yazılır.** Bunlar kodun satır satır yeniden anlatımı olamaz; kodun *neyi* sağladığını, kodu görmeyen birinin aynı işlevi gerçekleştirebileceği biçimde söylemelidir. Koddaki savunmacı denetimler (aralık denetimi, geçersiz veri kolu) de gereksinim olarak yazılır ki gürbüzlük (robustness) testlerinin konusu olsun.
4. **Yüksek seviyeli gereksinimler kurulur** ve sistem gereksinimlerine izlenir. Burada beklenen, düşük seviyeli gereksinimlerin özeti değil, yazılımın dışarıdan gözlenen davranışdır.
5. **Çıkarılan gereksinimler geçerlenir.** Koddan bağımsız referansla karşılaştırılır; her fark problem raporuna dönüşür ve akış ilgili adıma geri döner.
6. **Doğrulama ileri yönde yapılır.** Gözden geçirme, analiz ve testler geçerlenmiş gereksinimleri esas alır; izlenebilirlik ve kapsam kanıtları bu adımda üretilir.

Dikkat edilmesi gereken nokta, akışın sonunda ok yönünün tekrar "ileri" dönmesidir: gereksinimler bir kez kurulduktan sonra doğrulama, sanki proje baştan ileri yönde geliştirilmiş gibi gereksinimlerden koda doğru yapılır.

Ele alınması gereken konular

Tersine mühendislik bir sertifikasyon projesinde "kodu okuduk, dokümanları yazdık" diye özetlenebilecek gayriresmî bir faaliyet olarak kalamaz. Otorite gözünde bu yaklaşım, normal geliştirme akışının tersine işletilmesidir ve tam da bu yüzden dört konunun baştan netleştirilmesini bekler: otoritenin bilgilendirilmesi, sürecin planlarda tanımlanması, kaynak malzemenin konfigürasyon yönetimi (configuration management) altına alınması ve çıkarılan gereksinimlerin geçerlenmesi.

Otorite beklentileri

Sertifikasyon otoriteleri tersine mühendisliği yasaklamaz; ancak "ucuz bir kestirme" olarak kullanılmasına da izin vermez. Deneyim, şu beklentilerin hemen her projede gündeme geldiğini gösterir:

- Yaklaşım, katılım aşaması (Stage of Involvement, SOI) denetimlerinden önce, tercihen planlama aşamasında otoriteyle açıkça konuşulmalıdır ([12. Sertifikasyon İrtibatı](#)). SOI-1'de sürpriz olarak ortaya çıkan bir tersine mühendislik stratejisi neredeyse her zaman bulgu üretir; kapsamı ancak sonraki denetimlerde anlaşılan bir strateji ise onayın kendisini riske atar.
- Sonuçta üretilen yaşam döngüsü verileri, ileri yönde geliştirilmiş bir projeninkiyle **aynı kalitede** olmalıdır: yüksek ve düşük seviyeli gereksinimler, yazılım mimarisi (software architecture), izlenebilirlik ve doğrulama kanıtları eksiksiz beklenir.

- "Kod böyle yazılmış, demek ki gereksinim budur" mantığı kabul görmez. Kodun mevcut davranışı ile sistemin **istenen** davranışı ayrı ayrı ele alınmalı; farklar problem raporuna dönüştürülmelidir.
- Kaynak kodda bulunan ama hiçbir gereksinime bağlanamayan her yapı için karar verilmelidir: eksik bir gereksinim mi, devre dışı bırakılmış kod (deactivated code) mu, yoksa ölü kodu (dead code) da içeren gereksiz kod (extraneous code) mu? Tersine mühendislik bu sınıflandırmadan muafiyet sağlamaz; kararın sonuçları aşağıda "İleri yönde doğrulama" başlığında ele alınır.

Sürecin planlarda tanımlanması

Tersine mühendislik yapılacaksa bu, yazılım planlarında adıyla anılmalı ve süreç adım adım tanımlanmalıdır. Yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC) yaklaşımı ve kapsamını açıkça söyler; yazılım geliştirme planı (Software Development Plan, SDP) verinin hangi sırayla ve hangi geçiş kriterleriyle yeniden kurulacağını, yazılım doğrulama planı (Software Verification Plan, SVP) çıkarılan verinin nasıl gözden geçirilip doğrulanacağını tanımlar ([5. Yazılım Planlama](#)). DO-178C'nin bu konuda açıkça istediği PSAC tarafıdır: önceden geliştirilmiş yazılım kullanma niyeti ve temel çizgi yükseltirken izlenecek uyum stratejisi bu planda belirtilir. Asgari olarak planlarda şunlar yer almalıdır:

Plan içeriği	Cevaplaması gereken soru
Kapsam	Hangi bileşenler tersine mühendisliğe tabi, hangileri yeniden yazılacak?
Girdi verisi	Hangi kaynak malzeme (kod, eski dokümanlar, test kayıtları) kullanılacak?
Üretilcek veri	Hangi yaşam döngüsü verileri, hangi sırayla yeniden kurulacak?
Geçiş kriterleri	Bir adım ne zaman "tamamlandı" sayılır, geri dönüşler nasıl yönetilir, hangi veri ne zaman konfigürasyon kontrolüne girer?
Roller	Yazılımın niyetini bilen alan uzmanları kimlerdir, hangi adımda ne yaparlar?
Doğrulama ve geçerleme	Çıkarılan gereksinimler kime, hangi yöntemle onaylatılacak?
Araçlar	Statik analiz veya kod anlama araçları kullanılacaksa araç kalifikasyonu (tool qualification) gerekiyor mu?

Son satırın cevabı çoğu zaman "hayır"dır: kod anlama aracının çıktısı (çağrı çizgesi, veri akış özeti) gözden geçirmeye ayrıca doğrulanıyorsa kalifikasyon gerekmez. Aracın çıktısına doğrulanmadan güvenilecekse durum değişir; ölçütler [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#) bölümündedir. Planlama denetimine hazırlanırken [SW SOI-1](#) kontrol listesi, tersine mühendislik kapsamındaki bileşenler için de aynen uygulanır.

Kaynak malzemenin konfigürasyon kontrolü

Tersine mühendisliğin girdisi olan kod ve eski dokümanlar çoğu zaman dağınıktır: farklı sürümler, yerel kopyalar, hangi çalıştırılabilir nesne koduna karşılık geldiği belirsiz kaynak ağaçları. Bu malzeme konfigürasyon yönetimi altına alınmadan analiz başlarsa, üç ay sonra "hangi kodu tersine mühendislikten geçirdik?" sorusuna cevap verilemez.

- Analize başlamadan önce kaynak kodun **tek ve tanımlı bir temel çizgisi** oluşturulmalı; sahadaki çalıştırılabilir nesne kodu ile bu kaynağın eşleştiği, örneğin kayıtlı derleme ortamında yeniden üretip karşılaştırarak gösterilebilmelidir. Eşleşme gösterilemiyorsa sahadaki ürünün geçmişi bu kaynak için kanıt değeri taşımaz.
- Eski dokümanlar, test kayıtları ve ürün servis geçmişi de aynı temel çizgiye referansla saklanmalıdır; bunlar "güvenilir gerçek" değil, **girdi verisi** statüsündedir.
- Önceki projeden devralınan açık problem raporları da girdidir. DO-178C, önceden geliştirilmiş yazılımın çözülmemiş problem raporlarının etkisinin değerlendirilmesini ve önceki uygulamanın ürünü ile verisinden yeni uygulamaya izlenebilirlik kurulmasını ister.
- Analiz sırasında kodda düzeltme ihtiyacı doğarsa değişiklik, normal problem raporu ve değişiklik kontrol mekanizmasından geçmelidir; "nasılsa her şeyi yeniden yazıyoruz" rahatlığıyla kontrolsüz düzenleme yapılmamalıdır.

Temel çizgi, değişiklik kontrolü ve problem raporlamanın işleyişi [10. Yazılım Konfigürasyon Yönetimi](#) bölümünde anlatılır; tersine mühendislikte fark, bu mekanizmanın ilk çıkarım yapılmadan önce kurulmuş olması gerektiğidir.

Çıkarılan gereksinimlerin geçerlenmesi

Koddan çıkarılan bir gereksinim, tanımı gereği koddan türetilmiştir; bu yüzden kodla uyumu neredeyse otomatiktir ama **doğruluğu** hakkında hiçbir şey söylemez. Kodun içindeki bir hata, aynı hatayı tarif eden bir gereksinime dönüşür ve doğrulama bu döngüyü kırmadan yalnızca kendi kendini onaylar. Döngüyü kırmak için:

- Çıkarılan yüksek seviyeli gereksinimler, koddan **bağımsız bir referansla** geçerlenmelidir: sistem gereksinimleri, sistem emniyet değerlendirme sürecinin (system safety assessment process) çıktıları, arayüz kontrol dokümanları (interface control document), alan uzmanlarının bilgisi ve varsa saha deneyimi.
- Gereksinim gözden geçirmelerine kodu yazan veya analiz eden kişiden **başka**, sistemin işlevini bilen bir uzman katılmalıdır; aksi hâlde gözden geçirme, analiz notlarının ikinci kez okunmasından ibaret kalır.
- Sistem gereksinimlerine bağlanamayan çıkarımlar, türetilmiş gereksinim (derived requirement) olarak işaretlenip sistem emniyet değerlendirme süreci dahil sistem süreçlerine geri beslenmelidir; tersine mühendislik bu yükümlülüğü ortadan kaldırmaz (kavram için bkz. [6. Yazılım Gereksinimleri](#)).

- Kod davranışı ile bağımsız referans çeliştiğinde varsayılan kabul, **kodun hatalı olabileceğidir**; fark bir problem raporuyla kayda geçirilir ve mühendislik kararı gerekçesiyle birlikte belgelenir.

Bir örnek: koddaki hatanın gereksinime taşınması

Aşağıdaki kurgusal işlev, radyo irtifası bir eşiğe indiğinde alçak irtifa uyarısını etkinleştiriyor:

```
#define UYARI_ESIGI_FT (100)

bool irtifa_uyarisi_etkin(int32_t radyo_irtifa_ft, bool irtifa_gecerli)
{
    bool etkin = false;

    if (irtifa_gecerli && (radyo_irtifa_ft < UYARI_ESIGI_FT))
    {
        etkin = true;
    }

    return etkin;
}
```

Kodu çözümleyen mühendis buradan iki ifade çıkarır. Aynı işlev için sistem gereksinimleri ise başka bir şey söylemektedir:

Koddan çıkarılan (kodu tekrar eden) ifade	Bağımsız referanstaki niyet	Sonuç
"Radyo irtifası geçerli ve 100 ft'ten küçükse uyarı etkinleştirilir."	Uyarı, irtifa 100 ft'e indiğinde ve altında verilir	Kod sınır değerde hatalıdır; problem raporu açılır, kod düzeltilir ya da sistem gereksinimi gerekçesiyle güncellenir
"Radyo irtifası geçerli değilse uyarı etkinleştirilmez."	İrtifa verisi geçersizken uyarının kullanılmadığı mürettebata bildirilir	İşlev kodda hiç yoktur; problem raporu açılır, eksik gereksinim ve kod eklenir

İlk satır yanlış gerçekleştirilmiş bir işlevi, ikinci satır hiç gerçekleştirilmemiş bir işlevi gösterir. İkincisi daha sinsidir: kod ne kadar dikkatle okunursa okunsun, orada olmayan bir davranış koddan çıkarılamaz; eksikliği yalnızca bağımsız referans gösterir.

Geçerleme atlanıp testler soldaki ifadelerden yazılıysaydı üç test durumu yeterdi: geçerli 99 ft, geçerli 100 ft ve geçersiz 99 ft. Üçü de geçer; üstelik bu üç durum, tek karardaki iki koşul için MC/DC'yi de sağlar. Elde eksiksiz izlenebilirlik, geçen testler ve tam yapısal kapsam olur; yazılım ise iki noktada sistemin istediğini yapmıyordur. Yapısal kapsam, kodda olanın çalıştırıldığını gösterir; olması gerekenin kodda bulunduğunu göstermez.

İleri yönde doğrulama

Gereksinimler geçerlendikten sonra gözden geçirme, analiz ve testler ileri yönde geliştirilmiş bir projedekilerle aynıdır ([9. Yazılım Doğrulama](#)). Tersine mühendisliğe özgü olan, şu üç kuraldır:

- **Testler koddan değil, geçerlenmiş gereksinimden yazılır.** Beklenen sonucu kodu okuyarak ya da çalıştırarak belirleyen test, yukarıdaki örnekteki gibi kodun kendisiyle tutarlı olduğunu kanıtlar. Sınır değerler ve gürbüzlük durumları gereksinimden seçilir. Testleri, kodu çözümleyip gereksinimi yazan kişiden başka birinin hazırlaması döngüyü kırmanın en ucuz yoludur.
- **Eski testler de girdi verisidir.** Eldeki test prosedürleri yeniden kullanılabilir; ancak her biri yeni gereksinimlere izlenmeli ve beklenen sonuçları gözden geçirilmelidir. İzlenemeyen test ya eksik bir gereksinime ya da gereksiz bir teste işaret eder.
- **Düşük seviyeli gereksinim sözde kod olamaz.** Gereksinim koda fazla yakınsa ondan türetilen test yalnızca gerçekleştirimi çalıştırır; yanlış, eksik ya da istenmeyen işlevi yakalama gücü kalmaz. Gözden geçirme kontrol listesine "bu gereksinim, kodu görmeden yazılabilir miydi?" sorusu eklenmelidir.

Yapısal kapsam analizi (structural coverage analysis) burada ayrı bir dikkat ister. İleri yönde geliştirmede gereksinime izlenemeyen kod, kapsam boşluğu olarak kendini gösterir. Tersine mühendislikte ise her kod parçası için bir ifade yazıldığından, gereksiz kod da "gereksinimi olan" koda dönüşüp gözden kaçabilir. Sınıflandırma bu yüzden kapsam analizine bırakılmaz; geçerleme sırasında, sistem gereksinimlerine ya da gerekçeli bir türetilmiş gereksinime bağlanamayan her yapı için yapılır:

Bulgu	Karar	Sonuç
Kod gerekli bir davranışı gerçekleştiriyor ama karşılığı olan gereksinim yok	Eksik gereksinim	Gereksinim yazılır, bağımsız referansla geçerlenir, testi eklenir
Kod bir gereksinime izlenebiliyor ama tasarım gereği bu konfigürasyonda çalışmıyor (ör. başka bir uçak tipine ait seçenek)	Devre dışı bırakılmış kod	Planlarda beyan edilir; istem dışı etkinleşmeyi (inadvertent enabling) önleyen mekanizma doğrulanır. Kod başka bir onaylı konfigürasyonda çalışıyorsa kapsamı o konfigürasyon kurularak yapılan testlerle sağlanır
Kod hiçbir gereksinime izlenemiyor; ölü kod bunun hiç çalıştırılmayan özel hâlidir	Gereksiz kod	Kaldırılır

Kapsam analizinde yine de boşluk çıkarsa dördüncü olasılık, gereksinim tabanlı testlerin eksik olmasıdır. Tanımlar ve her kararın gerektirdiği kanıt [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#) bölümündedir. Kod kaldırmak, sahadaki sürümden farklı bir kod üretmek demektir: değişiklik problem raporu ve değişiklik etki analizinden (change impact analysis) geçer; ürün servis geçmişine dayanan bir argüman varsa ona etkisi de ayrıca değerlendirilir.

Bu bölümden akılda kalması gerekenler

- Tersine mühendislik bilgi kurtarma işidir, kestirme yol değildir: DO-178C hedefleri aynen geçerlidir ve üretilen veri ileri yönde geliştirilmiş bir projeninkiyle aynı kalitede olmalıdır.
- Yaklaşımın makul olup olmadığını, elde koddan bağımsız ne kadar bilgi bulunduğu belirler; niyeti anlatan bir referans ya da alan uzmanı yoksa çıkarılan gereksinimi geçerleyecek dayanak kalmaz.
- Yaklaşım planlarda açıkça tanımlanmalı ve otoriteyle erken, tercihen SOI-1 öncesinde konuşulmalıdır.
- Kaynak kod ve eski dokümanlar analiz başlamadan konfigürasyon yönetimi altına alınmalı; girdi malzemesi "güvenilir gerçek" değil veri olarak ele alınmalıdır.
- Koddan çıkarılan gereksinimler koddan bağımsız bir referansla geçerlenmelidir; aksi hâlde koddaki hata gereksinime taşınır ve doğrulama kendi kendini onaylar.
- Testler koddan değil geçerlenmiş gereksinimden yazılır; düşük seviyeli gereksinim kodun yeniden anlatımı olamaz.
- Gereksinime bağlanamayan her kod için karar verilir: eksik gereksinim yazılır, devre dışı bırakılmış kodun mekanizması doğrulanır, gereksiz kod (ölü kod dahil) kaldırılır.

26. Yazılım Yaşam Döngüsü Faaliyetlerinde Dış Kaynak Kullanımı

Dış kaynak kullanımı (outsourcing), yazılım yaşam döngüsü faaliyetlerinin bir kısmının tedarikçi (supplier) tarafından yürütülmesidir; iş devredilir, sertifikasyon sorumluluğu devredilmez. Bu bölüm iş bölümünü, tedarikçi gözetiminin (supplier oversight) planlarda nasıl tanımlandığını, iki kuruluş arasındaki sınırdaki sahipsiz kalmaya yatkın sertifikasyon konularını ve tedarikçi verisinin projeye nasıl alındığını anlatır.

Önce roller ve terimler. Sertifikasyon otoritesi karşısında sorumlu taraf **başvuru sahibidir** (applicant). Ticari zincirde işi dışarı veren kuruluş ise her zaman başvuru sahibi değildir: bir ekipman üreticisi uçak üreticisine göre tedarikçi, kendi alt yüklenicisine göre işi veren taraftır. Bu bölümde işi dışarı veren kuruluşa kısaca **ana yüklenici** denecek; sorumluluk zinciri onun üzerinden başvuru sahibine uzanır. **Gözetim**, tedarikçinin süreç ve çıktılarının proje boyunca izlenmesidir; **denetim** (audit) bu gözetimin tekil, kayıtlı bir faaliyetidir. **İş ürünü** ile tedarikçinin teslim ettiği yazılım yaşam döngüsü verisi (software life cycle data) ve kod kastedilir.

Kapsam dışı bir durum da not edilmeli: ticari hazır yazılımın (commercial off-the-shelf, COTS), örneğin hazır bir işletim sisteminin ya da kütüphanenin satın alınması farklı bir ilişkidir, çünkü ürün proje başlamadan önce vardır ve geliştirme süreci artık yönlendirilemez. DO-178C'nin tanımı da aynı çizgiyi çeker: belirli bir uygulama için sözleşmeyle geliştirilen yazılım COTS sayılmaz, bu bölümün konusudur. COTS ise [24. Yazılım Yeniden Kullanımı](#), [20. Gerçek Zamanlı İşletim Sistemleri](#) ve [Ek C: Gerçek Zamanlı İşletim Sistemi Seçiminde Sorulacak Sorular](#) içinde ele alınır.

Neden dikkat gerekir?

Tedarikçi iyi iş çıkarsa bile ana yüklenici, bu işin sertifikasyon açısından yeterli olduğunu göstermek zorundadır. Yani dış kaynak, sorumluluğu azaltmaz; sadece iş dağıtımını değiştirir.

DO-178C başvuru sahibinin iç örgütlenmesini ve tedarikçileriyle ticari ilişkisini kapsamı dışında tutar; işi kimin yaptığıyla değil, hedeflerin (objective) karşılandığını gösteren kanıtla ilgilenir. Sorumluluk konusunda ise açıktır: standart, yaşam döngüsü süreçlerinde ya da bu süreçlerin çıktılarında payı olan her tedarikçi için de geçerlidir ve bütün tedarikçilerin gözetiminden başvuru sahibi sorumludur. Tedarikçinin ürettiği veri bu yüzden aynı hedeflere, aynı yazılım seviyesine (software level) ve aynı onaylı plan ve standartlara tabidir. "Bu kısmı tedarikçi yaptı" cümlesi, kanıt zincirinde bir boşluğu açıklamaz.

Otorite beklentisi nerede yazılır?

Tedarikçi gözetimi için ayrı bir standart yoktur; beklenti DO-178C'nin içine dağılmıştır ve standart bu konuyu önceki sürümüne göre daha açık hâle getirmiştir. Gözetimin nasıl yapılacağı planlama sürecinde belirlenir ve planlara yazılır. Konfigürasyon yönetimi (configuration management) faaliyetleri tedarikçi yürüten işe de uygulanır; kalite güvencesi (quality assurance) süreci tedarikçi süreç ve çıktıların onaylı plan ve standartlara uyduğuna güvence verir. Yazılım başarı özeti (Software Accomplishment Summary, SAS) bu uyumun nasıl sağlandığını ayrı bir başlıkta anlatır.

FAA AC 20-115D ve EASA AMC 20-115D, DO-178C'yi kabul edilebilir uyum yöntemi olarak tanıdığından bu beklentiler otorite karşısında da geçerlidir. FAA'nın yazılım onay yönergesi Order 8110.49'da bir dönem tedarikçi gözetimine ayrılmış bir bölüm vardı; bu bölüm sonradan yönergeden çıkarıldı ve güncel sürüm olan Order 8110.49A'da (2018) yer almıyor. Eski proje dokümanlarında o bölüme atıf görülürse, güncel dayanağın DO-178C'nin kendisi ve projenin otoriteyle mutabık kalınan planları olduğu bilinmelidir.

Dış kaynak kullanımının nedenleri

Aviyonik projelerinde dış kaynak kararı genellikle üç ana gerekçeye dayanır: maliyet, kapasite ve uzmanlık. Bu gerekçelerin her biri kendi başına makuldür; sorun, çoğu zaman gerekçenin arkasındaki varsayımların sorgulanmadan kabul edilmesidir.

Maliyet. En sık dile getirilen gerekçe, saat ücreti daha düşük bir tedarikçiyle toplam geliştirme maliyetini düşürmektir. Ancak emniyet-kritik yazılımda maliyetin büyük kısmı kod yazmaktan değil; gereksinim (requirement) geliştirme, doğrulama, gözden geçirme ve sertifikasyon kanıtı üretiminden gelir. Saat ücreti üzerinden yapılan karşılaştırma, bu faaliyetlerin tedarikçi tarafında ne kadar verimli yürütüleceğini hesaba katmazsa yanıltıcı olur.

Kapasite. Proje takvimi sıkıştığında veya aynı anda birden fazla program yürütüldüğünde, iç ekip yetmez ve iş dışarıya taşınır. Bu gerekçe kısa vadede geçerli olsa da, tedarikçiye aktarılan işin tanımlanması, aktarılması ve geri geldiğinde projeye alınması da kapasite tüketir. "Dışarı verdik, yükümüz azaldı" varsayımı, koordinasyon yükü ölçülmeden doğrulanamaz.

Uzmanlık. Bazı alanlarda — örneğin belirli bir gerçek zamanlı işletim sistemi (real-time operating system, RTOS) entegrasyonu, araç kalifikasyonu (tool qualification) veya model tabanlı geliştirme (model-based development) — tedarikçinin birikimi iç ekipten fazla olabilir. Bu, dış kaynak gerekçelerinin en sağlamıdır; ancak uzmanlığın gerçekten var olduğu, referans proje ve sertifikasyon geçmişiyle doğrulanmalıdır.

Bu gerekçelerin gerçekleşmesini engelleyen **gizli maliyetler** (hidden costs) çoğu zaman sözleşme aşamasında görünmez:

Gizli maliyet	Nerede ortaya çıkar?
Beklenti aktarımı	Standartların, şablonların ve süreçlerin tedarikçiye öğretilmesi
Gözden geçirme yükü	Tedarikçi çıktılarının ana yüklenici tarafından gözden geçirilmesi
Yeniden çalışma	Kabul kriterlerini karşılamayan iş ürünlerinin düzeltme döngüleri
Koordinasyon	Toplantılar, soru-cevap trafiği, değişiklik bildirimleri
Projeye alma	Tedarikçi çıktısının ana konfigürasyon yönetimine ve izlenebilirlik (traceability) zincirine alınması
Gözetim	Tedarikçi süreçlerinin yerinde veya uzaktan denetlenmesi, bulguların takibi

Deneyim şunu gösterir: Dış kaynak kararının başarısı, saat ücreti farkından çok, bu gizli maliyetlerin baştan öngörülüp bütçelenmesine bağlıdır. Gizli maliyetler bütçelenmediğinde, ya proje takvimi kayar ya da — daha kötüsü — gözden geçirme ve kabul faaliyetleri kısaltılarak kanıt kalitesinden ödün verilir.

İş bölümü: ne dışarı verilir, ne içeride kalır?

Her faaliyet dışarı verilmeye aynı ölçüde uygun değildir. Kaba ölçüt şudur: bir faaliyetin girdisi ne kadar yazılı ve kararlıysa, çıktısı da ne kadar nesnel bir ölçütle kabul edilebiliyorsa, o faaliyet o kadar rahat devredilir. Sistem bağlamı, hedef donanım ya da otoriteyle mutabakat gerektiren kararlar ise içeride kalır. Aşağıdaki tablo tipik bir iş bölümünü gösterir; satırlar kural değil, karar verirken sorulacak soruların özetidir.

Faaliyet	Dışarı verilmeye uygunluk	Tedarikçinin teslim ettiği	Ana yüklenicide kalan rol ve kanıt
Yüksek seviyeli gereksinimler (high-level requirements)	Düşük: sistem bilgisi ve sistem süreçleriyle sürekli geri bildirim ister	Gereksinim taslağı, sistem gereksinimlerine izlenebilirlik	Gereksinimlerin sahipliği; türetilmiş gereksinimlerin (derived requirement) sistem emniyet değerlendirme süreci dahil sistem süreçlerine iletilmesi; gözden geçirme kaydı
Tasarım ve düşük seviyeli gereksinimler	Orta: mimari kısıtlar ve arayüzler yazılı ve kararlıysa	Tasarım tanımı, düşük seviyeli gereksinimler, iz verisi (trace data)	Arayüz ve mimari uyumunun gözden geçirilmesi; yeni türetilmiş gereksinimlerin değerlendirilmesi
Kodlama	Yüksek: girdi gereksinimler olgun, kodlama standardı ortaksa	Kaynak kod, kod gözden geçirme kayıtları	Örneklemler kod gözden geçirmesi; derlemenin kendi ortamında yinelenmesi

Faaliyet	Dışarı verilmeye uygunluk	Tedarikçinin teslim ettiği	Ana yüklenicide kalan rol ve kanıt
Gereksinim tabanlı test (requirements-based testing)	Yüksek: gereksinimler kararlı, test ortamı iki tarafta eşdeğerse	Test durumları, test prosedürleri, sonuçlar, gereksinim-test izlenebilirliği	Testin gereksinimi gerçekten sınıadığının örneklemleri gözden geçirmesi; seçilmiş testlerin yeniden koşulması
Yapısal kapsam analizi (structural coverage analysis)	Orta: ölçüm devredilir, kapsanmayan kodun çözümlenmesi gereksinim ve tasarım bilgisi ister	Kapsam raporu, her boşluk için gerekçe önerisi	Her kapsam boşluğunun nedeninin ve çözümünün onaylanması
Donanım-yazılım entegrasyonu	Düşük: hedef donanıma ve sistem test ortamına erişim gerekir	Entegrasyon desteği, sorun çözümü	Hedef ortamdaki testlerin yürütülmesi ya da yakından izlenmesi
Kalite güvencesi	Kısmen: tedarikçi kendi süreçlerini denetler	Kendi kalite güvencesi kayıtları	Tedarikçi gözetimi ve yazılım uygunluk gözden geçirmesi (software conformity review); gözetim kayıtları
Sertifikasyon irtibatı	Devredilmez	Planlar ve SAS için girdi	Otoriteyle mutabakat ve otoriteye sunulan veri

Hangi satır devredilirse devredilsin, iş tanımında beş şey açık olmalıdır: teslim edilecek iş ürünleri, kabul kriterleri, gözden geçirmeyi kimin yapacağı, değişikliklerin iki taraf arasında nasıl iletileceği ve doğrulamanın hangi kısmının kimde olduğu. Bunlardan biri yazılmadığında boşluk genellikle "karşı taraf yapıyor sanmıştık" cümlesiyle ortaya çıkar.

Zorluklar ve riskler

Dış kaynak kullanımının riskleri, işin teknik zorluğundan çok bilgi akışının kesintiye uğramasından kaynaklanır. Aşağıdaki dört risk alanı, uygulamada en sık karşılaşılanlardır.

İletişim ve saat dilimi farkları. Tedarikçi farklı bir ülkede veya saat diliminde çalışıyorsa, basit bir sorunun yanıtı bir iş gününü bulabilir. Gereksinim belirsizliği gibi hızlı netleştirme gerektiren konularda bu gecikme birikir: tedarikçi beklememek için varsayım yapar, varsayım yanlış çıkar ve yeniden çalışma doğar. Dil farkı da ayrı bir katmandır; teknik terimlerin iki tarafta farklı anlaşılması, gözden geçirmede geç fark edilen tutarsızlıklara yol açar.

Alan bilgisi eksikliği. Genel yazılım geliştirme becerisi ile aviyonik alan bilgisi aynı şey değildir. ARINC 429 etiket (label) yapısını, yazılım bölümlenmesi (software partitioning) kısıtlarını veya donanım-yazılım arayüzünün zamanlama davranışını bilmeyen bir ekip, sözdizimsel olarak doğru ama alan açısından hatalı iş üretebilir. Bu tür hatalar birim seviyesindeki doğrulamadan kaçır ve genellikle entegrasyon testinde — yani düzeltmenin en pahalı olduğu yerde — ortaya çıkar.

Kalite görünürlüğünün azalması. İç ekipte kalite sorunları gündelik temasla erken fark edilir: kod gözden geçirmeleri, koridor konuşmaları, ekip toplantıları. Tedarikçi tarafında bu doğal görünürlük yoktur; ana yüklenici yalnızca teslim edilen iş ürünlerini görür. Sorunlar teslimata kadar gizli kalır ve teslimatta toplu hâlde ortaya çıkar. Kalite güvencesi faaliyetleri tedarikçi sahasını kapsamıyorsa, süreç sapmaları hiç görülmeyebilir.

Sertifikasyon beklentilerinin aktarılamaması. En kritik risk budur. Tedarikçi "çalışan yazılım" teslim etmeye odaklanırken, ana yüklenicinin asıl ihtiyacı **kanıtlanabilir** yazılımdır: izlenebilirlik kayıtları, gözden geçirme tutanakları, yapısal kapsam analizi sonuçları, konfigürasyon yönetimi kayıtları. Sertifikasyon deneyimi olmayan bir tedarikçi bu çıktıları ya hiç üretmez ya da biçimsel olarak üretir; içeriğe bakılana kadar eksiklik fark edilmez. Katılım aşaması (Stage of Involvement, SOI) denetimlerinde ([12. Sertifikasyon İrtibatı](#)) tedarikçi kaynaklı bulgular başvuru sahibinin bulgusu olarak kaydedilir — otorite açısından "tedarikçi yaptı" diye bir mazeret yoktur.

Risk	Erken belirti	Görülmezse sonucu
İletişim gecikmesi	Soru-cevap süresinin uzaması, varsayım listelerinin şişmesi	Yanlış varsayım dayalı yeniden çalışma
Alan bilgisi eksikliği	Gereksinim yorum sorularının azlığı (soru sormayan tedarikçi şüphe uyandırmalı)	Entegrasyonda geç ve pahalı hatalar
Görünürlük kaybı	Ara teslimat/ölçüm paylaşımının aksaması	Teslimatta toplu sürpriz
Beklenti aktarımı eksikliği	Kanıt paketlerinin biçimsel ama içeriksiz olması	SOI denetim bulguları, sertifikasyon gecikmesi

Sertifikasyon açısından sınırda dikkat edilecekler

Yukarıdaki riskler her dış kaynak ilişkisinde görülür. Aşağıdaki altı konu ise doğrudan sertifikasyon kanıtıyla ilgilidir; iki kuruluş arasındaki sınırda sahipsiz kaldıklarında eksiklik çoğu zaman son denetimde fark edilir.

Doğrulama bağımsızlığı. Bağımsızlık (independence) kuruluş düzeyinde değil, kişi düzeyinde aranır: bağımsızlığın arandığı hedeflerde bir iş ürününü doğrulayan, onu geliştiren kişi olmamalıdır ([9. Yazılım Doğrulama](#)). İşin ayrı bir şirkette yapılması bunu kendiliğinden sağlamaz; tedarikçi hem geliştirip hem doğruluyorsa bağımsızlık tedarikçinin içinde kurulmalı ve kayıtlardan okunabilmelidir. Denetçi tek bir iş ürününü seçip "bunu kim yazdı, kim doğruladı?" diye sorduğunda cevap tedarikçinin gözden geçirme ve test kayıtlarında bulunmalıdır. Bağımsızlık iki kuruluş arasında da kurulabilir (tedarikçi geliştirir, ana yüklenici doğrular); ancak o zaman doğrulama ilgili verinin tamamını kapsamalı ve planlarda böyle tanımlanmalıdır. Ana yüklenicinin örneklemli gözden geçirmesi bir kabul faaliyetidir; tedarikçide eksik kalan bağımsızlığın yerini tutmaz.

Problem raporlama. İki kuruluş çoğunlukla iki ayrı problem raporu (problem report) sistemi kullanır. Hangi problemin hangi sistemde açılacağı, kimliklerin nasıl eşleneceği ve tedarikçinin kendi bulduğu problemleri ne zaman bildireceği baştan yazılmalıdır. Tedarikçi bir problemin emniyet etkisini tek başına değerlendiremez, çünkü sistem bağlamını göremez; değerlendirme başvuru sahibine kadar taşınmak zorundadır. Sertifikasyon anında açık kalan raporlar SAS'ta gerekçesiyle özetlendiği için, tedarikçide açık duran her rapor başvuru sahibince bilinmeli ve 9. bölümde anlatılan sınıflarla sınıflandırılmalıdır. FAA AC 20-189 ve EASA AMC 20-189, açık problem raporlarının öge, ekipman, sistem ve ürün düzeylerindeki taraflar arasında nasıl yönetileceğini ele alır.

Otoritenin erişimi. Sözleşmeler çoğunlukla ana yüklenicinin denetim hakkını düzenler, otoritenin erişimini unuttur. Oysa otorite, SOI denetimlerinin bir kısmını verinin üretildiği yerde yapmak ya da tedarikçide duran veriyi görmek isteyebilir. DO-178C de otorite gözden geçirmelerinin tedarikçinin tesisinde yapılabileceğini öngörür ve bunları ayarlamayı başvuru sahibine bırakır. Otoriteye sunulmayan yaşam döngüsü verisi de talep edildiğinde erişime açılmak zorunda olduğundan, bu erişim sözleşmede tedarikçi için de bir yükümlülük olarak yer almalıdır.

Alt tedarikçiler. Tedarikçi işin bir kısmını başka bir kuruluşa devredebilir. Sözleşme bunu ana yüklenicinin onayına bağlamalı ve aynı plan, standart, erişim ve gözetim koşullarının zincirin her halkasına aktarılmasını şart koşturmalıdır. Gözetim planında adı geçmeyen bir kuruluşun ürettiği veri, kökeni açıklanamayan veridir.

Ortam ve araç eşdeğerliği. Derleyici, bağlayıcı ve doğrulama araçlarının sürümleri ile seçenekleri iki tarafta aynı değilse, tedarikçide alınan test ve kapsam sonuçları teslim edilen çalıştırılabilir nesne kodunu (executable object code) temsil etmeyebilir; nesne kodunun arşivlenen kaynaktan yeniden üretilmesi de tehlikeye girer. Ortam, konfigürasyon kontrolü altında tanımlanmalı ve tedarikçinin kullandığı ortam bu tanımla karşılaştırılmalıdır. Kalifiye bir araç kullanılıyorsa kalifikasyonun hangi kullanım amacı ve hangi işletim ortamı için geçerli olduğu, kalifikasyon verisinin kimde durduğu ve tedarikçinin aracı o sınırlar içinde kullandığı ayrıca gösterilmelidir ([13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)).

Veri hakları ve sözleşme sonrası erişim. Yaşam döngüsü verisi, ürün hizmette kaldığı sürece değişiklik, problem araştırması ve yeniden sertifikasyon için gerekir. Sözleşme bittiğinde kaynak kodun, test ortamının, gözden geçirme kayıtlarının ve problem raporu geçmişinin kimde kalacağı; fikrî mülkiyetin ve ihracat kontrolü (export control) kısıtlarının bu veriye erişimi engelleyip engellemeyeceği iş başlamadan kararlaştırılmalıdır. Tedarikçinin kendi araç ya da kütüphaneleriyle ürettiği ve ancak onun ortamında yeniden üretilen bir çıktı, ilişki bittiğinde bakımı yapılamayan bir çıktıya dönüşür.

Önerilen önlemler

Yukarıdaki risklerin ortak özelliği, iş başladıktan sonra düzeltilmelerinin zor olmasıdır. Bu nedenle önlemlerin ağırlık merkezi sözleşme öncesi ve proje başlangıcı dönemidir.

Gözetimin planlarda tanımlanması

Tedarikçi gözetimi sözleşmeyle başlar ama sertifikasyon açısından asıl yeri planlardır ([5. Yazılım Planlama](#)). Yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC) sürece katılan kuruluşları ve sorumluluklarını, ayrıca tedarikçi süreç ve çıktılarının onaylı plan ve standartlara uymasının nasıl sağlanacağını otoriteye bildirir. Yazılım kalite güvencesi planı (Software Quality Assurance Plan, SQAP) bu güvencenin yolunu tanımlar; uygulamada bu, tedarikçide hangi denetimlerin, hangi sıklıkta ve kim tarafından yapılacağıdır. Yazılım konfigürasyon yönetimi planı (Software Configuration Management Plan, SCMP) konfigürasyon yönetimi gereklerinin tedarikçiye nasıl uygulanacağını, dolayısıyla tedarikçi verisinin nasıl teslim alınıp kontrol altına alınacağını söyler. Tedarikçi kendi plan ve standartlarıyla çalışacaksa, bunların projenin onaylı planlarıyla uyumu planlama aşamasında gözden geçirilir; sonradan fark edilen bir uyumsuzluk, üretilmiş verinin yeniden işlenmesi demektir.

Planların ayrıca neyin **gözetim kaydı** sayılacağını söylemesi gerekir, çünkü son denetimde gözetimin yapıldığı ancak kayıtlarla gösterilebilir. Tipik kayıtlar şunlardır: tedarikçi denetim raporları, teslimat kabul tutanakları, gözden geçirme kayıtları, bulgu ve düzeltici faaliyet (corrective action) takibi, sapma onayları. Kitabın denetim listeleri bu iki ucu sorar: planlama denetiminde gözetim yaklaşımının tanımlı olup olmadığı ([SW SOI-1](#)), son denetimde gözetim kayıtlarının tamam olup olmadığı ([SW SOI-4](#)).

Erken ve yazılı beklenti aktarımı

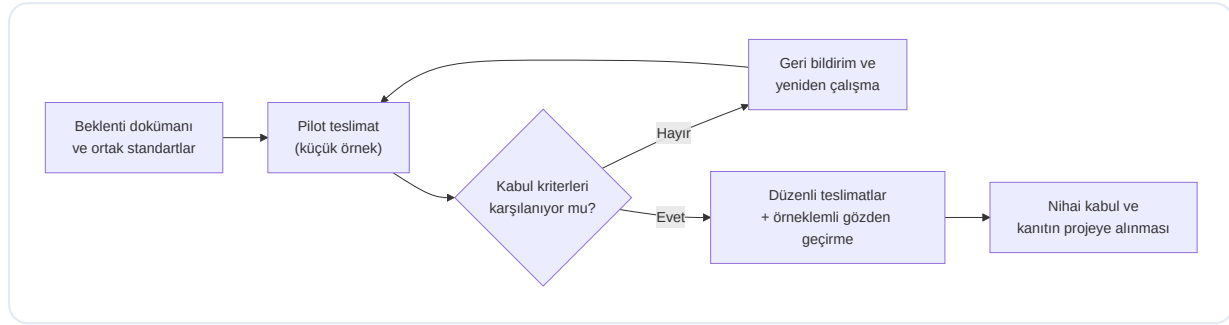
Sertifikasyon beklentileri sözlü anlatımla aktarılamaz. Tedarikçiye, hangi iş ürünlerinin hangi içerikle teslim edileceğini, hangi yazılım seviyesinin geçerli olduğunu ve hangi kanıtların bekleneceğini açıkça tanımlayan yazılı bir beklenti dokümanı verilmelidir. Pratikte bu, sözleşmenin ekinde yer alan bir iş tanımı ile ana yüklenicinin plan setinin (geliştirme, doğrulama, konfigürasyon yönetimi, kalite güvencesi planları) ilgili kısımlarının tedarikçiye uygulanabilir hâle getirilmesi anlamına gelir. "Nasıl olsa deneyimlidirler" varsayımı en pahalı varsayımdır.

Ortak standart seti

Tedarikçinin kendi kodlama standardı, kendi gereksinim şablonu ve kendi gözden geçirme kontrol listesiyle çalışması, teslim edilen ürünlerin ana projeye alınmasını zorlaştırır. En baştan tek bir set üzerinde anlaşılmalıdır: gereksinim, tasarım ve kod standartları, doküman şablonları, araç sürümleri ve isimlendirme kuralları. Ortak set, gözden geçirme maliyetini düşürür ve "biz farklı yapıyoruz" tartışmalarını iş başlamadan bitirir. Uçak ya da motor üreticisinin sözleşmeyle getirdiği ek standartlar varsa DO-178C bunların planlamada ve tedarikçi gözetiminde hesaba katılmasını bekler; set zincirin alt halkalarına da aynen aktarılır.

Aşamalı kabul

Tedarikçi çıktısını yalnızca proje sonunda tek seferde kabul etmek, sorunların en geç ve en pahalı noktada bulunması demektir. Bunun yerine kabul, küçük ve erken örneklerle başlamalıdır: ilk teslim edilen birkaç gereksinim, ilk kod modülü, ilk test prosedürü ayrıntılı gözden geçirilir; beklentiyle uyum doğrulanır ve sapmalar tedarikçiye erken geri bildirilir. Sonraki teslimatlarda örneklem genişletilir veya daraltılır — tedarikçinin gösterdiği olgunlukla orantılı olarak.



Pilot teslimatın amacı ürünü onaylamak değil, tedarikçinin beklentiyi anlayıp anlamadığını görmektir. Bu yüzden kabul kriterleri biçime değil içeriğe bakmalıdır. Küçük bir pilot paket (birkaç gereksinim, onları gerçekleyen kod ve testleri) için örnek bir kontrol listesi:

- Gereksinimler ortak gereksinim standardına uyuyor mu; her biri tek anlamlı ve doğrulanabilir mi, kimlikleri kararlaştırılan şemaya göre mi verilmiş?
- İzlenebilirlik iki yönde kurulu mu: her düşük seviyeli gereksinim bir üst gereksinime ya da gerekçesi yazılmış bir türetilmiş gereksinim kaydına, her kod birimi bir gereksinime bağlanıyor mu?
- Gözden geçirme kaydı kimin hazırladığını, kimin gözden geçirdiğini, hangi sürüme bakıldığını ve bulguların nasıl kapandığını gösteriyor mu? Hiç bulgusu olmayan kayıt ayrıca sorgulanmalıdır.
- Test prosedürü ana yüklenicinin ortamında yeniden koşulduğunda aynı sonucu veriyor mu?
- Teslimat tanımlı bir temel çizgiden (baseline) mi çıkmış; paketteki her ögenin kimliği ve sürümü teslimat listesiyle eşleşiyor mu?
- Tedarikçinin yaptığı varsayımlar ve açık problemler teslimatla birlikte bildirilmiş mi?

Tedarikçi verisinin projeye alınması

Kabul edilen iş ürünü, ana projenin konfigürasyon yönetimine ve izlenebilirlik zincirine girdiği anda sertifikasyon kanıtının parçası olur ([10. Yazılım Konfigürasyon Yönetimi](#)). Kökeni belirsiz çıktı, üzerine kurulan her kanıtı zayıflatır; bu yüzden devir teslimin kendisi tanımlı bir işlem olmalıdır.

İzlenebilirlik eşlemesi. En sağlam çözüm, iki tarafın aynı kimlik şemasını ve mümkünse aynı gereksinim yönetim ortamını kullanmasıdır. Tedarikçi kendi aracında kendi kimlikleriyle çalışıyorsa, iki kimlik kümesi arasındaki eşleme tablosu da konfigürasyon kontrolüne alınır; elle

tutulan ve sürümü belli olmayan bir eşleme, iz verisini güvenilirmez kılar. Tipik hata, tedarikçinin düşük seviyeli gereksinimlerini üst gereksinimlerin eski bir sürümüne bağlamış olmasıdır. Bunu görünür kılmanın yolu, her teslimatın hangi girdi temel çizgisine göre üretildiğini kayda geçirmektir.

Temel çizgi devri. Her teslimat tedarikçi tarafında tanımlı bir temel çizgiden çıkmalı ve bir teslimat notuyla gelmelidir: içerik listesi ve sürümler, bütünlüğü denetlemek için sağlama toplamları (checksum), esas alınan girdi temel çizgisi, bilinen problemler ve onaylı sapmalar. Ana yüklenici paketi bu nota göre denetler, kabul eder ve kendi konfigürasyon yönetimine alır. Bu andan sonra hangi kopyanın geçerli olduğu tek anlamlı olmalıdır; aynı dosyanın iki tarafta bağımsızca değiştiği durum, iki kuruluşlu projelerde sık rastlanan bir konfigürasyon sorunudur.

Değişikliklerin iki yönde akışı. Üst seviye gereksinim değiştiğinde tedarikçiye resmî bildirim gider ve değişiklik etki analizi (change impact analysis) tedarikçinin elindeki veriyi de kapsar. Tedarikçinin önerdiği değişiklik ise ana yüklenicinin değişiklik kontrolünden geçmeden temel çizgiye girmez.

Arşiv. Projenin sonunda arşivlenen paket, tedarikçide üretilen veriyi de içermelidir: yalnızca son sürümü değil, onu yeniden üretmek ve doğrulamak için gereken ortam tanımını, test prosedürlerini ve kayıtları da.

Tedarikçi süreçlerinin denetlenmesi

Teslim edilen ürünü gözden geçirmek yeterli değildir; ürünü üreten sürecin de plana uygun işlediği doğrulanmalıdır. Ana yüklenicinin kalite güvencesi ekibi ([11. Yazılım Kalite Güvencesi](#)), tedarikçi sahasında (veya uzaktan) periyodik denetimler yapmalı; gözden geçirme kayıtlarının gerçekten tutulduğunu, problem raporlarının izlendiğini ve konfigürasyon yönetiminin işlediğini örnekleme yoluyla kontrol etmelidir. Denetim bulguları kayda geçirilir ve kapanışları izlenir; bu kayıtlar gözetim kanıtının çekirdeğidir. Denetim hakkı sözleşmeye yazılmalıdır — sözleşmede yer almayan denetim, ihtiyaç doğduğunda pazarlık konusu olur.

İletişim kanalları

Tüm iletişimi tek bir temas noktasından geçirmek yerine, teknik sorular için mühendisler arasında **doğrudan bir teknik kanal** açmak, iletişim gecikmesi riskini belirgin biçimde azaltır. Resmî yazışma kanalı sözleşme ve değişiklik iletişimi için korunur; teknik netleştirme hızlı kanaldan yürür ve sonuçları kayıt altına alınır. Sınır şudur: hızlı kanalda verilen bir cevap bir gereksinimin anlamını ya da kapsamını değiştiriyorsa, artık netleştirme değil değişikliktir ve resmî değişiklik sürecine girer. Aksi hâlde tedarikçinin kodu, hiçbir temel çizgide yazılı olmayan bir karara dayanır.

Bu bölümden akılda kalması gerekenler

- Dış kaynak, sorumluluğu devretmez: otorite karşısında başvuru sahibi sorumludur ve tedarikçinin ürettiği veri aynı hedeflere, aynı plan ve standartlara tabidir.
- Saat ücreti karşılaştırması yanıltıcıdır; gözden geçirme, koordinasyon, projeye alma ve gözetim gibi gizli maliyetler baştan bütçelenmelidir.
- En kritik risk, sertifikasyon beklentilerinin tedarikçiye aktarılamamasıdır: "çalışan yazılım" ile "kanıtlanabilir yazılım" aynı şey değildir.
- Gözetim yaklaşımı PSAC'ta ve ilgili planlarda tanımlanır; gözetimin yapıldığı, son denetimde ancak gözetim kayıtlarıyla gösterilebilir.
- Bağımsızlık, problem raporlarının görünürlüğü, otoritenin erişimi, alt tedarikçiler, ortam eşdeğerliği ve veri hakları sınırda sahipsiz kalmaya en yatkın konulardır; sözleşmede ve planlarda karşılıkları olmalıdır.
- Beklentiler yazılı aktarılır, ortak bir standart seti üzerinde anlaşılır; kabul proje sonuna bırakılmaz, küçük pilot teslimatlarla erken başlar.
- Tedarikçi teslimatı tanımlı bir temel çizgi olarak alınır, izlenebilirlik eşlemesiyle projeye bağlanır; ürünün yanında süreç de denetlenir ve denetim hakkı sözleşmeye yazılır.

Ekler

Ek A: Örnek Geçiş Kriterleri

Geçiş kriterleri (transition criteria), bir yazılım sürecine ne zaman girilebileceğini ve o süreçten ne zaman çıkmış sayılacağını tanımlayan, planlarda yazılı ölçütlerdir. Bu ek, her süreç için örnek giriş ve çıkış kriterleri ile zayıf bir kriteri güçlü hâle getirmenin yollarını gösterir.

Buradaki tablolar bir şablon değil, başlangıç noktasıdır. Kriterler yazılım seviyesine, ekip yapısına ve kullanılan araçlara göre uyarlanmalı ve [5. Yazılım Planlama](#) bölümünde anlatılan planlara yazılmalıdır: geliştirme süreçlerinin kriterleri yazılım geliştirme planında (Software Development Plan, SDP), doğrulamaya giriş kriterleri yazılım doğrulama planında (Software Verification Plan, SVP) yer alır; konfigürasyon yönetimi ve kalite güvencesi süreçlerinin kriterleri de kendi planlarında tanımlanır.

DO-178C geçiş kriterini, bir sürece girmek için sağlanması gereken asgari koşullar olarak ele alır; "çıkış kriteri" standardın terimi değil, yaygın bir uygulamadır. İkisi aynı kapının iki yüzüdür: bir süreçten çıkış için yazılan koşul, uygulamada bir sonrakine giriş koşulunun parçası olur. Standart hazır bir kriter listesi de vermez; planlardan her süreç için üç şeyi belirtmesini bekler: sürece ne girecek (başka süreçlerin geri bildirimi dahil), o girdi üzerinde hangi doğrulama, konfigürasyon yönetimi ya da kalite güvencesi faaliyeti yapılmış olacak ve hangi araç, yöntem ve prosedür hazır bulunacak.

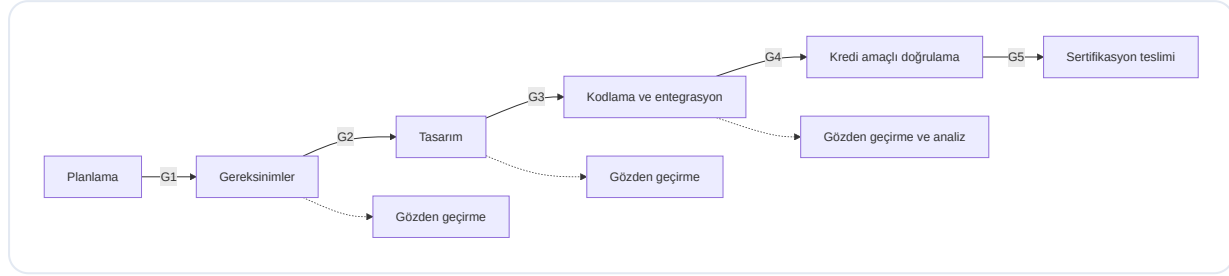
Geçiş kriteri ne değildir?

Geçiş kriteri bir şelale modeli dayatmaz. Gereksinimlerin tamamı bitmeden tasarıma başlanabilir; önemli olan, tasarıma giren her gereksinim kümesinin kendi giriş kriterini karşılamasıdır. Kriterler bu yüzden çoğu zaman tüm proje için değil, bir bileşen, bir işlev ya da bir sürüm için uygulanır. Aynı esneklik planlama için de geçerlidir: ilgili faaliyetin kriteri sağlanıyorsa başka süreçler planlama süreci tamamlanmadan başlayabilir.

Kısmi girdiyle ilerlemenin bir karşılığı vardır: girdilerinin bir kısmı eksikken başlatılan süreç, sonradan gelen ya da değişen girdiler karşısında önceki çıktıların hâlâ geçerli olduğunu yeniden incelemek zorundadır. On gereksinimle başlanan tasarıma on birincisi eklendiğinde soru "yeni gereksinim tasarlandı mı?" ile bitmez; "önceki tasarım kararları hâlâ doğru mu?" diye de sorulur. Geri dönüşler de aynı mantıkla kritere bağlanır. Temel çizgiye (baseline) alınmış bir veri değişecekse sürece yeniden girişin de koşulları vardır: bir problem raporu ya da değişiklik talebi, değişiklik etki analizi (change impact analysis) ve etkilenen gözden geçirmelerle testlerin yinelenmesi ([10. Yazılım Konfigürasyon Yönetimi](#)).

Geçiş kriteri bir toplantı kararı da değildir. "Ekip hazır olduğumuza karar verdi" ifadesi, neye bakılarak karar verildiğini söylemediği için kanıt değeri taşımaz. İyi bir kriter, bir kayda ya da ölçüme bakılarak "karşılandı / karşılanmadı" diye yanıtlanabilir.

Süreçler ve geçiş noktaları



Her geçiş noktasında (G1–G5) sorulan soru aynıdır: Bir sonraki sürecin ihtiyaç duyduğu girdi var mı, doğrulanmış mı ve konfigürasyon yönetimi altında mı? Kesikli oklar, gözden geçirme ve analizlerin geliştirmeye eşlik ettiğini hatırlatır; doğrulama, sıranın sonunda bekleyen tek bir adım değildir.

Tablolardaki "Çıkış" satırları geride bırakılan sürecin tamamlaması gerekenleri, "Giriş" satırları ise girilen sürecin bunlara ek olarak hazır bulmak istediği koşulları (temel çizgi, ortam, başka süreçlerden gelen girdi) gösterir. Bir geçiş noktası, ikisi birlikte sağlandığında geçilmiş sayılır.

Örnek kriterler

Tablolarda geçen belge kısaltmaları: yazılım sertifikasyon planı (Plan for Software Aspects of Certification, PSAC), yazılım başarı özeti (Software Accomplishment Summary, SAS), yazılım konfigürasyon indeksi (Software Configuration Index, SCI) ve yazılım yaşam döngüsü ortam konfigürasyon indeksi (Software Life Cycle Environment Configuration Index, SECI). "Kredi amaçlı (for-credit) doğrulama", sonuçları sertifikasyon kanıtı olarak kullanılacak resmî test koşullarını anlatır. Yazılım uygunluk gözden geçirmesi (software conformity review), kalite güvencesinin sertifikasyona sunulan sürüm için yaptığı kapanış denetimidir; katılım aşaması (Stage of Involvement, SOI) ise otoritenin projeyi denetlediği aşamalardan her biridir. Diğer kısaltmalar için [Kısaltmalar](#) sayfasına bakınız.

G1 – Planlamadan gereksinimlere

Tür	Örnek kriter	Kanıt
Çıkış	Beş plan ve üç standart gözden geçirilmiş, bulgular kapatılmış	Gözden geçirme kayıtları
Çıkış	Planlar ve standartlar temel çizgiye alınmış	Konfigürasyon kaydı
Çıkış	PSAC otoriteye gönderilmiş	Gönderim kaydı
Giriş	İlgili sistem gereksinimleri temel çizgide; yazılım seviyesi belirlenmiş	Konfigürasyon kaydı, sistem emniyet değerlendirme sürecinin çıktısı

G2 – Gereksinimlerden tasarıma

Tür	Örnek kriter	Kanıt
Çıkış	Bileşenin yüksek seviyeli gereksinimleri (high-level requirements) gözden geçirilmiş; açık büyük bulgu yok	Gözden geçirme kaydı
Çıkış	Her gereksinim bir sistem gereksinimine izlenebilir ya da türetilmiş gereksinim (derived requirement) olarak işaretli	İzlenebilirlik raporu
Çıkış	Türetilmiş gereksinimler gerekçeleriyle sistem süreçlerine (sistem emniyet değerlendirmesi dahil) iletilmiş	İletim kaydı
Giriş	Tasarımı yapılacak bileşenin yüksek seviyeli gereksinimleri temel çizgide	Konfigürasyon kaydı

G3 – Tasarımdan kodlamaya

Tür	Örnek kriter	Kanıt
Çıkış	Mimari ve düşük seviyeli gereksinimler (low-level requirements) gözden geçirilmiş	Gözden geçirme kaydı
Çıkış	Arayüz tanımları (veri tipleri, birimler, aralıklar) tamam	Arayüz belgesi
Çıkış	Düşük seviyeli gereksinimler yüksek seviyeli gereksinimlere izlenebilir ya da türetilmiş olarak işaretli; türetilmiş olanlar sistem süreçlerine iletilmiş	İzlenebilirlik raporu, iletim kaydı
Giriş	Kodlanacak bileşenin mimarisi ve düşük seviyeli gereksinimleri temel çizgide	Konfigürasyon kaydı
Giriş	Derleyici, bağlayıcı ve seçenekleri belirlenmiş ve kayıtlı	SECI

G4 – Kodlamadan kredi amaçlı doğrulamaya

Tür	Örnek kriter	Kanıt
Çıkış	Kaynak kod gözden geçirilmiş; kodlama standardı denetim aracında ihlal yok ya da sapmalar gerekçeli	Gözden geçirme kaydı, araç raporu
Çıkış	Kod derleniyor ve hedefe yüklenebiliyor; derleme yeniden üretilebilir	Derleme kaydı
Giriş	Test durumları ve prosedürleri gözden geçirilmiş	Gözden geçirme kaydı
Giriş	Test ortamı tanımlanmış, yapılandırılmış ve kayıtlı	SECI, ortam kaydı
Giriş	Koşulacak kod, test durumları ve prosedürleri temel çizgide	Konfigürasyon kaydı
Giriş	Kuru koşu (dry run) tamamlanmış; bilinen başarısızlıklar problem raporuna bağlı	Kuru koşu kaydı, problem raporları

G5 – Doğrulamadan sertifikasyon teslimine

Tür	Örnek kriter	Kanıt
Çıkış	Tüm testler koşulmuş; başarısızlar problem raporuna bağlanmış	Test sonuçları, problem raporları
Çıkış	Gereksinim kapsam analizi ve yapısal kapsam analizi (structural coverage analysis) tamam; boşluklar çözülmüş	Kapsam raporları
Giriş	Sertifikasyona esas sürüm son temel çizgide	Konfigürasyon kaydı, SCI
Giriş	Açık problem raporlarının her biri emniyet ve işlev etkisi açısından değerlendirilmiş	Problem raporu listesi ve değerlendirmesi
Giriş	SAS ve SCI yayımlanmış, birbiriyle tutarlı	Belgeler, kalite güvencesi kontrolü
Giriş	Yazılım uygunluk gözden geçirmesi tamamlanmış	Kalite güvencesi kaydı
Giriş	Önceki tüm katılım aşaması bulguları kapanmış	Bulgu kapanış listesi

Seviyeye göre uyarılama

Yukarıdaki tablolar seviyeden bağımsız yazılmıştır; plana aktarılırken her kriter, o yazılım seviyesinde aranan hedeflerle eşleştirilir. Örneğin yapısal kapsam ölçütü seviyeyle birlikte satır kapsamadan (statement coverage) karar kapsamaya (decision coverage), oradan değiştirilmiş koşul/karar kapsamaya (modified condition/decision coverage, MC/DC) yükselir. Veri ve kontrol bağlaşımı (data and control coupling) analizi Seviye A, B ve C'de aranır; bağımsızlık (independence) ise yalnızca belirli hedeflerde şarttır. Ölçütlerin tanımı [9. Yazılım Doğrulama](#) bölümündedir.

Kriter	Seviye A	Seviye B	Seviye C	Seviye D
Yapısal kapsam analizi tamam (G5)	MC/DC, karar ve satır kapsama; veri ve kontrol bağlaşımı	Karar ve satır kapsama; veri ve kontrol bağlaşımı	Satır kapsama; veri ve kontrol bağlaşımı	Yapısal kapsam aranmaz; kriter yüksek seviyeli gereksinimlerin test kapsamıyla sınırlı kalır
Gözden geçirmeyi yazar dışında biri yapmış (G2–G4)	Bağımsızlık aranan hedeflerde şart	Bağımsızlık aranan hedeflerde şart; bu hedefler A'dakinden azdır	Standart şartı değil; iyi uygulama	Standart şartı değil; iyi uygulama

Kriter	Seviye A	Seviye B	Seviye C	Seviye D
Kalite güvencesi kriterlerinin sağlandığını denetlemiş (tüm geçişler)	Standardın hedefi; bağımsızlıkla	Standardın hedefi; bağımsızlıkla	Standardın hedefi; bağımsızlıkla	Ayrı hedef olarak aranmaz, yaşam döngüsünün ve geçiş kriterlerinin tanımlanması da bu seviyede hedef değildir; planlara uygunluk hedefi geçerlidir

Düşük seviyede bir hedefin aranmaması, ilgili kriteri yazmayı yasaklamaz; yalnızca kriterin standart gereği değil, projenin kendi kararı olduğunu gösterir. Yine de plana yazılan her kriter bağlayıcıdır: kendi koyduğu kriteri uygulamayan ekip de plan sapması üretir.

Zayıf kriteri güçlendirmek

Zayıf ifade	Sorun	Güçlü ifade
"Gereksinimler büyük ölçüde tamamlandığında"	Ne kadar? Kim karar veriyor?	"Bileşenin tüm yüksek seviyeli gereksinimleri gözden geçirilmiş ve açık büyük bulgu yok"
"Tasarım uygun bulunduğunda"	Kim, neye göre uygun buluyor?	"Mimari ve düşük seviyeli gereksinimler tasarım standardına göre gözden geçirilmiş, bulgular kapatılmış"
"Kod hazır olduğunda"	"Hazır" tanımsız	"Kod gözden geçirilmiş, standart ihlali yok ya da sapmalar gerekçeli, temel çizgiye alınmış"
"Testler başarılı olduğunda"	Başarısız testlerin ne olacağı belli değil	"Tüm testler koşulmuş; başarısız her test bir problem raporuna bağlanmış ve değerlendirilmiş"
"Kalite güvencesi onayıyla"	Neye bakılarak onaylanacağı belli değil	"Kalite güvencesi, kriter listesini kayıtlarla karşılaştırmış ve uygunsuzluk açmamış"

Kriterlerin denetlenmesi

Geçiş kriterleri yazıldığı gibi uygulanmıyorsa, otorite bunu bir plan sapması olarak görür. Kriterlerin sağlandığına dair güvence elde etmek, DO-178C'de kalite güvencesi sürecinin hedeflerinden biridir; bu hedef Seviye A, B ve C'de, bağımsızlıkla aranır ([11. Yazılım Kalite Güvencesi](#)). Kalite güvencesinin bunu yapabilmesi için her kriterin arkasında gösterilebilir bir kayıt bulunmalı, kredi amaçlı koşu gibi tanıklık etmek isteyebileceği geçişler de kendisine önceden bildirilmelidir. Bir kriter pratikte işlemiyorsa sessizce aşılmamalı; plan güncellenmeli ve değişiklik kayda alınmalıdır. Geçiş kriterleri aynı zamanda katılım aşaması denetimlerine hazırlığın da omurgasıdır ([SW SOI-1](#)).

Bu ekten akılda kalması gerekenler

- Geçiş kriteri, bir kayda ya da ölçüme bakılarak yanıtlanabilen bir sorudur. DO-178C onu sürece giriş için asgari koşul olarak tanımlar, içeriğini planlara bırakır.
- Kriterler şelale dayatmaz; her iş ürününün bir sonraki sürece hazır olduğunu gösterir. Kısmi girdiyle başlanan işte sonradan gelen girdi, önceki çıktının yeniden incelenmesini gerektirir.
- Bir süreçten çıkış koşulu, bir sonrakine giriş koşulunun parçasıdır; kredi amaçlı doğrulamaya yalnızca temel çizgideki kod, test prosedürleri ve kayıtlı ortamla girilir.
- Kriterler yazılım seviyesine göre uyarlanır; standardın aramadığı bir kriter projenin kendi kararıdır, ama plana yazıldıysa bağlayıcıdır.
- "Büyük ölçüde", "hazır", "uygun" gibi ölçülemez ifadeler kriter değildir.
- Kriterlerin sağlandığına dair güvence kalite güvencesinin işidir; işlemeyen bir kriter aşılmaz, plan güncellenir.

Ek B: Gerçek Zamanlı İşletim Sistemlerinde Endişe Alanları

Bu ek, gerçek zamanlı işletim sistemi (real-time operating system, RTOS) kullanan bir projede tasarım ve doğrulama sırasında izlenmesi gereken risk alanlarını toplar. Her alan için neyin ters gidebileceği, bunun nasıl görüneceği ve hangi analiz, test ya da tasarım kuralıyla ele alınacağı özetlenir.

Ek C, bir RTOS'u **seçerken** sorulacak soruları içerir; bu ek ise seçilmiş bir RTOS'u projede **kullanırken** ortaya çıkan riskleri ele alır. Arka plan için [20. Gerçek Zamanlı İşletim Sistemleri](#) bölümüne bakınız. Her alanın sonundaki sorular, tasarım gözden geçirmesinde ya da bir denetime hazırlanırken kontrol listesi olarak kullanılabilir.

Özet tablo

Alan	Tipik risk	Ele alma yolu
Çizelgeleme	Zaman sınırının yük altında kaçırılması	Çizelgelenebilirlik analizi, en kötü durum ölçümleri
Senkronizasyon	Öncelik terslenmesi, kilitlenme	Öncelik kalıtımı/tavanı, kilit sıralama kuralı, analiz
Kesmeler	Sınırsız kesme gecikmesi, kesme fırtınası	Kısa kesme rutinleri, kesme kapalı süresinin ölçümü, hız sınırlama
Bellek	Yığın taşması, parçalanma	Statik ayırma, yığın analizi, bellek koruma birimi
Görevler arası iletişim	Kuyruk taşması, bayat ya da yarım veri	Boyut analizi, tazelik kontrolü, atomik erişim
Saat ve zaman	Sayaç taşması, saat kayması	Taşma süresi hesabı, taşmaya dayanıklı kod, uzun süreli test
Başlatma ve hata yönetimi	Belirsiz başlangıç durumu, sessiz hata	Tanımlı başlatma sırası, sağlık izleme, bekçi köpeği
Bölümleme	Bir bölümün diğerini etkilemesi	Alan ve zaman bölümlemesi kanıtı
Çok çekirdek	Çekirdekler arası girişim	Girişim kanallarının analizi ve ölçümü
Yapılandırma	Paket dışı özellik, kullanılmayan kod	Yapılandırma kontrolü, kapsam analizi

Çizelgeleme ve zaman bütçesi

Çizelgeleme (scheduling), görevlerin işlemciyi hangi sırayla ve ne kadar süreyle kullanacağını belirler. Her görevin en kötü durum yürütme süresi (worst-case execution time, WCET), periyodu ve zaman sınırı bilinmeden sistemin zamanında çalışacağı gösterilemez. Ortalama ölçümlerle kurulan bir zaman bütçesi, nadir bir yük kombinasyonunda çöker. Görevlerin toplam işlemci kullanımı, RTOS çekirdek hizmetlerinin süresi ve kesmelerin araya girmesi birlikte hesaba katılmalıdır. WCET ve yığın kullanımı analizlerinin nasıl yapıldığı [9. Yazılım Doğrulama](#) bölümünde anlatılır.

- Çizelgelenebilirlik analizi (schedulability analysis) yapılmış mı ve ne kadar kalan pay (margin) gösteriyor?
- Görev öncelikleri hangi kurala göre atanmış (örneğin periyoda ya da zaman sınırına göre) ve gerekçesi tasarım verisinde yazılı mı?
- Aşırı yük durumunda hangi görevin zaman sınırını kaçıracağı tasarımda belirlenmiş mi?
- Zamanlayıcı tık (tick) çözünürlüğü en kısa periyotlu görev için yeterli mi?

Paylaşılan kaynaklar ve senkronizasyon

Paylaşılan kaynakların iki klasik tuzağı öncelik terslenmesi (priority inversion) ve kilitlenmedir (deadlock): ilkinde yüksek öncelikli görev, düşük öncelikli bir görevin tuttuğu kaynak yüzünden öngörülemez süre bekler; ikincisinde görevler birbirinin tuttuğu kaynağı bekleyerek hiç ilerleyemez. İkisi de nadir tetiklenir; testte görülmeyip sahada ortaya çıkabilir.

- Paylaşılan her kaynak için kullanılan koruma mekanizması ve protokol (öncelik kalıtımı ya da öncelik tavanı) belgelenmiş mi?
- İç içe kilit (lock) alınıyorsa, tüm görevlerin uyduğu tek bir kilit alma sırası var mı?
- Bir kilit için en uzun bekleme süresi sınırlı ve zaman bütçesine eklenmiş mi?
- Birden çok görevden ya da kesme bağlamından çağrılan kütüphane ve sürücü işlevlerinin yeniden girilebilir (reentrant) olduğu gösterilmiş mi; olmayanlar bir kilitte korunuyor mu?

Kesmeler

Kesme servis rutinlerinin (interrupt service routine, ISR) uzunluğu ve kesmelerin kapalı tutulduğu en uzun süre, sistemin tepki süresini doğrudan belirler. Arızalı bir çevre birimi saniyede binlerce kesme üreterek görevleri aç bırakabilir.

- Kesme rutinleri yalnızca veriyi alıp bir göreve aktaracak kadar kısa mı?
- Kesmelerin kapalı kaldığı en uzun süre ölçülmüş ya da analiz edilmiş mi?

- Bir kaynaktan gelen aşırı kesmeye karşı hız sınırlama ya da devre dışı bırakma stratejisi var mı?
- Kayan nokta (floating-point) işlemi yapan görev ve kesme rutinleri için kayan nokta yazmaçlarının bağlam değiştirme (context switch) sırasında saklanıp geri yüklendiği doğrulanmış mı?

Bellek

Çalışma sırasında dinamik bellek ayırma (dynamic memory allocation) parçalanma ve tükenme riski taşır; çoğu proje onu yasaklar ya da başlatma aşamasıyla sınırlar. Görev başına ayrılan yığın (stack) alanı yetersizse taşma, komşu bellek alanını sessizce bozar.

- Her görevin yığın ihtiyacı en kötü çağrı derinliğine ve kesme yüküne göre analiz edilmiş mi?
- Yığın taşması çalışma sırasında tespit ediliyor mu (koruma bölgesi ya da bellek koruma birimi; memory protection unit, MPU)?
- MPU, görevlerin birbirinin alanına yazmasını engelleyecek biçimde yapılandırılmış mı?
- Dinamik bellek ayırmaya ilişkin kural kodlama standardında yazılı mı ve uyulduğu araçla ya da gözden geçirmeyle denetleniyor mu?

Görevler arası iletişim

Kuyruklar ve paylaşılan değişkenler veri kaybı ve tutarsızlık için doğal noktalardır. Birden çok kelimedenden oluşan bir veri bir görev tarafından yazılırken diğer görev okursa, yarısı eski yarısı yeni bir değer elde edilir.

- Her kuyruğun boyutu en kötü üretim–tüketim oranına göre seçilmiş mi; taşmada ne olur?
- Okunan verinin tazeliği (ne kadar eski olduğu) kontrol ediliyor mu?
- Çok kelimelik veriye erişim atomik mi, yoksa tutarsız okuma mümkün mü?

Saat ve zaman

Zaman sayaçlarının taşması klasik bir hata sınıfıdır: milisaniyede bir artan 32 bitlik bir sayaç yaklaşık 49,7 günde başa döner. Sistem bu süreden uzun açık kalabiliyorsa, taşma anındaki davranış tasarlanmış ve test edilmiş olmalıdır.

Sorun kuramsal değildir. FAA'nın 2015 tarihli AD 2015-09-07 sayılı uçuşa elverişlilik direktifi (airworthiness directive, AD), Boeing 787'nin jeneratör kontrol birimlerindeki bir yazılım sayacının 248 gün kesintisiz enerjili kalındığında taşıdığını ve birlikte enerjilenmiş birimlerin aynı anda arıza emniyetli moda (failsafe mode) geçerek tüm alternatif akım gücünün kaybına yol açabileceğini

bildirmiştir. Direktif, geçici önlem olarak uçağın elektrik gücünün belirli aralıklarla kesilip yeniden verilmesini zorunlu kılmış; kalıcı çözüm, sonraki bir direktifle zorunlu tutulan yazılım güncellemesi olmuştur. Böyle bir hata olağan süreli testlerde ortaya çıkmaz, çünkü test düzenekleri nadiren aylarca kesintisiz çalıştırılır; onu erken ve güvenle bulan, sayacın genişliği ile artış hızından yapılan hesaptır.

Taşma kaçınılmazsa kod ona dayanıklı yazılır. İki sayaç değerinin farkı işaretersiz çıkarmayla alındığında, sayaç arada başa dönmüş olsa bile sonuç doğrudur:

```
#include <stdbool.h>
#include <stdint.h>

/* Koşul: ölçülen aralık, sayacın taşma süresinden (2^32 ms) kısa olmalıdır. */
bool sure_doldu(uint32_t simdi_ms, uint32_t baslangic_ms, uint32_t sure_ms)
{
    uint32_t gecen_ms = simdi_ms - baslangic_ms;

    return (gecen_ms >= sure_ms);
}
```

Bitiş anını toplayarak bulup `simdi_ms >= baslangic_ms + sure_ms` biçiminde karşılaştırmak ise hatalıdır: toplam başa döndüğünde koşul, süre dolmadan doğru olur.

- Her zaman sayacının taşma süresi hesaplanmış ve sistemin en uzun çalışma süresiyle karşılaştırılmış mı?
- Zaman farkı hesapları taşmaya dayanıklı biçimde yazılmış mı?
- Birden çok saat kaynağı varsa aralarındaki kayma ve eşzamanlama ele alınmış mı?

Başlatma, hata yönetimi ve yeniden başlatma

Başlatma sırasında görevlerin, sürücülerin ve paylaşılan verinin hangi sırayla hazır hâle geldiği tanımlı değilse, sistem her açılışta farklı bir durumdan başlayabilir. Çalışma sırasındaki hatalar da (istisna, zaman aşımı, yığın taşması) tanımlı bir tepkiye bağlanmalıdır.

- Başlatma sırası belgelenmiş ve her adımın başarısızlığında ne olacağı belli mi?
- Sağlık izleme (health monitoring) hangi olayları yakalıyor ve her birine hangi tepkiyi veriyor?
- Çekirdek çağrılarının hata dönüş değerleri (kuyruk dolu, zaman aşımı, geçersiz parametre) her çağrıda denetleniyor ve tanımlı bir tepkiye bağlanıyor mu?
- Bekçi köpeği zamanlayıcısı (watchdog) ile yeniden başlatma sonrası hangi durumun korunacağı tanımlı mı?

Bölümleme

Farklı yazılım seviyelerinden bileşenler aynı işlemciyi paylaşıyorsa, bir bölümdeki (partition) hatanın diğerinin belleğini ya da zaman bütçesini etkileyemeyeceği gösterilmelidir. İlki alan bölümlemesinin (spatial partitioning), ikincisi zaman bölümlemesinin (temporal partitioning) konusudur ve biri diğerinin yerini tutmaz: bellek koruma birimi yalnızca alan korumasını sağlar. Ayrıntı için [21. Yazılım Bölümlemesi](#) bölümüne bakınız.

- Bölümleme yapılandırması (bellek bölgeleri, zaman pencereleri, erişim hakları) konfigürasyon kontrolünde mi ve tasarımı karşılaştırılarak gözden geçirilmiş mi?
- Bir bölüm zaman penceresini aştığında ya da izinsiz belleğe eriştiğinde ne olduğu gürbüzlük testleriyle gösterilmiş mi?
- Bölümlerin ortak kullandığı kaynaklar (giriş/çıkış aygıtları, kesmeler, haberleşme kanalları) listelenmiş ve her biri için hatanın yayılma yolu analiz edilmiş mi?

Çok çekirdekli işlemciler

Çok çekirdekli işlemcilerde çekirdekler önbellek, bellek denetleyicisi ve veri yolu gibi kaynakları paylaşır. Bir çekirdekteki yazılım, diğer çekirdekteki yazılımın yürütme süresini bu paylaşılan kaynaklar üzerinden uzatabilir; bu yollara girişim kanalları (interference channels) denir. Çok çekirdekli işlemci kullanımına ilişkin otorite rehberliği (FAA AC 20-193, 2024 ve EASA AMC 20-193, 2022; ikisi daha önce kullanılan CAST-32A görüş belgesinin yerini almıştır), bu kanalların belirlenmesini, etkilerinin ölçülmesini ya da sınırlandırılmasını bekler. Tek çekirdekte yapılan WCET ölçümleri çok çekirdekli bir hedefe doğrudan taşınamaz.

- Girişim kanalları listelenmiş mi ve her biri için bir sınır, bir azaltma önlemi ya da ölçüm sonucu var mı?
- WCET değerleri, diğer çekirdekler en olumsuz yükü üretirken hedef donanımda belirlenmiş mi?
- İşlemcinin girişimi etkileyen ayarları (etkin çekirdekler, önbellek ve kaynak paylaşımı) tanımlı ve konfigürasyon kontrolünde mi?

Yapılandırma ve kullanılmayan özellikler

RTOS'un sertifikasyon paketi belirli bir yapılandırmayı, işlemciyi ve derleyici sürümünü kapsar. Proje bu yapılandırmanın dışına çıkarsa (başka bir özellik etkinleştirir, derleyici sürümünü değiştirir) paketin kanıtı o kısım için geçersiz kalır. Bağlanan ama kullanılmayan çekirdek özellikleri, kapsam analizinde devre dışı bırakılmış kod (deactivated code) ya da gereksiz kod (extraneous code) olarak karşınıza çıkar. Hangisi olduğunu özelliğin kullanılmaması değil, ardındaki veri belirler: özellik bir gereksinime izlenebiliyor ve tasarım gereği çalıştırılmıyorsa

devre dışı bırakılmış koddur; o zaman planlarda beyan edilmesi, etkin kod gibi standardın hedeflerine uygun geliştirilmiş olması ve istem dışı çalışmasının önlendiğinin analiz ile testle gösterilmesi gerekir. Hiçbir gereksinime izlenemiyorsa gereksiz koddur; DO-178C'nin öngördüğü çözüm kodun kaldırılması ve bunun etkisinin analiz edilmesidir. Kodu imajda tutmak isteyen proje, eksik gereksinimini ve doğrulama verisini tamamlayarak onu devre dışı bırakılmış kod olarak savunmak zorundadır ([17. Kapsanmayan Kodlar](#)).

- Projede kullanılan yapılandırma (etkin özellikler, işlemci, derleyici sürümü ve seçenekleri) paketin kapsamıyla madde madde karşılaştırılmış mı?
- Bağlanan ama kullanılmayan her çekirdek özelliği için sınıflandırma ve etkinleşemeyeceğinin gerekçesi kayıtlı mı?
- RTOS yapılandırma dosyaları konfigürasyon kontrolünde mi ve her değişiklik, değişiklik etki analizinden (change impact analysis) geçiyor mu?

Bu ekten akılda kalması gerekenler

- RTOS riskleri çoğunlukla nadir tetiklenen zamanlama ve kaynak etkileşimlerinden gelir; bu yüzden testle birlikte analiz gerekir.
- Her alan için soru aynıdır: en kötü durum ne, nasıl sınırlandı ve kanıtı nerede?
- Sayaç taşması, yığın taşması ve kuyruk taşması gibi "taşma" riskleri hesapla dışlanabilir; hesap yapılmadıysa risk açıktır. Aylar sonra ortaya çıkacak bir sayaç taşmasını olağan sürekli test göstermez, hesap gösterir.
- Alan bölümlenmesi ile zaman bölümlenmesi ayrı ayrı gösterilir; bellek koruması zaman etkisini önlemez.
- Çok çekirdekli bir hedef, tek çekirdekte kurulan zamanlama varsayımlarını geçersiz kılabilir; girişim kanalları belirlenip sınırlandırılmalıdır.
- Bağlanan ama kullanılmayan çekirdek özelliğinin devre dışı bırakılmış kod mu, gereksiz kod mu olduğunu gereksinime izlenebilirliği belirler; devre dışı bırakılmış kod için istem dışı çalışmanın önlendiği ayrıca gösterilir.

Ek C: Gerçek Zamanlı İşletim Sistemi Seçiminde Sorulacak Sorular

Bu ek, emniyet-kritik bir aviyonik proje için aday bir gerçek zamanlı işletim sistemini (real-time operating system, RTOS) değerlendirirken tedarikçiye ve ekibin kendisine sorulacak soruları gruplar hâlinde verir. Her soru için neden önemli olduğu ve iyi bir yanıtın nasıl görüldüğü belirtilir.

Seçimin neden bir risk yönetimi kararı olduğu [20. Gerçek Zamanlı İşletim Sistemleri](#) bölümünde anlatılır. Seçilen RTOS'u projede kullanırken izlenecek riskler için [Ek B](#)'ye bakınız.

Nasıl kullanılır?

Sorular bir puan tablosu değildir; her yanıt bir kanıta dayanmalıdır. "Evet, destekliyoruz" yanıtı, ancak bunu gösteren bir belge, ölçüm ya da sertifikasyon verisiyle birlikte geldiğinde değer taşır. Yanıtlar ve dayandıkları kanıtlar bir karar kaydında toplanır; böylece seçim, aylar sonra "neden bunu seçtik?" sorusu geldiğinde de savunulabilir kalır.

Zamanlama davranışı

Soru	Neden önemli	İyi yanıt
Çekirdek hizmetlerinin en kötü durum yürütme süreleri (worst-case execution time, WCET) hedef işlemci için yayımlanmış mı?	Zamanlama analizi ancak üst sınırlarla kurulabilir	Hedef işlemci ailesi ve önbellek ayarları için ölçüm ya da analiz raporu
Kesme gecikmesi ve kesmelerin kapalı kaldığı en uzun süre ne?	Tepki süresi bütçesine doğrudan girer	Belgelenmiş üst sınır ve ölçüm yöntemi
Hangi çizelgeleme (scheduling) modelleri var (öncelik tabanlı, zaman tetiklemeli)?	Mimarının hangi zamanlama yaklaşımıyla kurulacağını belirler	Projenin ihtiyacına uyan model ve analiz yöntemi
Öncelik terslenmesine karşı hangi protokol var?	Nadir ama ciddi gecikmelerin ana kaynağı	Öncelik kalıtımı ya da tavanı; davranışını gösteren test verisi
Zamanlayıcı tıki yapılandırılabilir mi; tıksız (tickless) çalışma var mı?	Zaman çözünürlüğü ve ek yük	Gerekçeli seçenekler ve her birinin etkisi

Bellek ve koruma

Soru	Neden önemli	İyi yanıt
Çalışma sırasında dinamik bellek ayırma gerekiyor mu?	Parçalanma ve tükenme riski	Tüm nesnelerin statik ya da başlatmada ayrılabilmesi
Bellek koruma birimi (memory protection unit, MPU) ya da bellek yönetim birimi (memory management unit, MMU) destekleniyor mu?	Görevlerin birbirinin alanına yazmasının önlenmesi	Görev ya da bölüm başına koruma ve ihlal tepkisi
Yığın taşması tespit ediliyor mu?	Sessiz bellek bozulmasını önler	Koruma bölgesi ya da donanım desteğiyle tespit, tanımlı tepki
Çekirdeğin bellek ayak izi ne?	Kaynak kısıtlı hedeflerde belirleyici	Yapılandırmaya göre belgelenmiş boyutlar

Bölümleme, çok çekirdek ve uygulama arayüzü

Soru	Neden önemli	İyi yanıt
Alan ve zaman bölümlemesi (spatial and temporal partitioning) sağlanıyor mu (örneğin ARINC 653 tarzı)?	Farklı yazılım seviyelerinin aynı işlemciyi paylaşması	Bölümleme mekanizması ve bunun doğrulama kanıtı
Bölümler arası haberleşme hangi mekanizmalarla yapılıyor?	Bölümleme iddiasının haberleşme kanalıyla delinmemesi	Sınırlı, yapılandırma ile tanımlanan kanallar
Çok çekirdekli işlemci desteği var mı; girişim kanalları için ne sunuluyor?	Çok çekirdekte zamanlama varsayımları değişir	Girişim analizi desteği, kaynak bölüştürme seçenekleri
Sağlık izleme (health monitoring) ve hata tepkileri yapılandırılabilir mi?	Hata durumunda öngörülebilir davranış	Olay türüne göre tanımlanabilen tepkiler
Uygulama arayüzü hangi standarda dayanıyor (ARINC 653 uygulama arayüzü APEX, bir POSIX alt kümesi ya da tedarikçiye özgü arayüz)?	Uygulama kodunun başka bir RTOS'a taşınabilirliğini ve tedarikçi bağımlılığını belirler	Desteklenen hizmetlerin, standarttan sapmaların ve standart dışı uzantıların açık listesi
Çekirdek çağrıları geçersiz parametrede ya da izin verilmeyen bağlamda (örneğin kesme servis rutininden bekleyen çağrı) nasıl davranıyor?	Uygulamadaki bir hata çekirdeği bozmamalıdır; gürbüzlük (robustness) kanıtı paketin içinde değilse o testler projeye kalır	Her çağrı için tanımlı hata dönüşü ve gürbüzlük test sonuçları

Bölümleme iddiasının hangi analiz ve testlerle savunulacağı [21. Yazılım Bölümlemesi](#) bölümünde anlatılır; buradaki sorular, tedarikçinin o kanıtın ne kadarını hazır getirdiğini ortaya çıkarmak içindir.

Sertifikasyon paketi

Soru	Neden önemli	İyi yanıt
Paket hangi yazılım seviyesine kadar DO-178C yaşam döngüsü verisi içeriyor?	Projenin seviyesini karşılamayan paket boşluk bırakır	Gereksinim, tasarım, kod, test, kapsam ve izlenebilirlik verisi
Paket bizim işlemcimizi, derleyici sürümümüzü ve yapılandırmamızı kapsıyor mu?	Kanıt yalnızca kapsadığı yapılandırma için geçerlidir	Açık bir kapsam listesi; farklılıklar için uyarlama hizmeti
Kart destek paketi (board support package, BSP) ve sürücüler paketin içinde mi?	Paket dışındaki her katmanın kanıtını proje üretir	Kapsamın katman katman belirtilmesi
Paket, entegratöre hangi doğrulama işlerini ve kullanım kısıtlarını bırakıyor?	Tedarikçinin kanıtı varsayımlara dayanır; varsayımları sağlamak ve hedef karttaki entegrasyonu doğrulamak projenin işidir	Varsayımların, kısıtların ve entegratöre kalan faaliyetlerin tek tek sayıldığı belge
Kullanmayacağımız çekirdek özellikleri nasıl kapatılıyor: imaja hiç bağlanmıyor mu, yapılandırmayla mı devre dışı kalıyor?	İmajda kalan ama kullanılmayan kod ya devre dışı bırakılmış kod (deactivated code) olarak gerekçelendirilir ya da gereksiz kod (extraneous code) tartışması açar	Özellğin imaja girmediğinin ya da etkinleşemeyeceğinin kanıtı; paketin bu yapılandırmayı kapsadığının beyanı
Seviye A hedefleniyorsa, bu seviyeye özgü ek kanıt paketin içinde mi?	Değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC) kanıtı Seviye B için hazırlanmış bir pakette bulunmak zorunda değildir; kaynak koda doğrudan izlenemeyen nesne kodunun belirlenmesi ve ek doğrulaması ise derleyiciye ve seçeneklerine bağlıdır	MC/DC sonuçları; projenin derleyici sürümü ve seçenekleriyle üretilen, kaynak koda doğrudan izlenemeyen nesne kodunun belirlendiğini ve ek doğrulamayla doğruluğunun gösterildiğini ortaya koyan kayıt
Daha önce hangi projelerde, hangi otoritelerle sertifikasyon geçti?	Paketin otorite gözden geçirmesinden geçmiş olması riski azaltır	Referans projeler (gizlilik sınırları içinde)
Paket, bir otorite tarafından yeniden kullanılabilir yazılım bileşeni olarak kabul edilmiş mi (örneğin FAA AC 20-148 kapsamında)?	Kabul edilmiş bileşenin kanıtı sonraki projelerde yeniden kullanılabilir; ancak kabul belirli bir kapsam ve kısıtlarla verilir, ürünün onayı anlamına gelmez	Kabul belgesi, kapsadığı yapılandırma ve entegratöre düşen yükümlülüklerin listesi

Soru	Neden önemli	İyi yanıt
Bilinen problemler listesi paylaşıyor mu ve ne sıklıkla güncelleniyor?	Önceden geliştirilmiş yazılımın çözülmemiş problem raporlarının projeye etkisi değerlendirilmek zorundadır; bilinen kusurlar emniyet analizine girer	Düzenli, sınıflandırılmış ve etki açıklamalı liste

Hazır bir RTOS, proje açısından önceden geliştirilmiş yazılımdır (previously developed software, PDS); kanıtının projeye nasıl taşınacağı ve boşlukların nasıl kapatılacağı [24. Yazılım Yeniden Kullanımı](#) bölümünde, imajda kalan kullanılmayan kodun nasıl sınıflandırılacağı [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#) bölümünde ele alınır.

Araçlar ve geliştirme ortamı

Soru	Neden önemli	İyi yanıt
Hangi derleyici ve hata ayıklayıcılarla çalışıyor?	Araç zinciri değişikliği paket kapsamını etkiler	Desteklenen sürümler listesi
Yapılandırma ya da kod üreten araçlar var mı; kalifikasyon verisi sunuluyor mu?	Bir süreci ortadan kaldıran, azaltan ya da otomatikleştiren ve çıktısı doğrulanmayan araç kalifikasyon gerektirir; kalifikasyon her projede o sistemin bağlamında yeniden gösterilir	Araç kalifikasyon seviyesine (tool qualification level, TQL) uygun, projede yeniden kullanılacak veri
Zamanlama ve iz (trace) analiz araçları var mı?	Doğrulama ve hata ayıklama yükü	Hedefte düşük ek yükü çalışan izleme
Hedef dışı (host) simülasyon ortamı var mı?	Erken geliştirme ve birim test	Hedefle farkları belgelenmiş simülasyon ortamı

Bir aracın hangi koşulda kalifikasyon gerektirdiği ve seviyesinin nasıl belirlendiği [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#) bölümünde anlatılır.

Tedarikçi, destek ve ticari koşullar

Soru	Neden önemli	İyi yanıt
Ürün ne kadar süre desteklenecek?	Aviyonik programların ömrü onlarca yıldır	Yazılı uzun dönem destek taahhüdü
Kaynak koda ve problem raporu geçmişine erişim koşulları ne?	Hata analizinde ve otorite sorularında gerekir	Sözleşmede tanımlı erişim hakları
Lisans modeli nedir (geliştirici, ürün, birim başına)?	Toplam maliyeti ve filoya yayılmayı etkiler	Açık ve öngörülebilir koşullar

Soru	Neden önemli	İyi yanıt
Tedarikçi ortadan kalkarsa ne olur?	Bağımlılık riski	Kaynak kod emaneti (escrow) ya da benzeri güvence

Kırmızı bayraklar

- En kötü durum değerleri yerine yalnızca ortalama performans rakamları veriliyorsa.
- "Sertifike edilmiş RTOS" deniyor ama hangi seviye, hangi işlemci ve hangi yapılandırma için olduğu söylenmiyorsa (sertifika hava aracına, motora ya da pervaneye verilir; yazılım tek başına sertifikaya edilmez, kurulu olduğu sistemin ya da ekipmanın parçası olarak ve yalnızca o sertifikasyon bağlamında onaylanır).
- Paketin entegratöre bıraktığı işler ve kullanım kısıtları yazılı olarak verilmiyorsa.
- Bilinen problemler listesi paylaşılmıyor ya da "hiç problem yok" deniyorsa.
- Sertifikasyon paketi ayrı satılıyor ve içeriği satın almadan önce incelenemiyorsa.

Deneme uygulaması

Karar vermeden önce hedefe yakın bir kartta küçük bir deneme uygulaması koşturmak, belgelerde görünmeyen davranışları erken ortaya çıkarır. Birkaç periyodik görev, bir kesme kaynağı ve paylaşılan bir kaynak yeterlidir. Ölçülecek ve gözlenecekler:

- görevler arası bağlam değiştirme (context switch) süresi ve titreşim (jitter),
- kesmeden göreve tepki süresi,
- paylaşılan kaynakta en uzun bekleme süresi,
- yük altında zaman sınırı kaçırma davranışı,
- birkaç çekirdek çağrısının geçersiz parametreye verdiği tepki,
- derleme ve hata ayıklama zincirinin ekibin araçlarıyla uyumu.

Bu ekten akılda kalması gerekenler

- Her yanıt bir kanıta dayanmalıdır; kanıtsız yanıt yanıt değildir.
- Sertifikasyon paketinin değeri, projenin seviyesini, işlemcisini, derleyicisini ve yapılandırmasını kapsadığı ölçüdedir.
- Paketin kapsamadığı her şey projenin işidir: entegratöre kalan doğrulama, kullanım kısıtları ve kullanılmayan özelliklerin kapatılma biçimi seçimden önce bilinmelidir.
- "Sertifika RTOS" ifadesi ya da bir otorite kabulü ürünün onayı değildir; yazılım, içinde çalıştığı sistemin parçası olarak onaylanır.

- Tedarikçi seçimi, program ömrü boyunca sürececek bir bağımlılıktır.
- Yanıtlar ve gerekçeler bir karar kaydında saklanmalıdır.

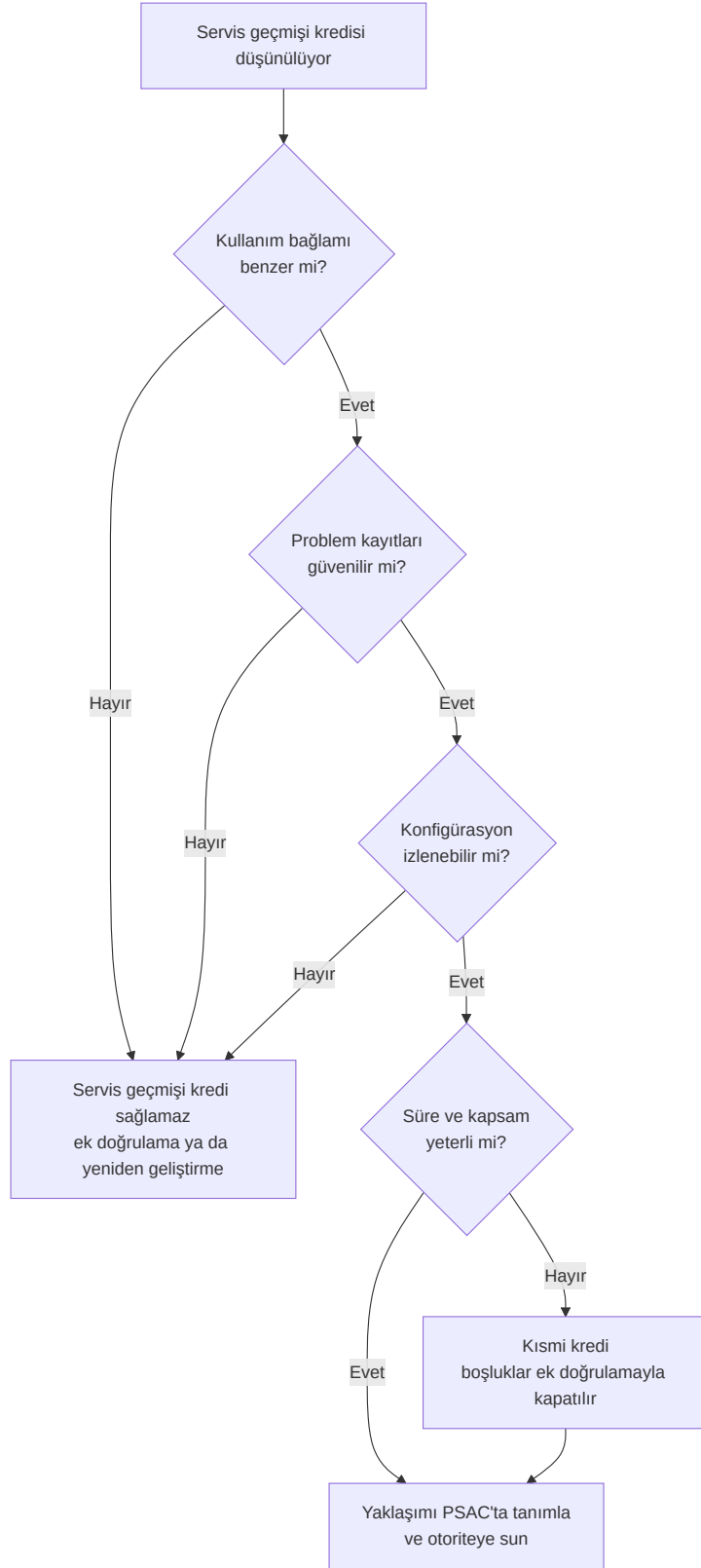
Ek D: Yazılım Servis Geçmişi Soruları

Bu ek, bir yazılımın sahadaki geçmişini sertifikasyon kredisi olarak öne sürmeden önce sorulması gereken soruları toplar. Sorular, ürün servis geçmişi (product service history) argümanının dayanması gereken dört koşula göre gruplanmıştır: benzer kullanım bağlamı, güvenilir problem kayıtları, konfigürasyon kararlılığı ve yeterli çalışma süresi.

Servis geçmişinin neden sanıldığından zor bir argüman olduğu [24. Yazılım Yeniden Kullanımı](#) bölümünde anlatılır. Bu liste, yazılım sertifikasyon planını (Plan for Software Aspects of Certification, PSAC) yazmadan önce dürüst bir ön eleme yapmak içindir. Dört koşul aynı ağırlıkta değildir: ilk üçü ön koşuldur, dördüncüsü kredinin ölçüsünü belirler. Kullanım bağlamı, problem kayıtları ya da konfigürasyon grubunda tatmin edici yanıt yoksa, argümana yatırım yapmadan önce başka bir güvence yolunu değerlendirmek daha ucuzdur.

DO-178C, ürün servis geçmişini alternatif yöntemler arasında (12.3.4) ele alır; geçmişin ilgililiğini, birikmiş miktarın yeterliliğini, serviste bulunan problemlerin toplanıp analiz edilmesini ve PSAC'ta bildirilecek bilgileri ayrı başlıklarda değerlendirir. Dört koşul ve buradaki sorular bu kitap için özgün olarak derlenmiştir; standardın başlıklarıyla bire bir örtüşmez. Daha ayrıntılı bir soru kümesi arayan okur, FAA'nın DO-178B döneminde yayımladığı Software Service History Handbook'a (DOT/FAA/AR-01/116) bakabilir.

Ön eleme akışı



İlk üç sorunun "Hayır" kolu krediyi sıfırlar, çünkü bağlamı farklı, kaydı güvenilir ya da sürümü belirsiz bir geçmişin neyi kanıtladığı söylenemez. Bağlam sorusu yine de ya hep ya hiç değildir: ortam yalnızca kısmen farklıysa standart, farkın hedef ortamda ek doğrulamayla kapatılmasını bekler; "Hayır" kolu karşılaştırılamayacak kadar farklı bir bağlam içindir. Süre ve kapsam ise derecelidir: geçmiş yalnızca gerçekten çalışmış işlev ve modlar için kanıt üretir, geri kalanı ek doğrulamayla kapatılır. İstenen kredi büyüdükçe gösterilmesi gereken geçmiş de büyür; standart bunun için sayısal bir eşik vermez. Bu yüzden kredi tam da olsa kısmi de olsa yaklaşım PSAC'ta tanımlanır ve sertifikasyon otoritesiyle erkenden konuşulur.

Kullanım bağlamı

- Yazılım sahada hangi sistemde, hangi işlev için kullanıldı?
- Yeni kullanımda işlev, arayüzler ve çalışma modları aynı mı? Farklıysa hangi kısımlar farklı?
- Ortam koşulları (sıcaklık, titreşim, elektromanyetik ortam, güç kesintileri) karşılaştırılabilir mi?
- Girdi aralıkları ve veri hızları yeni kullanımla aynı mı? Yeni kullanım, yazılımı daha önce hiç görmediği aralıklara itiyor mu?
- Önceki kullanımda yazılım seviyesi (software level) neydi; yeni kullanım daha yüksek bir seviye mi istiyor?
- Önceki kullanım yerde mi, havada mı, başka bir sektörde mi? (Yer sistemi geçmişi uçuş bağlamı için doğrudan kredi sağlamaz.)

Problem kayıtları

- Servis dönemi boyunca hatalar sistematik olarak raporlandı mı? Raporlama süreci yazılı mı ve kim işletti?
- Raporlar sınıflandırılmış mı; yazılım kaynaklı olanlar donanım ve operasyon kaynaklı olanlardan ayrılabilir mi?
- Kapatılan raporlar için kök neden ve düzeltme bilgisi var mı?
- "Hiç hata raporu yok" deniyorsa, bunun raporlama sisteminin işlemediği anlamına gelmediği nasıl gösterilecek?
- Kullanıcıların hatayı fark edip raporlama olasılığı ne? (Gözlenmesi zor hatalar, örneğin yanlış ama makul görünen bir değer, raporlara hiç yansımamış olabilir.)
- Servis döneminde emniyeti etkileyen bir problem ya da yazılıma bağlanan bir uçuşa elverişlilik direktifi (airworthiness directive) çıktı mı? Çıktıysa kök neden kredi istenen sürümde giderilmiş mi? (Standart, serviste görülen emniyetle ilgili bütün problemlerin düzeltildiğinin teyit edilmesini bekler.)

- Bulunan hatalar geliştirme sürecindeki bir yetersizliğe işaret ediyor mu? (Hep aynı arayüzde ya da hep sınır değerlerde çıkan hatalar, aynı türden bulunmamış başka hatalar olabileceğini düşündürür.)
- Kredi istenen sürümde açık kalan problem raporları hangileri; her birinin yeni kullanım bağlamındaki etkisi değerlendirildi mi? (Eski bağlamda zararsız sayılan bir kısıt yeni bağlamda emniyeti etkileyebilir.)
- Kayıtlara erişim hakkınız var mı; veri başka bir kuruluşun elindeyse sözleşmesel olarak kullanılabilir mi?

Konfigürasyon kararlılığı

- Kredi istenen sürüm tam olarak hangisi; sahadaki sürümlerle ilişkisi konfigürasyon kayıtlarıyla gösterilebiliyor mu?
- Servis dönemi boyunca kaç değişiklik yapıldı? Her değişiklikten sonra geçmiş saatlerin hangi kısmı hâlâ geçerli sayılabilir?
- Aynı kaynak koddan farklı derleyici, derleme seçeneği ya da hedef işlemciyle üretilmiş sürümler var mı?
- Konfigürasyon verisi ya da parametre verisi öğeleri (parameter data item, PDI) sahada değiştirildi mi? Değiştirse saatler hangi değer kümesiyle birikti? (Bkz. [22. Konfigürasyon Verisi](#).)
- Yeni kullanım için yazılımda değişiklik gerekiyor mu? Değişen kısım için servis geçmişi kredisi yoktur; değişiklik etki analizi (change impact analysis) gerekir. Analizin taradığı eksenler [10. Yazılım Konfigürasyon Yönetimi](#) bölümündedir.

Çalışma süresi ve kapsamı

- Süre, yazılımın çalışma biçimine uyan bir ölçüyle mi veriliyor: sürekli çalışan yazılım için uçuş saati, istek üzerine çalışan yazılım için istek sayısı?
- Toplam çalışma saati ne kadar; filo büyüklüğü ve kullanım profili ne?
- Çalışma saatleri nasıl ölçüldü; tahmin mi, kayıt mı?
- Saatler farklı yazılım sürümleri ya da donanım varyantları üzerinden toplanarak mı elde edildi? Öyleyse her birinin kredi istenen konfigürasyonu temsil ettiği nasıl gösteriliyor?
- Bu süre, yeni kullanımın yazılım seviyesine ve sistemin emniyet hedeflerine göre anlamlı mı? (Bu, yazılıma sayısal bir arıza olasılığı atamak demek değildir; olasılık hedefi sistem düzeyindeki arıza durumuna aittir.)
- Serviste gözlenen hata oranı (error rate) ne; dönem boyunca düşüyor mu, yoksa yeni hatalar aynı hızla mı çıkıyor? (Artan bir eğilimin açıklanması beklenir.)
- Kritik çalışma modları ve nadir durumlar (acil durum modları, yedeğe geçiş, aşırı yük) servis döneminde gerçekten tetiklendi mi, kaç kez?

- Serviste hiç çalışmamış işlevler ya da devre dışı bırakılmış kod (deactivated code) var mı; yeni kullanım bunları etkinleştirecek mi? (Çalışmamış kod için servis geçmişi kanıt üretmez.)
- Servis geçmişi hangi DO-178C hedefleri için kanıt olarak kullanılacak, hangileri için kullanılmayacak? (Servis geçmişi gözlenen davranışı kanıtlar; yapısal kapsam gibi hedeflerin yerini tam olarak tutmaz.)

Otoriteyle mutabakat

- Yaklaşım PSAC'ta açıkça tanımlandı mı: hangi sürüm, hangi hedefler için, hangi veriyle?
- Otoriteyle hangi aşamada konuşuldu? (Tercihen planlama aşamasında, SOI-1'den önce; bkz. [12. Sertifikasyon İrtibatı](#) ve [SW SOI-1](#).)
- Kabul edilebilir süre ve hata oranı için ölçüt önceden anlaşıldı mı? Neyin hata sayılacağı ve hangi tür problemin servis geçmişi geçersiz kılacağı PSAC'ta tanımlı mı?
- Servis geçmişinin kullanımı ve sonuçları sistem emniyet değerlendirme sürecinin önünden geçti mi?
- Servis geçmişinin karşılamadığı hedefler için hangi ek doğrulama faaliyetleri planlandı?

Sık yapılan hatalar

Hata	Sonuç
Toplam saati tek ölçüt olarak sunmak	Kritik modların hiç tetiklenmediği ortaya çıkınca argüman çöker
Kayıt sisteminin işleyişini göstermeden "hata yok" demek	Kanıt değeri sıfıra iner
Sürüm ilişkisini sonradan kurmaya çalışmak	Hangi geçmişin hangi sürüme ait olduğu tartışmalı kalır
Farklı sürüm ve varyantların saatlerini ayırmadan toplamak	Kredi istenen konfigürasyonun gerçek süresi bilinmez
Otoriteye proje sonunda gitmek	Reddedilen argüman için harcanan aylar boşa gider

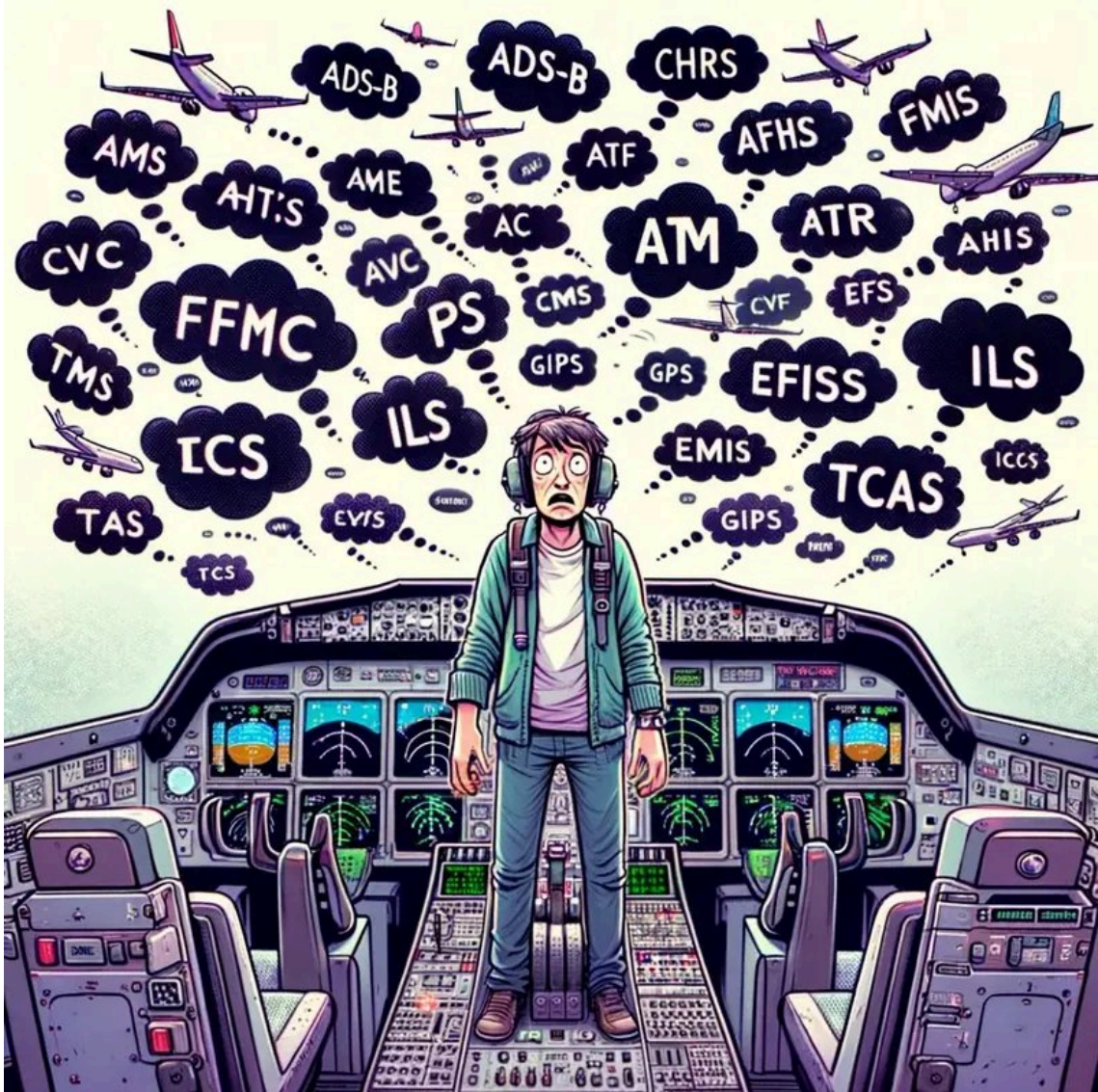
Bu ekten akılda kalması gerekenler

- Benzer kullanım bağlamı, güvenilir problem kayıtları ve konfigürasyon kararlılığı ön koşuldur; biri eksikse servis geçmişi kredi sağlamaz.
- Süre ve kapsam kredinin ölçüsünü belirler; yetersizse kredi kısımdır ve boşluk ek doğrulamayla kapatılır.

- Gereken geçmiş miktarı yazılım seviyesine ve sistemin emniyet hedeflerine, ortam farkına, servis geçmişiyle karşılanacak hedeflere ve eldeki öteki kanıta bağlıdır; hata oranı izlenir, yazılıma arıza olasılığı atanmaz.
- "Uzun süre sorunsuz çalıştı" bir kanıt değil, kanıt aranacak bir iddiadır.
- Servis geçmişi çoğu zaman tek başına bir yol değil, ek doğrulamayla birleşen destekleyici bir argümandır.
- Otoriteyle erken mutabakat, en pahalı riski (geç gelen reddi) ortadan kaldırır.

Kaynaklar

Kısaltmalar



Bu sayfa, kitapta ve aviyonik yazılım projelerinde karşılaşacağınız kısaltmaların başvuru sözlüğüdür. İlk tablo, kitabın DO-178C çekirdeğinde sık kullanılan kısaltmaları Türkçe karşılıkları ve ele alındıkları bölümle birlikte verir; ardından gelen A'dan Z'ye liste, havacılık elektronığının daha geniş alanındaki kısaltmaların İngilizce açıklamalarını içerir.

Her kısaltma tek satırdadır. Birden çok anlamı olan kısaltmalarda anlamlar noktalı virgülle ayrılır, kitapta geçen anlam başa yazılır ve gerekiyorsa bağlam parantez içinde belirtilir; "bkz." notu aynı kavramın daha yaygın kısaltmasına yönlendirir. Eksik ya da hatalı bir girdi görürseniz [ana sayfadaki katkı bölümü](#) üzerinden bildirebilirsiniz.

Kitapta kullanılan temel kısaltmalar

Tablo, kısaltmaların kitapta ele alındığı bölüm sırasına göre dizilmiştir; Türkçe karşılıklar bölümlerde kullanılan terimlerdir.

Kısaltma	İngilizce açılım	Türkçe karşılık	Bölüm
IMA	Integrated Modular Avionics	tümleşik modüler aviyonik	Bölüm 2
DAL	Development Assurance Level	geliştirme güvence seviyesi	Bölüm 3
FDAL	Function Development Assurance Level	fonksiyon geliştirme güvence seviyesi	Bölüm 3
IDAL	Item Development Assurance Level	öge geliştirme güvence seviyesi	Bölüm 3
FHA	Functional Hazard Assessment	fonksiyonel tehlike değerlendirme	Bölüm 3
PASA	Preliminary Aircraft Safety Assessment	ön uçak emniyet değerlendirme	Bölüm 3
PSSA	Preliminary System Safety Assessment	ön sistem emniyet değerlendirme	Bölüm 3
SSA	System Safety Assessment	sistem emniyet değerlendirme	Bölüm 3
ASA	Aircraft Safety Assessment	uçak emniyet değerlendirme	Bölüm 3
CCA	Common Cause Analysis	ortak neden analizi	Bölüm 3
PSAC	Plan for Software Aspects of Certification	yazılım sertifikasyon planı	Bölüm 5
SDP	Software Development Plan	yazılım geliştirme planı	Bölüm 5
SVP	Software Verification Plan	yazılım doğrulama planı	Bölüm 5
SCMP	Software Configuration Management Plan	yazılım konfigürasyon yönetimi planı	Bölüm 5
SQAP	Software Quality Assurance Plan	yazılım kalite güvencesi planı	Bölüm 5
HLR	High-Level Requirements	yüksek seviyeli gereksinimler	Bölüm 6
LLR	Low-Level Requirements	düşük seviyeli gereksinimler	Bölüm 7
EOC	Executable Object Code	çalıştırılabilir nesne kodu	Bölüm 8
SVCP	Software Verification Cases and Procedures	yazılım doğrulama durumları ve prosedürleri	Bölüm 9
SVR	Software Verification Results	yazılım doğrulama sonuçları	Bölüm 9
SCA	Structural Coverage Analysis	yapısal kapsam analizi	Bölüm 9
MC/DC	Modified Condition/Decision Coverage	değiştirilmiş koşul/karar kapsama	Bölüm 9
WCET	Worst-Case Execution Time	en kötü durum yürütme süresi	Bölüm 9
PR	Problem Report	problem raporu	Bölüm 9
CC1	Control Category 1	kontrol kategorisi 1	Bölüm 10

Kısaltma	İngilizce açılım	Türkçe karşılık	Bölüm
CC2	Control Category 2	kontrol kategorisi 2	Bölüm 10
SCI	Software Configuration Index	yazılım konfigürasyon indeksi	Bölüm 10
SECI	Software Life Cycle Environment Configuration Index	yazılım yaşam döngüsü ortam konfigürasyon indeksi	Bölüm 10
QA	Quality Assurance	kalite güvencesi	Bölüm 11
SAS	Software Accomplishment Summary	yazılım başarı özeti	Bölüm 12
SOI	Stage of Involvement	katılım aşaması	Bölüm 12
TQL	Tool Qualification Level	araç kalifikasyon seviyesi	Bölüm 13
TOR	Tool Operational Requirements	araç operasyonel gereksinimleri	Bölüm 13
TQP	Tool Qualification Plan	araç kalifikasyon planı	Bölüm 13
TAS	Tool Accomplishment Summary	araç başarı özeti	Bölüm 13
FLS	Field-Loadable Software	sahada yüklenebilir yazılım	Bölüm 18
LSP	Loadable Software Part	yüklenebilir yazılım parçası	Bölüm 18
UMS	User-Modifiable Software	kullanıcı tarafından değiştirilebilir yazılım	Bölüm 19
RTOS	Real-Time Operating System	gerçek zamanlı işletim sistemi	Bölüm 20
MMU	Memory Management Unit	bellek yönetim birimi	Bölüm 21
MPU	Memory Protection Unit	bellek koruma birimi	Bölüm 21
PDI	Parameter Data Item	parametre verisi ögesi	Bölüm 22
DQR	Data Quality Requirements	veri kalite gereksinimleri	Bölüm 23
DPAL	Data Process Assurance Level	veri süreci güvence seviyesi	Bölüm 23

Kısaltma	İngilizce açılım	Türkçe karşılık	Bölüm
PDS	Previously Developed Software	önceden geliştirilmiş yazılım	Bölüm 24
COTS	Commercial Off-The-Shelf	ticari hazır yazılım	Bölüm 24

A

Kısaltma	Açıklama
A/A	Air-to-Air
A/C	Aircraft
A/D	Analog to Digital
A/G	Air-to-Ground
A/P	Autopilot
A429	ARINC 429
ABAS	Aircraft-Based Augmentation System
ABROAD	ADS Broadcast
AC	Advisory Circular (FAA)
ACARS	Aircraft Communications Addressing and Reporting System
ACAS	Airborne Collision Avoidance System
ACC	Area Control Center
ACE	Actuator Control Electronics
ACMS	Aircraft Condition Monitoring System
ACO	Aircraft Certification Office (FAA; 2023 yeniden yapılanmasıyla Certification Branch)
ACP	Audio Control Panel
ACS	Audio Control System
ACSL	ANSI/ISO C Specification Language (C için biçimsel belirtim dili)
AD	Airworthiness Directive
ADAHRS	Air Data and Attitude Heading Reference System
ADC	Air Data Computer; Analog-to-Digital Converter
ADF	Automatic Direction Finder

Kısaltma	Açıklama
ADI	Attitude Director Indicator
ADIRS	Air Data Inertial Reference System
ADIRU	Air Data Inertial Reference Unit
ADM	Air Data Module
ADR	Air Data Reference (ADIRU'nun hava verisi kısmı)
ADS	Automatic Dependent Surveillance
ADS-A	Automatic Dependent Surveillance-Addressed (güncel adı ADS-C)
ADS-B	Automatic Dependent Surveillance-Broadcast
ADS-C	Automatic Dependent Surveillance-Contract
ADSB	Alternating Double Sideband
AEH	Airborne Electronic Hardware
AESA	Active Electronically Scanned Array
AFCS	Automatic Flight Control System
AFD	Autopilot Flight Director
AFDC	Autopilot Flight Director Computer
AFDS	Autopilot Flight Director System
AFDX	Avionics Full-Duplex Switched Ethernet
AFHA	Aircraft Functional Hazard Assessment (uçak seviyesi FHA)
AGACS	Automatic Ground-Air Communication System
AGC	Automatic Gain Control
AGDL	Air-Ground Data Link
AHC	Attitude Heading Computer
AHRS	Attitude Heading Reference System
AI	Action Item
AIDS	Aircraft Integrated Data System
AIM	Aeronautical Information Manual
AIP	Aeronautical Information Publication
AIR	Aerospace Information Report
AIRAC	Aeronautical Information Regulation and Control
AIS	Aeronautical Information Service

Kısaltma	Açıklama
AL	Assurance Level
ALT	Altitude
ALU	Arithmetic Logic Unit
AM	Amplitude Modulation
AMC	Acceptable Means of Compliance (EASA); Alternative Means of Compliance
AMS	Air Management System
ANC	Active Noise Cancellation
ANN	Annunciator
ANR	Active Noise Reduction
ANSI	American National Standards Institute
ANT	Antenna
AOC	Aeronautical Operational Control
AOP	Airport Operating Plan
AOPA	Aircraft Owners and Pilots Association
APARS	Automatic Pressure Altitude Reporting System
APEX	Application/Executive (ARINC 653 uygulama arayüzü)
API	Application Programming Interface
APS	Auto Pilot System
APU	Auxiliary Power Unit
APV	Approach with Vertical Guidance
ARINC	Aeronautical Radio, Incorporated
ARM	Advanced RISC Machine
ARP	Aerospace Recommended Practice (SAE International)
ARW	Airborne Rotary Winged
ASA	Aircraft Safety Assessment
ASCII	American Standard Code for Information Interchange
ASD	Aircraft Situation Display
ASDL	Aeronautical Satellite Data Link
ASIC	Application Specific Integrated Circuit
ASIL	Automotive Safety Integrity Level (ISO 26262)

Kısaltma	Açıklama
ASR	Airport Surveillance Radar
ASSP	Application Specific Standard Product
ASTERIX	All Purpose Structured Eurocontrol Surveillance Information Exchange
ASU	Avionics Switching Unit
AT	Auto Throttle
ATC	Air Traffic Control; Amended Type Certificate
ATCRBS	Air Traffic Control Radar Beacon System
ATCSS	Air Traffic Control Signaling System
ATCT	Airport Traffic Control Tower
ATE	Automatic Test Equipment
ATIS	Automatic Terminal Information Service
ATM	Air Traffic Management
ATP	Acceptance Test Procedure
ATS	Air Traffic Service
ATSAW	Airborne Traffic Situational Awareness
ATSU	Air Traffic Services Unit
ATT	Attitude
Avionics	Aviation Electronics
AVN	Office of Aviation System Standards (FAA; eski birim adı)
AWACS	Airborne Warning and Control System
AWOS	Automated Weather Observing System

B

Kısaltma	Açıklama
B-RNAV	Basic Area Navigation
BARO	Barometric Indication, Setting or Pressure
BC	Bus Controller
BCD	Binary Coded Decimal

Kısaltma	Açıklama
BDI	Bearing Distance Indicator
BFO	Beat Frequency Oscillator
BGAN	Broadband Global Area Network
BIT	Built-In Test
BITE	Built-In Test Equipment
BNR	Binary
BOM	Bill of Materials
BRAM	Block RAM
BSP	Board Support Package

C

Kısaltma	Açıklama
CA	Criticality Analysis
CAA	Civil Aviation Authority; Civil Aeronautics Administration (ABD, tarihsel; FAA'nın öncülü)
CAS	Calibrated Airspeed
CAST	Certification Authorities Software Team (artık etkin değil)
CAT	Catastrophic (emniyet bağlamında); Category
CAT I	Operational Performance Category I
CAT II	Operational Performance Category II
CAT IIIa	Operational Performance Category IIIa
CAT IIIb	Operational Performance Category IIIb
CAT IIIc	Operational Performance Category IIIc
CBIT	Continuous Built-In Test
CC	Control Category (DO-178C; bkz. CC1, CC2); Change Control
CC1	Control Category 1
CC2	Control Category 2
CCA	Common Cause Analysis (ARP4761A); Circuit Card Assembly
CCB	Change Control Board

Kısaltma	Açıklama
CDA	Continuous Descent Approach
CDI	Course Deviation Indicator
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CDTI	Cockpit Display of Traffic Information
CDU	Control Display Unit
CEA	Cascading Effects Analysis (ARP4761A)
CEH	Complex Electronic Hardware
CFIT	Controlled Flight Into Terrain
CFR	Code of Federal Regulations
CI	Configuration Item
CID	Configuration Index Document
CIDS	Cabin Intercommunication Data System
CII	Configuration Item Index
CM	Configuration Management
CMA	Common Mode Analysis
CME	Configuration Management Engineer
CMOS	Complementary Metal-Oxide Semiconductor
CMP	Configuration Management Plan
CMR	Certification Maintenance Requirement
CMS	Configuration Management Specialist
CNS	Communication, Navigation, Surveillance
CNS/ATM	Communication, Navigation, Surveillance / Air Traffic Management
CoC	Certificate of Conformity
CODEC	Coder/Decoder
COM	Communication
COM/MON	Command/Monitor (komut/izleme mimarisi)
COMM	Communications Receiver
COTS	Commercial Off-The-Shelf
CPDLC	Controller-Pilot Data Link Communications

Kısaltma	Açıklama
CPS	Cycles per Second
CPU	Central Processing Unit
CR	Change Request
CRC	Cyclic Redundancy Check; Cyclic Redundancy Code
CRI	Certification Review Item
CRT	Cathode Ray Tube
CS	Certification Specification
CSCI	Computer Software Configuration Item
CSV	Comma Separated Values
CTAF	Common Traffic Advisory Frequency
CV/DFDR	Cockpit Voice and Digital Flight Data Recorder
CVE	Compliance Verification Engineer
CVR	Cockpit Voice Recorder
CW	Continuous Wave
CWS	Control Wheel Steering

D

Kısaltma	Açıklama
DA	Decision Altitude; Drift Angle
DAC	Digital-to-Analog Converter
DAL	Development Assurance Level (ARP4754B; fonksiyon için FDAL, öge için IDAL); Design Assurance Level (DO-254)
DAPs	Downlink of Aircraft Parameters
DAR	Designated Airworthiness Representative
DAT	Data Services (EASA Part-DAT; veri hizmetleri sağlayıcıları)
DB	Database
DC	Direct Current; Decision Coverage
DCDU	Data Link Control and Display Unit
DCN	Document Change Notice

Kısaltma	Açıklama
DCP	Display Control Panel
DD	Dependence Diagram (ARP4761A); Digital Design
DDC	Digital Down Conversion
DDM	Difference in Depth of Modulation
DDP	Declaration of Design and Performance
DDR	Double Data Rate
DDS	Display Docking Station
DER	Designated Engineering Representative
DFMC	Dual Frequency Multiple Constellation
DG	Directional Gyroscope
DGPS	Differential Global Positioning System
DH	Decision Height
DIO	Discrete Input/Output
DISC	Discrete
DITS	Digital Information Transfer System
DL	Data Link
DLR	Data Link Recorder
DMA	Direct Memory Access
DME	Distance Measuring Equipment
DMIR	Designated Manufacturing Inspection Representative
DNC	Direct Noise Canceling
DO	RTCA Document (RTCA belge numarası öneki, ör. DO-178C; "Document Order" açılımı da anılır)
DOA	Design Organisation Approval (EASA)
DOD	Department of Defense
DOID	Design Organisation Interface Document
DP	Departure Procedures
DPAL	Data Process Assurance Level (DO-200B)
DQR	Data Quality Requirements
DRAM	Dynamic Random Access Memory
DRFS	Direct RF Sampling

Kısaltma	Açıklama
DSP	Digital Signal Processor
DUAT	Direct User Access Terminal (FAA; hizmet 2018'de sonlandırıldı)
DVE	Degraded Visual Environment
DVOR	Doppler VHF Omnidirectional Range
DWG	Drawing

E

Kısaltma	Açıklama
EADI	Electronic Attitude Director Indicator
EASA	European Union Aviation Safety Agency (2018'e kadar European Aviation Safety Agency)
EATP	Equipment Acceptance Test Procedure
EATR	Equipment Acceptance Test Report
ECN	Engineering Change Notice
ECU	Engine Control Unit (bkz. EEC)
ED	EUROCAE Document (EUROCAE belge numarası öneki; ör. ED-12C)
EDD	Equipment Design Description
EDL	Equipment Development Laboratory
EEC	Electronic Engine Control
EEPROM	Electrically Erasable Programmable Read-Only Memory
EFB	Electronic Flight Bag
EFD	Electronic Flight Display
EFIS	Electronic Flight Instrument System
EGPWS	Enhanced Ground Proximity Warning System
EGT	Exhaust Gas Temperature
EHS	Enhanced Surveillance
EHSI	Electronic Horizontal Situation Indicator
EIA	Electronic Industries Alliance (eski adı Electronic Industries Association; kurum kapanmıştır)
EICAS	Engine Indicating and Crew Alerting System

Kısaltma	Açıklama
EIRD	Equipment Installation Requirement Document
EIRP	Equivalent Isotropically Radiated Power
ELT	Emergency Locator Transmitter
EMC	Electromagnetic Compatibility
EMCON	Emission Control
EMI	Electromagnetic Interference
EMS	Emergency Medical Service
ENC	Electronic Noise Canceling
ENG	Engine
ENR	Electronic Noise Reduction
EOC	Executable Object Code
EPR	Engine Pressure Ratio
EPROM	Erasable Programmable Read-Only Memory
ESD	Electrostatic Discharge
ETOPS	Extended-range Twin-engine Operational Performance Standards (güncel kullanım: Extended Operations)
ETSO	European Technical Standard Order
EUROCAE	European Organisation for Civil Aviation Equipment
EUT	Equipment Under Test
eVTOL	Electric Vertical Take-Off and Landing
EVTP	Equipment Verification Test Procedure
EVTR	Equipment Verification Test Report

F

Kısaltma	Açıklama
FAA	Federal Aviation Administration
FADEC	Full Authority Digital Engine Control
FAI	First Article Inspection
FANS	Future Air Navigation System

Kısaltma	Açıklama
FAR	Federal Aviation Regulation
FAS	Final Approach Segment
FBW	Fly-By-Wire
FC	Failure Condition
FCC	Flight Control Computer; Failure Condition Classification
FCR	Final Certification Review
FCS	Flight Control System
FD	Flight Director
FDAL	Function Development Assurance Level
FDE	Fault Detection and Exclusion
FDPS	Flight Data Processing System
FDR	Flight Data Recorder
FDRS	Flight Data Recorder System
FDU	Flux Detector Unit
FE	Front-End
FF	Fuel Flow
FFR	First Flight Review
FFS	Functional Failure Set
FG	Flight Guidance
FHA	Functional Hazard Assessment
FIC	Flight Information Centre
FIFO	First In, First Out
FIS	Flight Information Service
FIS-B	Flight Information Service - Broadcast
FL	Flight Level
FLIR	Forward-Looking Infra-Red
FLS	Field-Loadable Software
FLTA	Forward Looking Terrain Awareness
FM	Frequency Modulation; Formal Methods (DO-333)
FMA	Flight Mode Annunciator

Kısaltma	Açıklama
FMEA	Failure Modes and Effects Analysis
FMECA	Failure Mode, Effects and Criticality Analysis
FMES	Failure Modes and Effects Summary
FMGS	Flight Management and Guidance System
FMS	Flight Management System
FOB	Fuel On-Board
FPA	Front Panel Assembly
FPE	Floating Point Emulation
FPGA	Field Programmable Gate Array
FPU	Floating-Point Unit
FRACAS	Failure Reporting and Corrective Action System
FREQ	Frequency
FRS	Firmware Requirements Specification
FSS	Flight Service Station
FT	Functional Test
FTA	Fault Tree Analysis
FTP	File Transfer Protocol; Firmware Test Procedure
FW	Firmware; Failure Warning
FWS	Flight Warning System
FYDS	Flight Director/Yaw Damper System

G

Kısaltma	Açıklama
G/A	Ground-to-Air
G/S	Glideslope (bkz. GS)
GA	General Aviation; Go-Around
GAST	GBAS Approach Service Type
GB	Gigabyte

Kısaltma	Açıklama
GBAS	Ground Based Augmentation System
GCAS	Ground Collision Avoidance System
GCU	Generator Control Unit
GDOP	Geometric Dilution of Precision
GGs	Global Positioning System Ground Station
GHz	Gigahertz
GLNS	GPS Landing and Navigation System
GLNU	GPS Landing and Navigation Unit
GLONASS	Global Navigation Satellite System (Rusya'nın uydu seyrüsefer sistemi)
GLS	GNSS Landing System
GLU	GPS Landing Unit
GMT	Greenwich Mean Time
GND	Ground
GNSS	Global Navigation Satellite System
GO	Go-Around (bkz. GA)
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GPWC	Ground Proximity Warning Computer
GPWS	Ground Proximity Warning System
GS	Ground Speed; Glideslope (G/S olarak da yazılır); Ground Station

H

Kısaltma	Açıklama
HAS	Hardware Accomplishment Summary
HAZ	Hazard or Hazardous
HBLP	Hardware Build and Load Control Procedure
HCDD	Hardware Conceptual Design Data
HCI	Hardware Configuration Index

Kısaltma	Açıklama
HCM	Hardware Configuration Management
HCMP	Hardware Configuration Management Plan
HDDD	Hardware Detailed Design Data
HDG	Heading
HDG SEL	Heading Select
HDL	Hardware Description Language
HDOP	Horizontal Dilution of Precision
HDP	Hardware Design Plan; Hardware Development Plan
HDStd	Hardware Design Standards
HF	High Frequency
HHLD	Heading Hold
HIRF	High Intensity Radiated Field
HLR	High-Level Requirements
HMD	Helmet-Mounted Display
HMI	Human Machine Interface
HOL	High-Order Language
HPAP	Hardware Process Assurance Plan
HPR	High Pressure Rotor
HRD	Hardware Requirements Document
HRStd	Hardware Requirements Standards
HSD	High-Speed Data
HSI	Horizontal Situation Indicator
HSL	Heading Select
HTAWS	Helicopter Terrain Awareness and Warning System
HTP	Hardware Test Procedure
HTR	Hardware Test Results
HUD	Head-Up Display
HUMS	Health and Usage Monitoring Systems
HVVP	Hardware Validation and Verification Plan
HVVStd	Hardware Validation and Verification Standards

Kısaltma	Açıklama
HW	Hardware
Hz	Hertz

I

Kısaltma	Açıklama
I	Input
I/O	Input/Output
I2S	Inter-IC Sound
IA	Internal Audit
IAS	Indicated Airspeed
IBIT	Initiated Built-In Test
ICA	Instructions for Continued Airworthiness
ICAO	International Civil Aviation Organization
ICD	Interface Control Document
ICE	In-Circuit Emulator
ICNI	Integrated Communication, Navigation and Identification System
ICS	Internal Communication System
ID	Identify, Identification or Identifier
IDAL	Item Development Assurance Level
IDD	Interface Design Description
IDE	Integrated Development Environment
IDENT	Identify, Identifier, or Identification
IDS	Information Display System; Integrated Display System
IFE	In-Flight Entertainment
IFF	Identification Friend or Foe
IFICS	Integrated Flight Instrument and Control System
IFR	Instrument Flight Rules
IHA	Intrinsic Hazard Analysis

Kısaltma	Açıklama
ILS	Instrument Landing System; Integrated Logistics Support
ILSP	Integrated Logistics Support Plan
IMA	Integrated Modular Avionics
IMC	Instrument Meteorological Conditions
IND	Indicator
INS	Inertial Navigation System
IOC	Initial Operational Capability
IOMMU	Input/Output Memory Management Unit
IP	Internet Protocol; Intellectual Property (ör. FPGA IP çekirdeği); Issue Paper (FAA)
IPV	Instrument Procedure with Vertical Guidance (güncel adı APV)
IQ	In-phase/Quadrature
IRS	Inertial Reference System; Interface Requirements Specification
ISA	International Standard Atmosphere
ISIS	Integrated Standby Instrument System
ISO	International Organization for Standardization
ISP	Integrated Switching Panel
ISR	Interrupt Service Routine
IT	Information Technology
ITT	Interstage Turbine Temperature
ITU	International Telecommunication Union
IVSI	Instantaneous Vertical Speed Indicator

J

Kısaltma	Açıklama
JAA	Joint Aviation Authorities (2009'da kapandı; görevlerini EASA devraldı)
JASC	Joint Aircraft System/Component Code
JSF	Joint Strike Fighter (JSF AV C++: Joint Strike Fighter Air Vehicle C++ Coding Standards)
JTAG	Joint Test Action Group

Kısaltma	Açıklama
JTIDS	Joint Tactical Information Distribution System

K

Kısaltma	Açıklama
KB	Kilobyte
kHz	Kilohertz
KIAS	Knots Indicated Airspeed
KT	Knot (saatte bir deniz mili)
KTAS	Knots True Airspeed

L

Kısaltma	Açıklama
LAAS	Local Area Augmentation System (FAA terimi; güncel uluslararası terim GBAS)
LADGPS	Local Area Differential GPS
LCD	Liquid Crystal Display
LDGPS	Local Area Differential Global Positioning System (bkz. LADGPS)
LED	Light-Emitting Diode
LF	Low Frequency
LLR	Low-Level Requirements
LMM	Locator Middle Marker
LO	Local Oscillator
LOA	Letter of Acceptance (FAA AC 20-153B; havacılık verisi süreçleri); Letter of Agreement; Letter of Authorization
LOC	Localizer
LODA	Letter of Design Approval (FAA; TSO tasarım onay mektubu); Letter of Deviation Authority
LOI	Level of Involvement
LOM	Locator Outer Marker
LORAN	Long-Range Navigation

Kısaltma	Açıklama
LPR	Low Pressure Rotor
LPV	Localizer Performance with Vertical Guidance
LRM	Line Replaceable Module
LRU	Line Replaceable Unit
LSB	Least Significant Bit
LSP	Loadable Software Part (ARINC 665); Liskov Substitution Principle (DO-332)
LTE	Loss of Tail Rotor Effectiveness (helikopter)
LTL	Linear Temporal Logic
LVDS	Low-Voltage Differential Signaling

M

Kısaltma	Açıklama
M1553	MIL-STD-1553
MA	Markov Analysis
MAP	Manifold Absolute Pressure; Missed Approach Point
MAPS	Minimum Aviation Performance Standards
MASPS	Minimum Aviation System Performance Standard
MB	Marker Beacon; Megabyte
MBD	Model-Based Development
MBSA	Model-Based Safety Analysis
MC/DC	Modified Condition/Decision Coverage
MCBF	Mean Cycles Between Failures
MCDU	Multi-Purpose/Multi-Function Control and Display Unit
MCW	Modulated Continuous Wave
MDA	Minimum Descent Altitude
MEL	Minimum Equipment List
MF	Medium Frequency
MFD	Multi-Function Display

Kısaltma	Açıklama
MFDS	Multi-Function Display System
MFR	Manufacturer
MIC	Microphone
MIDO	Manufacturing Inspection District Office (FAA; 2023 yeniden yapılanmasıyla Certificate Management Section)
MIDS	Multifunctional Information Distribution System
MIL-STD	Military Standard
MILSPEC	Military Specification
MIO	Multiplexed I/O
MISRA	Motor Industry Software Reliability Association (MISRA C kodlama kılavuzu)
MKP	Multi-Function Keypad
MKR	Marker Beacon (bkz. MB)
MLS	Microwave Landing System
MM	Middle Marker
MMD	Moving Map Display
MMEL	Master Minimum Equipment List
MMH	Maintenance Man-Hours
MMR	Multi-Mode Receiver
MMU	Memory Management Unit
MNPS	Minimum Navigation Performance Specifications
MntR	Maintainability Report
MOA	Military Operations Area
MoC	Means of Compliance
Mode A	Transponder Pulse-Code Reporting
Mode C	Transponder Code and Altitude Reporting
Mode S	Transponder Code, Altitude, and TCAS Reporting
MOPS	Minimum Operational Performance Standards
MOSArt	Modular Open System Architecture
MPE	Minimum Processing Environment
MPS	Minimum Performance Standard

Kısaltma	Açıklama
MPU	Memory Protection Unit
MRAM	Magnetoresistive Random Access Memory
MSA	Minimum Safe Altitude
MSB	Most Significant Bit
MSG	Message
MSG-3	Maintenance Steering Group-3
MSP	Mode S Specific Protocol
MSSS	Mode S-Specific Services
MTBF	Mean Time Between Failures
MTBUR	Mean Time Between Unscheduled Removals
MTTF	Mean Time to Failure
MTTR	Mean Time to Repair
MVA	Minimum Vectoring Altitude
MVFR	Marginal Visual Flight Rules

N

Kısaltma	Açıklama
NA	Not Applicable
NACO	National Aeronautical Charting Office (FAA; eski birim adı)
NAS	U.S. National Airspace System
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
NAV	Navigation; Navigation Receiver
NAVAID	Navigational Aid
NAVCOMM	Navigation and Communications Equipment or Receiver
NAVSTAR GPS	Navigation Satellite Timing and Ranging Global Positioning System
NCATT	National Center for Aircraft Technician Training
NCD	No Computed Data

Kısaltma	Açıklama
ND	Navigation Display; Not Defined
NDB	Non-Directional Beacon
NFF	No Fault Found
NFPO	National Flight Procedures Office (FAA; eski birim adı)
NIMA	National Imagery and Mapping Agency (eski ad; 2003'ten beri NGA, National Geospatial-Intelligence Agency)
NM (NMI)	Nautical Mile(s)
NO	Normal Operation
NOAA	National Oceanic and Atmospheric Administration
NOTAM	Notice to Airmen
NPA	Non-Precision Approach
NPRM	Notice of Proposed Rulemaking
NTAP	Notice to Airmen Publication
NTSB	National Transportation Safety Board
NVD	Night Vision Device
NVG	Night Vision Goggles
NVRAM	Non-Volatile Random-Access Memory
NWS	National Weather Service

O

Kısaltma	Açıklama
O	Output
OAT	Outside Air Temperature
OBS	Omnibearing Selector
OCC	Operations Control Center
ODA	Organization Designation Authorization
OEM	Original Equipment Manufacturer
OM	Outer Marker
OMP	Obsolescence Management Plan

Kısaltma	Açıklama
OOT	Object-Oriented Technology
OOTiA	Object-Oriented Technology in Aviation (FAA el kitabı)
OPR	Open Problem Report
OS	Operating System
OWE/OEW	Operating Weight Empty/Operating Empty Weight

P

Kısaltma	Açıklama
P	Parity
P-Code	GPS Precision Code
P-RNAV	Precision Area Navigation
PA	Public Address (System); Process Assurance (DO-254)
PAC	Process Asset Checklist
PAL	Pilot Activated Lighting
PAM	Pulse Amplitude Modulation; Power Amplifier Module
PAPI	Precision Approach Path Indicator
PAR	Precision Approach Radar
PASA	Preliminary Aircraft Safety Assessment
PBIT	Power-up (Power-on) Built-In Test
PC	Personal Computer
PCAP	Processor Configuration Access Port
PCB	Printed Circuit Board
PCL	Pilot Controlled Lighting
PCN	Product Change Notice
PD	Profile Descent; Process Definition
PDI	Parameter Data Item
PDL	Product Development Leader
PDOP	Position Dilution of Precision

Kısaltma	Açıklama
PDR	Preliminary Design Review
PDS	Previously Developed Software
PESA	Passive Electronically Scanned Array
PFD	Primary Flight Display
PFDE	Predicted Fault Detection and Exclusion
PHAC	Plan for Hardware Aspects of Certification
PIDS	Prime Item Development Specification
PL	Programmable Logic
PLD	Programmable Logic Device
PLL	Phase Locked Loop
PMA	Parts Manufacturer Approval
PMG	Permanent Magnet Generator
PMP	Project Management Plan
PMU	Power Management Unit
PND	Primary Navigation Display
PNR	Passive Noise Reduction
POA	Production Organization Approval
POF	Phase of Flight
POH	Pilot's Operating Handbook
POS	Position
POSIX	Portable Operating System Interface
PPS	Pulse Pair Spacing
PR	Problem Report (DO-178C); Peer Review
PRA	Particular Risk Analysis (ARP4761A); Pre-Recorded Announcement
PRF	Pulse Repetition Frequency
PS	Processing System
PSAA	Plan for Software Aspects of Approval (DO-278A)
PSAC	Plan for Software Aspects of Certification
PSCP	Project Specific Certification Plan
PSP	Partnership for Safety Plan

Kısaltma	Açıklama
PSR	Primary Surveillance Radar
PSSA	Preliminary System Safety Assessment
PSU	Passenger Service Unit
PTN	Problem Tracking Number
PTR	Program Trouble Report
PTS	Purchaser Technical Specification
PTT	Push-To-Talk
PU	Processor Unit

Q

Kısaltma	Açıklama
QA	Quality Assurance
QAR	Quick Access Recorder
QE	Quality Engineer
QEC	Quadrantal Error Correction
QM	Quality Management
QMP	Quality Management Plan
QMS	Quality Management System
QNH	Deniz seviyesine indirgenmiş barometrik basınç (Q kodu; altimetre ayarı)
QP	Qualification Plan
QPL	Qualified Product List
QRH	Quick Reference Handbook
QSPI	Quad Serial Peripheral Interface
QTP	Qualification Test Procedure
QTR	Qualification Test Report

R

Kısaltma	Açıklama
R-NAV	Area Navigation (bkz. RNAV)
RA	Resolution Advisory (TCAS)
Rad Alt	Radio Altitude (Radio Altimeter)
RAI	Radio Altimeter Indicator
RAIM	Receiver-Autonomous Integrity Monitoring
RALT	Radar or Radio Altimeter
RAM	Random-Access Memory
RAT	Ram Air Turbine
RCR	Reverse Current Relay
RCVR	Receiver
RDMI	Radio Distance Magnetic Indicator
RDP	Radar Data Processing System
RDR	Radar
RDU	Remote Display Unit
REC	Receive
REF	Reference
REIL	Runway End Identifier Lights
REL	Relative
RelR	Reliability Report
RF	Radio Frequency
RFCU	Radio Frequency Control Unit
RFI	Radio Frequency Interference
RHSM	Reduced Horizontal Separation Minimum
RISC	Reduced Instruction Set Computer
RJ	Regional Jet
RLG	Ring Laser Gyroscope
RLY	Relay
RMB	RNE Main Board

Kısaltma	Açıklama
RMI	Radio Magnetic Indicator
RMP	Risk Management Plan
RNAV	Area Navigation
RNG	Range
RNP	Required Navigation Performance
ROC	Rate of Climb
ROD	Rate of Descent
ROM	Read-Only Memory
RPA	Remotely Piloted Aircraft (bkz. UAV)
RPM	Revolutions per Minute
RSC	Reusable Software Component (FAA AC 20-148)
RSP	Reversion Switch Panel
RSS	Root Sum Square
RT	Remote Terminal
RTCA	RTCA, Inc. (eski adı Radio Technical Commission for Aeronautics)
RTE	Route
RTL	Run Time Library
RTOS	Real-Time Operating System
RVR	Runway Visual Range
RVSM	Reduced Vertical Separation Minimum
RX	Receiver

S

Kısaltma	Açıklama
SAAAR	Special Aircraft and Aircrew Authorization Required (güncel adı RNP AR)
SAE	SAE International (eski adı Society of Automotive Engineers)
SAR	Search and Rescue; Smart ACMS Recorder
SAS	Software Accomplishment Summary

Kısaltma	Açıklama
SAT	Static Air Temperature
SATCOM	Satellite Communication
SATNAV	Satellite Navigation
SBLI	Software Build and Load Instructions
SBLP	Software Build and Load Control Procedure
SC	Special Committee (RTCA özel komitesi; ör. SC-205)
SCA	Structural Coverage Analysis
SCCB	Software Configuration Control Board
SCI	Software Configuration Index
SCInd	Source Code Indices
SCM	Software Configuration Management
SCMP	Software Configuration Management Plan
SCR	Software Conformity Review
SCS	Software Coding Standard (bkz. SCStd)
SCStd	Software Code Standards (DO-178C veri adı; "coding standards" olarak da anılır)
SD	Secure Digital
SDD	Software Design Description; System Design Description
SDDD	Software Design Description Document (bkz. SDD)
SDF	Simplified Directional Facility
SDI	Source Destination Identifier
SDK	Software Development Kit
SDP	Software Development Plan
SDR	Software Defined Radio
SDRAM	Synchronous Dynamic Random-Access Memory
SDRL	Supplier (Subcontractor) Data Requirements List
SDS	Software Design Standard (bkz. SDStd)
SDStd	Software Design Standards
SECI	Software Life Cycle Environment Configuration Index
SELCAL	Selective Calling
SEMP	System Engineering Management Plan

Kısaltma	Açıklama
SES	Supplier Equipment Specification
SFHA	System Functional Hazard Assessment (sistem seviyesi FHA)
SHGM	Sivil Havacılık Genel Müdürlüğü (Türkiye'nin sivil havacılık otoritesi)
SID	Standard Instrument Departure
SIRD	Software Interface Requirement Document
SIU	Satellite Interface Unit
SLA	Software Load Application
SLECI	Software Life Cycle Environment Configuration Index (yaygın kısaltması SECI)
SMA	Sub-Miniature Version A
SMP	Sub-Miniature Push-on (RF konnektör); Supplier Management Plan
SMS	Short Messaging Service
SNR	Signal-To-Noise Ratio
SoC	System on a Chip
SOI	Stage of Involvement
SOP	Standard Operating Procedure
SOUP	Software of Unknown Pedigree (kökeni belgelenmemiş yazılım)
SOW	Statement of Work
SP	Safety Plan
SPI	Serial Peripheral Interface
SPL	Spare Parts List
SPP	Safety Program Plan
SPR	Software Planning Review
SQA	Software Quality Assurance
SQAP	Software Quality Assurance Plan
SQAR	Software Quality Assurance Representative
SRATS	System Requirements Allocated to Software
SRD	Software Requirements Data (DO-178C); Source Requirement Document
SRMT	Safety Reliability Maintainability Testability
SRMTP	Safety Reliability Maintainability Testability Plan
SRR	System Requirement Review

Kısaltma	Açıklama
SRS	Software Requirements Specification; Speed Reference System
SRStd	Software Requirements Standards
SRU	Shop Replaceable Unit
SSA	System Safety Assessment
SSCV/DR	Solid-State Cockpit Voice/Data Recorder
SSCVR	Solid-State Cockpit Voice Recorder
SSFDR	Solid-State Flight Data Recorder
SSM	Sign/Status Matrix
SSPP	System Safety Program Plan
SSR	Secondary Surveillance Radar; System Specification Review; Software Specification Review
SSS	System Segment Specification
STAR	Standard Terminal Arrival Route
STARS	Standard Terminal Automation Replacement System
STC	Supplemental Type Certificate
STCA	Short-Term Conflict Alert
STP	Software Test Procedure; Software Test Plan; Standard Temperature and Pressure
STR	Software Test Results
STS	Software Test Specification
SUA	Special Use Airspace
SVA	Software Validation Attestation
SVCP	Software Verification Cases and Procedures
SVP	Software Verification Plan
SVR	Software Verification Results
SW (S/W)	Software
SW-IDD	Software Interface Design Description
SWCEH	Software and Complex Electronic Hardware
SYRD	System Requirements Document

T

Kısaltma	Açıklama
T/R (TR)	Transmit/Receive; Transmitter-Receiver (Transceiver); Thrust Reverser; Tail Rotor
TA	Traffic Advisory
TACAN	Tactical Air Navigation System
Tach	Tachometer
TAD	Terrain Awareness Display
TAF	Terminal Aerodrome Forecast
TAS	Tool Accomplishment Summary (DO-330); True Airspeed
TAT	Total Air Temperature
TAWS	Terrain Awareness and Warning System
TBD	To Be Defined; To Be Determined (eş anlamlı kullanılır)
TBO	Time Between Overhauls; Time Before Overhaul
TC	Type Certificate; Test Case
TCA	Throttle Control Assembly; Terminal Control Area
TCAD	Traffic Collision Alert Device
TCAS	Traffic Alert and Collision Avoidance System
TCF	Terrain Clearance Floor
TCI	Tool Configuration Index
TCN	TACAN
TCR	Test Completeness Review
TCU	TACAN Control Unit
TDOP	Time Dilution of Precision
TDR	Transponder (bkz. XPDR)
TDWR	Terminal Doppler Weather Radar
TEMP	Test and Evaluation Master Plan
TEP	Test and Evaluation Plan
TERPS	Terminal Instrument Procedures (United States Standard for Terminal Instrument Procedures)
TFR	Temporary Flight Restrictions
TFT	Thin-Film Transistor

Kısaltma	Açıklama
TGT	Turbine Gas Temperature; Target
THD	Total Harmonic Distortion
THD+N	Total Harmonic Distortion Plus Noise
THDG	True Heading
TIA	Type Inspection Authorization (FAA); Telecommunications Industry Association
TIS	Traffic Information Service
TK	Track Angle
TKE	Track-Angle Error
TLA	Thrust Lever Angle
TNC	Threaded Neill-Concelman (RF konnektör)
TOD	Top of Descent Point
TOR	Tool Operational Requirements
TQ	Tool Qualification
TQL	Tool Qualification Level
TQP	Tool Qualification Plan
TRACON	Terminal Radar Approach Control
TRANS	Transmit, Transmission, or Transition
TRK	Track
TrnP	Training Plan
TRP	Mode S Transponder (bkz. XPDR)
TRR	Test Readiness Review
TRX	Transceiver
TSO	Technical Standard Order
TstbR	Testability Report
TTL	Transistor-Transistor Logic; Tuned to Localizer
TTR	TCAS II Transmitter/Receiver
TTS	Time to Station
TVE	Total Vertical Error
TWDL	Two-Way Data Link; Terminal Weather Data Link
TWIP	Terminal Weather Information for Pilots

Kısaltma	Açıklama
TWR	Terminal Weather Radar
TX	Transmit

U

Kısaltma	Açıklama
UAM	Urban Air Mobility
UART	Universal Asynchronous Receiver-Transmitter
UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle
UHF	Ultra High Frequency
ULB	Underwater Locator Beacon
UML	Unified Modeling Language
UMS	User-Modifiable Software
USAF	United States Air Force
USB	Universal Serial Bus
USGS	United States Geological Survey
UTC	Coordinated Universal Time
UTRX	UHF Transceiver Module
UUT	Unit Under Test

V

Kısaltma	Açıklama
V	Volts or Voltage
V&V	Verification and Validation
V/L	VOR/Localizer
V/NAV	Vertical Navigation
V/R	Voltage Regulator

Kısaltma	Açıklama
V/REF	Reference Velocity
V/S	Vertical Speed
V/TRK	Vertical Track
VAR	Variable
VASI	Visual Approach Slope Indicator
VASIS	Visual Approach Slope Indicator (System)
VDB	VHF Data Broadcast
VDF	VHF Direction Finding
VDL	VHF Data Link
VDR	VHF Data Radio (VHF Digital Radio olarak da anılır)
VFO	Variable Frequency Oscillator
VFR	Visual Flight Rules
VG/DG	Vertical Gyroscope/Directional Gyroscope
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language (VHSIC = Very High Speed Integrated Circuit)
VHF	Very High Frequency
VMC	Visual Meteorological Conditions
VNE	Never Exceed Speed
VNO	Maximum Structural Cruising Speed
VNR	VHF Navigation Receiver
VOR	VHF Omnidirectional Range
VOR/DME	VOR with Distance Measuring Equipment
VOR/MB	VOR Marker Beacon
VORILS	VHF Omnidirectional Range and Instrument Landing System
VORTAC	VOR and TACAN Combination
VOX	Voice-Operated Exchange (sesle tetiklenen gönderme)
VPA	Vertical Path Angle
VPATH	Vertical Path
VRP	Visual Reference Point; Visual Reporting Point
VRX	VHF Receiver Module

Kısaltma	Açıklama
VSG	Vector Signal Generator
VSI	Vertical Speed Indicator
VSM	Vertical Separation Minimum
VSO	Stall Speed in Landing Configuration
VSWR	Voltage Standing Wave Ratio
VX	Speed for Best Angle of Climb
VY	Speed for Best Rate of Climb

W

Kısaltma	Açıklama
WAAS	Wide Area Augmentation System
WBS	Work Breakdown Structure
WCET	Worst-Case Execution Time
WD	Wind Direction
WG	Working Group (EUROCAE çalışma grubu; ör. WG-71)
WINDR	Wind Direction (bkz. WD)
WMA	WXR Waveguide Adapter
WMI	WXR Indicator Mount
WMS	Wide-Area Master Station
WMSC	Weather Message Switching Center
WMSCR	Weather Message Switching Center Replacement
WPT	Waypoint
WRT	WXR Receiver Transmitter
WS	Wind Shear
WX	Weather
WXR	Weather Radar System
WYPT	Waypoint (bkz. WPT)

X

Kısaltma	Açıklama
XADC	Xilinx Analog to Digital Converter
XCVR	Transceiver
XFR	Transfer
XMIT	Transmit
XMSN	Transmission
XMTR	Transmitter
XPDR	Transponder (bkz. TDR, TRP)
XTAL	Crystal
XTK	Crosstrack

Z

Kısaltma	Açıklama
ZSA	Zonal Safety Analysis

SW SOI-1

SOI-1, sertifikasyon otoritesinin projeye ilk resmî katılımıdır: yazılım planları ve standartları, yazılım seviyesinin gerektirdiği hedefleri karşılayacak biçimde yazılmış mı, birbiriyle tutarlı mı ve ekip gerçekten bu planlara göre çalışmaya başlamış mı? Bu sayfa, denetime girmeden önce ekibin kendi kendine sorabileceği soruları bir kontrol listesi olarak toplar.

Katılım aşaması (Stage of Involvement, SOI) denetimlerinin genel mantığı, dört aşamanın birbirine bağlantısı ve denetim akışı [12. Sertifikasyon İrtibatı](#) bölümünde anlatılır. Buradaki maddeler bir otorite listesinin çevirisi değil, sahada işe yarayan hazırlık sorularıdır; denetimin kapsamı ve biçimi otoriteye, yazılım seviyesine ve projenin özelliklerine göre değişir.



BU LİSTEYİ NASIL KULLANMALI?

Listeyi denetimden birkaç hafta önce, kalite güvencesinin yürüttüğü bir iç ön denetimde (mock audit) kullanın. "Hayır" ya da "kısmen" yanıtı alan her madde için ya düzeltici faaliyet açın ya da gerekçeyi yazın; otoritenin aynı soruyu sorduğunda alacağı yanıt o gerekçedir.

Ne zaman yapılır?

SOI-1, planlar yayımlandıktan sonra, ama geliştirme veri üretimi hızlanmadan önce yapılmalıdır. Çok erken çağrılırsa planlar henüz olgunlaşmamıştır; çok geç çağrılırsa otoritenin isteyeceği değişiklikler, planlara göre zaten üretilmiş gereksinim, tasarım ve kodu da etkiler. Tipik giriş kriterleri şunlardır:

- ☐ Yazılım seviyesinin gerektirdiği planlar ve geliştirme standartları (Seviye A, B ve C'de beş plan ile gereksinim, tasarım ve kodlama standartları; Seviye D'de beş plan) yazılmış, gözden geçirilmiş ve konfigürasyon yönetimi altına alınmıştır. Planların tek belgede birleştirilmesi ya da standartların yazılım geliştirme planıyla (Software Development Plan, SDP) aynı belgede paketlenmesi engel değildir; önemli olan içeriğin eksiksiz olmasıdır. Yalnız, bazı otoriteler yazılım sertifikasyon planını (Plan for Software Aspects of Certification, PSAC) ayrı belge olarak ister.
- ☐ Planların gözden geçirme kayıtları ve kalite güvencesinin planlara ilişkin gözden geçirme kayıtları hazırdır.
- ☐ Sistem tarafından atanan yazılım seviyesi ve dayandığı emniyet değerlendirmesi çıktıları (fonksiyonel tehlike değerlendirmesi (functional hazard assessment, FHA) ve ön sistem emniyet değerlendirmesi (preliminary system safety assessment, PSSA)) gösterilebilir durumdadır.
- ☐ Araç kalifikasyonu (tool qualification) gerektiren araçlar belirlenmiş, varsa araç kalifikasyon planları yazılmıştır.
- ☐ PSAC otoriteye denetimden önce, okunmaya yetecek süre bırakılarak gönderilmiştir.

Seviye D'de kapsam daralır: planlama hedeflerinden yalnızca yaşam döngüsü faaliyetlerinin tanımlanması ile ek hususların ele alınması aranır. Beş plan yine beklenir; geliştirme standartlarının tanımlanması, planların DO-178C'ye uyumunun gözden geçirilmesi ve kalite güvencesinin plan ve standartlara ilişkin güvencesi bu seviyede hedef değildir.

Planlar: otoritenin baktığı noktalar

Sertifikasyon yaklaşımı (PSAC)

- ☐ Sistem ve yazılım kısa ama anlaşılır biçimde tanıtılmış mı; yazılımın hangi işlevi yerine getirdiği ve hangi arıza durumlarına katkı verebileceği açık mı?
- ☐ Yazılım seviyesi ve gerekçesi, sistem emniyet değerlendirme sürecinin çıktılarıyla tutarlı mı?
- ☐ Uyumun neye göre gösterileceği açık mı: DO-178C'yi kabul edilebilir uyum yöntemi olarak tanıyan otorite dokümanı (FAA için AC 20-115D, EASA için AMC 20-115D) ve varsa projeye özgü ek otorite beklentileri yazılmış mı?
- ☐ Ek değerlendirme gerektiren durumlar listelenmiş mi: önceden geliştirilmiş yazılım (previously developed software, PDS), ticari hazır yazılım (commercial off-the-shelf, COTS), kalifiye edilecek araçlar, sahada yüklenebilir yazılım, kullanıcı tarafından değiştirilebilir yazılım, parametre verisi öğeleri (parameter data item, PDI), devre dışı bırakılmış kod (deactivated code), bölümlendirme iddiaları, çok çekirdekli işlemci (multi-core processor) kullanımı (AC 20-193 / AMC 20-193), alternatif yöntemler?
- ☐ DO-331, DO-332 ya da DO-333 eklerinden (supplement) hangilerinin neden uygulanacağı ve hangi araçların DO-330'a göre kalifiye edileceği yazılmış mı?
- ☐ Sertifika anında açık kalabilecek problem raporlarının nasıl sınıflandırılıp yönetileceği (AC 20-189 / AMC 20-189 doğrultusunda) planlarda tanımlı mı? Sınıflar 9. bölümün problem raporlama kısmında anlatılır.
- ☐ Hangi yaşam döngüsü verisinin otoriteye sunulacağı, hangisinin yalnızca talep üzerine gösterileceği belli mi?
- ☐ Takvim, SOI denetimlerinin yaklaşık zamanlarını içeriyor mu?
- ☐ Tedarikçiler veya dış kaynak kullanılıyorsa gözetim yaklaşımı tanımlı mı?

Geliştirme, doğrulama ve destek planları

- ☐ Yaşam döngüsü modeli ve her sürecin geçiş kriterleri (transition criteria) ölçülebilir biçimde tanımlı mı? ([Ek A](#))
- ☐ SDP, kullanılan dili, derleyiciyi, bağlayıcıyı ve geliştirme ortamını sürümleriyle belirtiyor; gereksinim, tasarım ve kodlama kuralları için ilgili standartlara atıf yapıyor mu?
- ☐ Yazılım doğrulama planı (Software Verification Plan, SVP), yazılım seviyesinin gerektirdiği bağımsızlığı (independence) kimin, hangi faaliyette sağlayacağını söylüyor mu? Bağımsızlık her hedefte değil, seviyeye göre belirli hedeflerde aranır ([9. Yazılım Doğrulama](#)).

- ☐ Gereksinim tabanlı test (requirements-based testing), gürbüzlük (robustness) testi, hedef donanımda test ve yapısal kapsam analizi (structural coverage analysis) yaklaşımı tanımlı mı?
- ☐ Kapsam boşluklarının, ölü kodun (dead code) ve devre dışı bırakılmış kodun nasıl ele alınacağı planlanmış mı?
- ☐ Yazılım konfigürasyon yönetimi planı (Software Configuration Management Plan, SCMP), konfigürasyon öğelerini, temel çizgi (baseline) kurallarını, değişiklik kontrolünü ve problem raporlama (problem reporting) akışını tanımlıyor mu?
- ☐ SCMP, hangi yaşam döngüsü verisinin hangi kontrol kategorisinde (control category; CC1 ya da CC2) yönetileceğini, arşivleme ve geri getirme yöntemini ve yaşam döngüsü ortamının (derleyici, araçlar, test düzeneği) nasıl kontrol altında tutulacağını söylüyor mu? ([10. Yazılım Konfigürasyon Yönetimi](#))
- ☐ Yazılım kalite güvencesi planı (Software Quality Assurance Plan, SQAP), kalite güvencesinin bağımsızlığını, denetim sıklığını ve uygunsuzluk kayıtlarının nasıl kapatılacağını açıklıyor mu?
- ☐ Planlar arasındaki çapraz başvurular tutarlı mı? (Bir planın "SVP'de tanımlanmıştır" dediği konu gerçekten SVP'de var mı?)

Standartlar

- ☐ Gereksinim standardı, iyi bir gereksinimin ölçütlerini ve türetilmiş gereksinimin (derived requirement) nasıl işaretlenip sistem süreçlerine iletileceğini tanımlıyor mu?
- ☐ Tasarım standardı karmaşıklık, kesme kullanımı, dinamik bellek ve özyineleme gibi konularda kısıt koyuyor mu?
- ☐ Kodlama standardı dil alt kümesini (örneğin MISRA C tabanlı bir kural seti), sapma (deviation) sürecini ve otomatik denetim yöntemini tanımlıyor mu?
- ☐ Standartlardaki kurallar gözden geçirmede "uyuyor / uymuyor" diye yanıtlanabilecek kadar somut mu?

Araçlar

- ☐ Kullanılan her aracın rolü listelenmiş ve hangilerinin kalifikasyon gerektirdiği gerekçelendirilmiş mi?
- ☐ Kalifikasyon gerektiren araçlar için araç kalifikasyon seviyesi (tool qualification level, TQL) belirlenmiş mi?
- ☐ Kalifikasyon gerektirmediği söylenen araçlar için "çıktısı başka bir faaliyetle doğrulanıyor" gerekçesi gerçekten doğru mu?

Hazırlanacak veri paketi

DO-178C'nin otoriteye sunulmasını asgari olarak beklediği veri PSAC, yazılım konfigürasyon indeksi (Software Configuration Index, SCI) ve yazılım başarı özetidir (Software Accomplishment Summary, SAS); bu aşamada bunlardan yalnızca PSAC vardır. Kalifikasyon gerektiren araç varsa DO-330, TQL-1 ile TQL-4 arasındaki araçlar için araç kalifikasyon planının da PSAC ile birlikte sunulmasını bekler. Geri kalan veri denetimde gösterilir ve talep edildiğinde erişime açılır. Otorite başka planların da önceden gönderilmesini isteyebilir; bu yüzden aşağıdaki ayırım bir alt sınırdır, kesin liste PSAC'ta otoriteyle kararlaştırılır.

Veri	Otoriteye	Neden istenir
PSAC	Sunulur	Sertifikasyon yaklaşımının otoriteyle üzerinde anlaşılan özeti; denetimin çıkış noktası
SDP, SVP, SCMP, SQAP	Denetimde gösterilir	Hedeflerin hangi süreç, yöntem ve ortamla karşılanacağını gösterir
Gereksinim, tasarım ve kodlama standartları	Denetimde gösterilir	Geliştirme verisinin hangi kurallarla üretileceğini gösterir
Planların gözden geçirme kayıtları	Denetimde gösterilir	Planların kendisinin de doğrulandığının kanıtı
Kalite güvencesi kayıtları	Denetimde gösterilir	Kalite güvencesinin plan ve standartları gözden geçirdiğinin, bulduğu uygunsuzlukları izleyip kapattığının kanıtı
Araç kalifikasyon planları (varsa)	Sunulur (TQL-1 – TQL-4); TQL-5'te ayrı plan beklenmez, yaklaşım PSAC'ta anlatılır	Araç kredisinin baştan doğru kurgulandığını gösterir; kalifikasyon yaklaşımı üzerindeki mutabakatın dayanağıdır
Emniyet değerlendirmesinden ilgili çıktılar	Denetimde gösterilir	Yazılım seviyesinin gerekçesi
Konfigürasyon kayıtları	Denetimde gösterilir	Denetlenen plan ve standartların hangi sürüm olduğunu, temel çizgiye alındığını ve sonraki değişikliklerin kontrol altında olduğunu gösterir

Sık görülen bulgular

Bulgu	Tipik nedeni	Önlem
Planlar başka bir projeden kopyalanmış, proje adı dışında farkı yok	Takvim baskısı, "şablon yeter" varsayımı	Planı projeye özgü kararlarla (araçlar, ortam, ekip yapısı) doldurmak

Bulgu	Tipik nedeni	Önlem
Geçiş kriterleri "tamamlandığında" gibi ölçülemez ifadeler	Kriterin kim tarafından nasıl kontrol edileceği düşünülmemiş	Her kriteri bir kayıt ya da ölçümle ilişkilendirmek
Planlar arasında çelişki (araç, sürüm, sorumluluk)	Planların farklı kişilerce, birbirinden habersiz yazılması	Planları birlikte gözden geçirmek; tek bir terim ve araç listesi tutmak
Bağımsızlık gerekliliği planda yok ya da belirsiz	Ekip yapısının sonradan belirlenmesi	Bağımsızlığın arandığı hedefleri seviyeye göre çıkarıp her biri için kimin sağlayacağını SVP'ye yazmak
Kalifikasyon gerektiren bir araç hiç anılmamış	Araç envanterinin çıkarılmaması	Geliştirme ve doğrulama zincirindeki her aracı listeleyip tek tek değerlendirmek
Planlar konfigürasyon yönetimi altında değil	Planların "belge" değil "taslak" gibi görülmesi	Planları ilk temel çizginin parçası yapmak

Denetimden sonra

Otorite bulgularını ve gözlemlerini yazılı olarak iletir. Her bulgu için bir düzeltici faaliyet planı, sorumlu ve tarih belirlenir; planlarda yapılan değişiklikler yeni sürümle yeniden yayımlanır ve kayda bağlanır. SOI-1 bulguları açıkken geliştirmeye hız vermek, düzeltilmiş planlara göre yeniden iş yapma riskini büyütür. Bulgular kapatılmadan ya da en azından kapanış planında otoriteyle mutabık kalınmadan SOI-2'ye girilmez.

İlgili bölümler

- [5. Yazılım Planlama](#)
- [9. Yazılım Doğrulama](#)
- [10. Yazılım Konfigürasyon Yönetimi](#)
- [11. Yazılım Kalite Güvencesi](#)
- [12. Sertifikasyon İrtibatı](#)
- [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)
- Sonraki aşama: [SW SOI-2](#)

SW SOI-2

SOI-2, otoritenin geliştirme verisine ilk kez yakından baktığı denetimdir: gereksinim, tasarım ve kod planlarda söylendiği gibi mi üretiliyor, standartlara uyuyor mu, birbirine izlenebilir mi ve gözden geçirmeler gerçekten iş görüyor mu? Bu sayfa, SOI-2 öncesinde ekibin kendi verisini sınamak için kullanabileceği bir kontrol listesidir.

Katılım aşaması (Stage of Involvement, SOI) denetimlerinin genel mantığı [12. Sertifikasyon İrtibatı](#) bölümünde anlatılır. Aşağıdaki maddeler bir otorite listesinin çevirisi değil, sahada işe yarayan hazırlık sorularıdır; kapsam otoriteye ve yazılım seviyesine göre değişir.



ÖRNEKLEMeye HAZIRLANIN

SOI-2'de otorite veriyi baştan sona okumaz, örnekler. Tipik yöntem, rastgele seçilen bir gereksinimin izini sistem gereksiniminden koda kadar aşağı, bir kod parçasının izini de gereksinime kadar yukarı sürmektir. Bu iki yönlü izi araç üzerinde dakikalar içinde gösteremiyorsanız, denetim o noktada uzar.

Ne zaman yapılır?

SOI-2'nin amacı süreci erken sınamaktır; bu yüzden geliştirmenin bitmesi beklenmez. Beklenen, gereksinim, tasarım ve kodun temsil edici bir bölümünün üretilip gözden geçirilmiş olmasıdır. Eşik otoriteye göre değişir: FAA'nın Order 8110.49'u tipik olarak en az %50'yi anar, EASA'nın artık yürürlükte olmayan CM-SWCEH-002 notu en az %75'i arıyordu. Projeniz için geçerli eşik, yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) otoriteyle kararlaştırılır. Tipik giriş kriterleri:

- ☐ SOI-1 bulguları kapatılmış ya da kapanış planında otoriteyle mutabık kalınmıştır.
- ☐ Yüksek seviyeli gereksinimlerin (high-level requirements) önemli bir bölümü, bunlara karşılık gelen tasarım ve kod üretilmiş ve gözden geçirilmiştir.
- ☐ Örnek alınabilecek uçtan uca en az birkaç işlev vardır (gereksinimden koda).
- ☐ İzlenebilirlik (traceability) verisi araç üzerinde güncel ve sorgulanabilir durumdadır.
- ☐ Kalite güvencesi, geliştirme sürecinde en az bir denetim yapmış ve kaydını tutmuştur.

Geliştirme verisi: otoritenin baktığı noktalar

Aşağıdaki soruların hepsi her projede sorulmaz; yazılım seviyesi düştükçe aranan hedef azalır. Özellikle Seviye D'de ağırlık yüksek seviyeli gereksinimlere ve çalıştırılabilir nesne koduna (executable object code) kayar: düşük seviyeli gereksinimlerin ve kaynak kodun ne geliştirilmesi

ne de doğrulanması hedef olarak aranır; mimarinin geliştirilmesi beklenir, ama üzerindeki doğrulama hedeflerinden yalnızca bölümlene bütünlüğünün teyidi kalır. Projeniz için geçerli kapsamı [4. DO-178C ve Destekleyici Dokümanlara Genel Bakış](#) bölümündeki seviye–hedef özetine ve PSAC'a göre belirleyin.

Gereksinimler

- ☐ Yüksek seviyeli gereksinimler sistem gereksinimlerine, düşük seviyeli gereksinimler (low-level requirements) yüksek seviyeli gereksinimlere izlenebilir mi; izlenemeyenler türetilmiş olarak işaretli mi?
- ☐ Gereksinimler doğrulanabilir mi; her biri test ya da analizle "karşılandı / karşılanmadı" diye yanıtlanabilir mi?
- ☐ Türetilmiş gereksinimler (derived requirements), yüksek ya da düşük seviyeli olsun, gerekçelendirilmiş ve sistem süreçlerine (sistem emniyet değerlendirme süreci dahil) geri bildirilmiş mi?
- ☐ Yüksek seviyeli gereksinimler tasarım ayrıntısı (fonksiyon adı, değişken) içermiyor mu?
- ☐ Gereksinim standardına uyum gözden geçirmede kontrol edilmiş mi?

Tasarım ve mimari

- ☐ Yazılım mimarisi (software architecture) bileşenleri, arayüzleri, veri ve kontrol akışını gösteriyor mu?
- ☐ Bölümlene ya da koruma mekanizması iddia ediliyorsa tasarımda nerede sağlandığı görünüyor mu?
- ☐ Düşük seviyeli gereksinimler, kodun gereksinim dışında karar vermesine gerek bırakmayacak kadar ayrıntılı mı?
- ☐ Tasarım standardındaki kısıtlara (karmaşıklık, kesme, dinamik bellek) uyulmuş mu?

Kaynak kod ve entegrasyon

- ☐ Düşük seviyeli gereksinimlerin hepsi kaynak koda (source code) dönüştürülmüş ve iz verisi iki yönde kurulmuş mu; hiçbir gereksinime bağlanmayan, belgelenmemiş bir işlevi gerçekleştiren kod var mı?
- ☐ Kodlama standardına uyum araçla denetlenmiş, sapmalar gerekçelendirilmiş mi?
- ☐ Kod ile tasarım arasında fark var mı (tasarımda olmayan bir işlev, farklı bir arayüz)?
- ☐ Derleme seçenekleri ve araç sürümleri planlarda yazılanlarla aynı mı?
- ☐ Derleme ve bağlama betikleri konfigürasyon yönetimi altında mı; çalıştırılabilir nesne kodu bir geliştiricinin masasında değil, kayıtlı ortamda bu betiklerle mi üretiliyor?
- ☐ Derleyici ve bağlayıcı (linker) uyarıları ile bağlama haritası (link map) gözden geçirilmiş mi: açıklanmamış uyarı, yanlış adres, çakışan bellek bölgesi, eksik ya da beklenmeyen bileşen var mı?

☐ Parametre verisi ögesi (parameter data item, PDI) dosyaları, model ya da otomatik üretilen kod varsa izlenebilirlik ve doğrulama yaklaşımı planlarda anlatıldığı gibi mi uygulanıyor?

Gözden geçirmeler

☐ Her gözden geçirme kaydında incelenen ürünün sürümü, katılımcılar, kullanılan kontrol listesi, bulgular ve kapanış var mı?

☐ Bağımsızlığın arandığı hedeflerde (yazılım seviyesine göre değişir) gözden geçiren yazardan bağımsız mı ve bu, kayıttan okunabiliyor mu?

☐ Bulgular gerçekten kapatılmış ve kapanış doğrulanmış mı?

☐ Kontrol listeleri madde madde doldurulmuş mu, yoksa tek bir "uygundur" ile mi geçilmiş?

Süreçler

☐ Geliştirme verisi temel çizgiye (baseline) alınıyor ve değişiklikler kontrol altında yapılıyor mu?

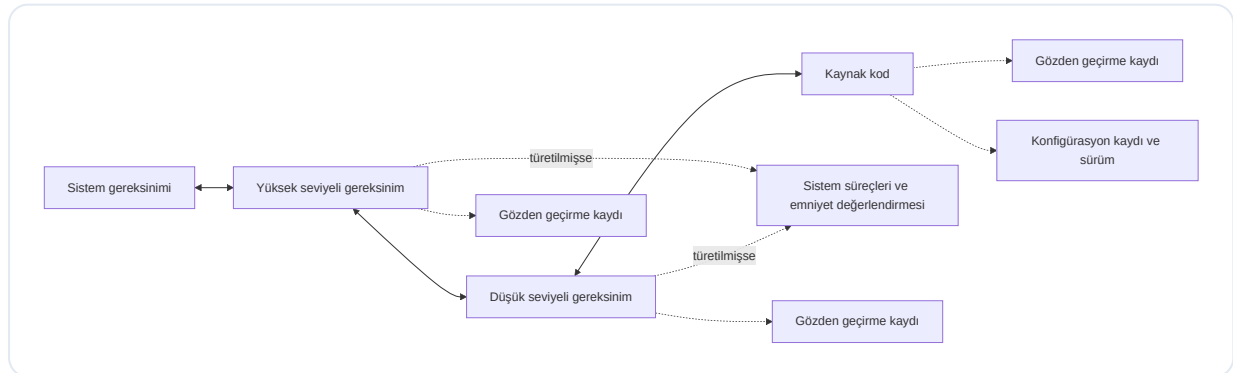
☐ Problem raporları (problem reports) planda tanımlandığı gibi açılıyor, sınıflandırılıyor ve izleniyor mu?

☐ Plandan yapılan sapmalar kayıt altında mı; gerekiyorsa planlar güncellenmiş mi?

☐ Kalite güvencesi, planlara uyumu denetlemiş ve uygunsuzlukları kayda geçirmiş mi?

İz sürme alıştırması

Denetimden önce aşağıdaki zinciri birkaç rastgele gereksinim için aşağı doğru, birkaç rastgele kod parçası için de yukarı doğru kendiniz yürütün. Düz oklar izin iki yönde de sürülebilmesi gerektiğini, kesikli oklar her halkada açıp gösterebilmeniz gereken kaydı anlatır; yolda türetilmiş bir gereksinime rastlarsanız gerekçesini ve sistem süreçlerine iletilişinin kaydını da açın. Zincirin herhangi bir halkasında takılıyorsanız, otorite de aynı yerde takılacaktır.



Hazırlanacak veri paketi

Veri	Neden istenir
Yüksek ve düşük seviyeli gereksinimler	Gereksinim kalitesi ve standart uyumu
Mimari ve tasarım verisi	Gereksinimlerin nasıl gerçekleştirildiği
Kaynak kod ve kodlama standardı denetim çıktıları	Kodun standarda ve tasarıma uyumu
Derleme ve bağlama verisi (betikler, seçenekler, bağlama haritası, uyarı kayıtları)	Çalıştırılabilir nesne kodunun kayıtlı ortamda, yinelenabilir biçimde üretildiği
İzlenebilirlik verisi	İki yönlü iz sürme
Gözden geçirme kayıtları ve kontrol listeleri	Doğrulamanın geliştirmeye birlikte yürüdüğü kanıtı
Problem raporları ve değişiklik kayıtları	Sorunların sistematik yönetildiği
Temel çizgi ve konfigürasyon kayıtları	İncelenen verinin hangi sürüme ait olduğu
Kalite güvencesi denetim kayıtları	Süreç uyumunun bağımsız olarak izlendiği

Sık görülen bulgular

Bulgu	Tipik nedeni	Önlem
Gözden geçirme kayıtlarında hiç bulgu yok	Gözden geçirmenin onay formalitesine dönüşmesi	Bulgusuz kayıtları kalite güvencesinin örnekleme; kontrol listesinin madde madde doldurulması
İzlenebilirlik toplu bağlantılarla kurulmuş ("bu modül şu 40 gereksinimi karşılar")	İzin sonradan, toplu olarak eklenmesi	İzi gereksinim yazılırken ve kod değişirken güncellemek
Türetilmiş gereksinimler sistem süreçlerine ve emniyet değerlendirmesine iletilmemiş	Yazılım ve sistem ekipleri arasında geri besleme kanalı olmaması	Türetilmiş gereksinimler için ayrı bir onay adımı tanımlamak
Kod tasarımdan farklı	Kodun değiştirilip tasarımın güncellenmemesi	Değişiklik etki analizinde (change impact analysis) tasarımı zorunlu kalem yapmak
Problem raporu yerine e-posta ya da sözlü düzeltme	Raporlama sürecinin ağır görülmesi	Hafif ama tek bir problem raporu akışı; kalite güvencesinin bunu denetlemesi
Plan dışı araç ya da derleme seçeneği kullanılmış	Planın güncellenmeden pratiğin değişmesi	Sapmayı kayda almak, planı yeni sürümle yayımlamak

Denetimden sonra

SOI-2 bulguları çoğu zaman tekil hatalara değil, süreç boşluklarına işaret eder. Düzeltici faaliyet yapılırken "bu kayıt düzeltildi" demek yetmez; aynı hatanın başka verilerde de olup olmadığı taranmalı ve süreç (kontrol listesi, araç ayarı, eğitim) buna göre değiştirilmelidir. Bulgular kapatılmadan, ya da en azından kapanış planında otoriteyle mutabık kalınmadan SOI-3'e girmek, doğrulama kanıtının dayandığı verinin sorgulanması demektir.

İlgili bölümler

- [6. Yazılım Gereksinimleri](#)
- [7. Yazılım Tasarımı](#)
- [8. Yazılım Gerçekleştirme: Kodlama ve Entegrasyon](#)
- [9. Yazılım Doğrulama](#)
- [10. Yazılım Konfigürasyon Yönetimi](#)
- [11. Yazılım Kalite Güvencesi](#)
- [12. Sertifikasyon İrtibatı](#)
- Önceki aşama: [SW SOI-1](#) · Sonraki aşama: [SW SOI-3](#)

SW SOI-3

SOI-3, otoritenin doğrulama kanıtını incelediği denetimdir: testler gereksinimlerden mi türetilmiş, normal ve gürbüzlük durumlarını kapsıyor mu, sonuçlar doğru yazılım sürümüne ait mi ve kapsam analizi geride açıklanamamış bir boşluk bırakıyor mu? Bu sayfa, SOI-3 öncesinde doğrulama verisini sınamak için bir kontrol listesi sunar.

Katılım aşaması (Stage of Involvement, SOI) denetimlerinin genel mantığı [12. Sertifikasyon İrtibatı](#) bölümünde, doğrulamanın kendisi [9. Yazılım Doğrulama](#) bölümünde anlatılır. Buradaki maddeler bir otorite listesinin çevirisi değil, sahada işe yarayan hazırlık sorularıdır; hangi maddenin geçerli olduğu otoriteye ve yazılım seviyesine (software level) göre değişir, seviyeye bağlı maddelerde bu ayrıca belirtilmiştir.

ÖRNEKLEME BU KEZ TESTTEN BAŞLAR

SOI-3'te otorite çoğu zaman bir gereksinim seçer ve sorar: Bu gereksinimi hangi testler sınıyor, beklenen sonuç nereden geliyor, test hangi yazılım sürümünde, hangi ortamda koştu, sonuç nerede kayıtlı, kapsam analizinde bu kodun durumu ne? Bu zinciri her halkasıyla gösterebilmek, SOI-3 hazırlığının özüdür.

Ne zaman yapılır?

SOI-3 için doğrulamanın bitmesi beklenmez; denetim, doğrulama verisinin temsil edici bir bölümü tamamlanıp gözden geçirildiğinde ve yapısal kapsam analizi (structural coverage analysis) sonuç üretmeye başladığında yapılır. "Temsil edici bölüm" için FAA'nın yazılım onay yönergesi (Order 8110.49) tipik olarak en az yarısını, EASA'nın artık yürürlükte olmayan CM-SWCEH-002 sertifikasyon notu ise en az dörtte üçünü anıyordu; projede geçerli eşik, yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) otoriteyle kararlaştırılır. Tipik giriş kriterleri:

- ☐ SOI-2 bulguları kapatılmış ya da kapanış planında otoriteyle mutabık kalınmıştır.
- ☐ Test durumlarının (test cases) ve test prosedürlerinin (test procedures) kararlaştırılan eşiği karşılayan bölümü yazılmış, gözden geçirilmiş ve kredi amaçlı (for-credit) olarak koşulmuştur.
- ☐ Gereksinim kapsam analizi ve yapısal kapsam analizi, koşulan testler için sonuç vermiştir; bulunan boşluklar sınıflandırılmaktadır ve örnek gösterilebilecek çözülmüş boşluk kayıtları vardır.
- ☐ Test ortamı ve araçları yazılım yaşam döngüsü ortam konfigürasyon indeksinde (Software Life Cycle Environment Configuration Index, SECI) tanımlıdır.
- ☐ Zamanlama, yığın ve bellek analizleri gibi test dışı doğrulama çalışmalarının yöntemi belirlenmiş ve ilk sonuçları alınmıştır.

Doğrulama verisi: otoritenin baktığı noktalar

Testlerin türetilmesi

- ☐ Her test durumu bir ya da daha fazla gereksinime izlenebilir mi; gereksinimi olmayan test var mı?
- ☐ Beklenen sonuçlar gereksinimden mi türetilmiş, yoksa kodun gerçek çıktısından mı kopyalanmış?
- ☐ Normal aralık testlerinin yanında gürbüzlük (robustness) testleri var mı: geçersiz ve sınır dışı girdiler, bozuk veri, anormal başlatma, aşılın çerçeve süresi, izin verilmeyen durum geçişleri?
- ☐ Normal aralık testlerinde geçerli eşdeğerlik sınıfları (equivalence classes) ve sınır değerleri (boundary values) bilinçli olarak seçilmiş mi?
- ☐ Testler ve prosedürler gözden geçirilmiş; gözden geçirme kaydı var mı?

Testlerin koşulması ve sonuçlar

- ☐ Test sonuçları, hangi yazılım sürümüyle, hangi ortamda ve hangi prosedür sürümüyle elde edildiklerini açıkça gösteriyor mu?
- ☐ Beklenen sonuçla gerçekleşen sonuç arasındaki her fark açıklanmış mı; başarısız testler problem raporuyla (problem report) ilişkilendirilmiş mi?
- ☐ Kod değiştikten sonra hangi testlerin yeniden koşulduğu bir regresyon analiziyle (regression analysis) gerekçelendirilmiş mi?
- ☐ Yazılım–donanım entegrasyon testleri hedef donanımda mı yapılmış; benzetici ya da emülatörden kredi alındıysa hedeften farkların hata yakalama yeteneğine etkisi değerlendirilmiş, orada yakalanamayacak hatalar için planda belirtilen başka doğrulama faaliyeti yapılmış mı?
- ☐ Test prosedürleri, başka bir kişi tarafından aynı sonuçla tekrarlanabilecek kadar açık mı?

Kapsam analizi

- ☐ Gereksinim kapsam analizi, her gereksinimin test edildiğini (ya da neden analiz veya gözden geçirmeye karşılandığını) gösteriyor mu?
- ☐ Yapısal kapsam, yazılım seviyesinin gerektirdiği ölçütlerle ölçülmüş mü: Seviye C'de satır kapsama (statement coverage), Seviye B'de buna ek olarak karar kapsama (decision coverage), Seviye A'da ek olarak değiştirilmiş koşul/karar kapsama (modified condition/decision coverage, MC/DC)?
- ☐ Kapsam verisi gereksinim tabanlı testlerden mi toplanmış; yalnızca boşluğu kapatmak için koda bakarak yazılmış, gereksinime izlenemeyen test var mı?
- ☐ Kapsam boşlukları sınıflandırılmış mı: eksik test, eksik gereksinim, ölü kodu (dead code) da içeren gereksiz kod (extraneous code), devre dışı bırakılmış kod (deactivated code)?

- ☐ Her boşluk için çözüm (yeni test, gereksinim değişikliği, kod kaldırma ya da gerekçeli analiz) kayıt altında mı?
- ☐ Kapsam ölçümü kod enstrümantasyonu (instrumentation) ile yapıldıysa, enstrümente edilmiş ve edilmemiş kod arasındaki farkın etkisi tartışılmış mı?
- ☐ Seviye A'da, derleyicinin ya da bağlayıcının ürettiği ve kaynak koda doğrudan izlenemeyen nesne kodu belirlenmiş ve doğruluğu ek doğrulamayla gösterilmiş mi; böyle kod yoksa bunu gösteren analiz var mı?
- ☐ Seviye A, B ve C'de veri bağlaşımı ve kontrol bağlaşımı (data coupling and control coupling) analizi, mimaride tanımlı bileşen arası ilişkilerin testlerde gerçekten işletildiğini gösteriyor mu?

Seviye D'de yapısal kapsam hedefi yoktur; bu grupta yalnızca yüksek seviyeli gereksinimlerin test kapsamı sorulur (düşük seviyeli gereksinimlerin test kapsamı Seviye A, B ve C'de aranır). Ölçütlerin ne istediği 9. bölümde, boşluk sınıflarının ayrıntısı [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#) bölümünde anlatılır.

Analizler ve araçlar

- ☐ En kötü durum yürütme süresi (worst-case execution time, WCET) ve yığın kullanımı analizle belirlenip ölçümle desteklenmiş, bellek haritası analizi yapılmış ve sonuçlar gereksinimlerle karşılaştırılmış mı?
- ☐ Gözden geçirme ve analiz sonuçları da doğrulama sonuçları arasında kayıtlı mı ve hangi veri sürümüne ait oldukları belli mi?
- ☐ Parametre verisi ögesi (parameter data item, PDI) dosyaları varsa, yapıları, öznitelikleri ve içerdikleri değerler doğrulanmış, dosyanın tüm ögeleri bu doğrulamada kapsanmış mı (bkz. [22. Konfigürasyon Verisi](#))?
- ☐ Test ve kapsam araçlarının kalifikasyon gerekip gerekmediği değerlendirilmiş; gerekiyorsa kalifikasyon verisi hazır mı?
- ☐ Bağımsızlığın arandığı doğrulama hedeflerinde (Seviye A ve B) bağımsızlık sağlanmış ve kayıtlardan izlenebilir mi?

Problem raporları ve süreçler

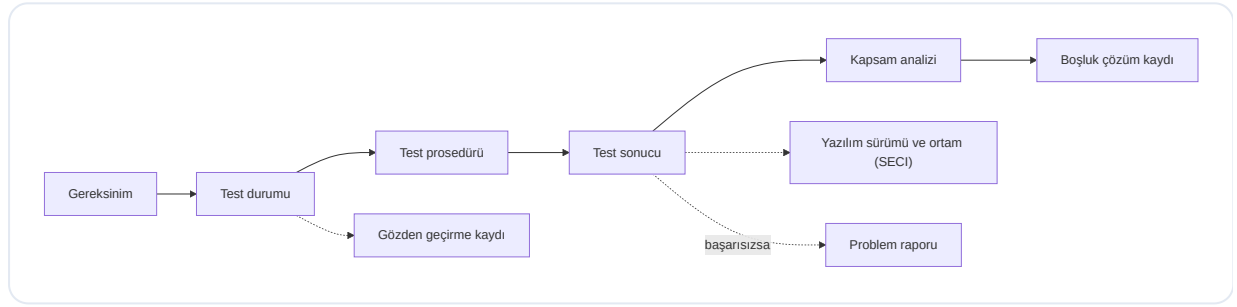
- ☐ Doğrulamada bulunan her kusur (başarısız test, kontrol altındaki veride gözden geçirme bulgusu, analizde çıkan sapma) problem raporuyla kayda alınmış mı?
- ☐ Raporlar planda tanımlandığı gibi sınıflandırılmış; emniyet etkisi olabilecekler sistem emniyet değerlendirme sürecine geri bildirilmiş mi?
- ☐ Kapatılan raporlarda düzeltmenin nasıl doğrulandığı (yeniden koşulan testler, değişiklik etki analizi (change impact analysis)) kayıttan izlenebiliyor mu?
- ☐ Test durumları, prosedürleri, sonuçları ve test ortamı konfigürasyon yönetimi altında mı; kredi amaçlı koşular temel çizgiye (baseline) alınmış veriyle mi yapılmış?

☐ Kalite güvencesi, kredi amaçlı koşullara tanıklık etmiş ya da kayıtlarını denetlemiş; bulduğu uygunsuzlukları kayda geçirmiş mi?

Açık kalacak raporların sınıflandırılması ve gerekçelendirilmesi asıl olarak SOI-4'ün konusudur; sınıflar 9. bölümün problem raporlama kısmında tanımlanır.

İz sürme alıştırması

Denetimden önce aşağıdaki akışı birkaç rastgele gereksinim için kendiniz yürütün; sonra ters yönden, rastgele seçtiğiniz bir test sonucundan gereksinime doğru deneyin. Herhangi bir okta takılıyorsanız, otorite de aynı yerde takılacaktır.



Hazırlanacak veri paketi

Veri	Neden istenir
Yazılım doğrulama durumları ve prosedürleri (Software Verification Cases and Procedures, SVCP)	Testlerin gereksinimden türetildiğini ve tekrarlanabilir olduğunu gösterir
Yazılım doğrulama sonuçları (Software Verification Results, SVR)	Hangi sürüm ve ortamda neyin geçtiği
Gereksinim–test izlenebilirlik verisi	Her gereksinimin hangi testle sınıandığı
Gereksinim ve yapısal kapsam analizi sonuçları	Testlerin yeterliliği ve boşlukların çözümü
Zamanlama, yığın ve bellek analizleri	Test dışı doğrulama hedefleri
Test ortamı tanımı ve SECI	Sonuçların yeniden üretilebilirliği
Araç kalifikasyon verisi (gerekliyorsa)	Araç çıktısına duyulan güvenin dayanağı
Problem raporları	Bulunan hataların sistematik yönetildiği
Gözden geçirme ve kalite güvencesi kayıtları	Doğrulama sürecinin de denetlendiği

Sık görülen bulgular

Bulgu	Tipik nedeni	Önlem
Beklenen sonuçlar kodun çıktısından alınmış	Testin kod yazıldıktan sonra, koda bakarak yazılması	Beklenen sonucun kaynağını test durumunda gereksinime bağlamak; gözden geçirmede sormak
Gürbüzlük testleri yetersiz	Testin yalnızca "mutlu yol" için yazılması	Gürbüzlük kategorilerini doğrulama planına ve test gözden geçirme kontrol listesine koymak
Sonuçlar temel çizgideki sürümle eşleşmiyor	Kod değiştikten sonra testlerin yeniden koşulmaması	Kredi amaçlı koşulları yalnızca temel çizgiye alınmış sürümlerle yapmak
Kapsam boşlukları "analizle karşılandı" deyip geçilmiş	Analiz kaydının ve gerekçenin yazılmaması	Her boşluk için tekil, gözden geçirilmiş bir çözüm kaydı
Yapısal kapsam, koda bakarak yazılmış testlerle doldurulmuş	Boşluğun nedenini aramak yerine yüzdeyi kapatma isteği	Boşluğu önce sınıflandırmak; eklenen her testi bir gereksinime izlemek
Başarısız test yeniden koşulup geçmiş, geride kayıt yok	Hatanın testte mi kodda mı olduğu kaydedilmeden düzeltilmesi	Kredi amaçlı koşudaki her başarısızlığa problem raporu; kapanışta yeniden doğrulama kaydı
Hedef dışı ortamda test, gerekçesiz	Hedef donanımın geç gelmesi	Ortam farklarını analiz eden ve kritik testleri hedefte tekrarlayan bir plan
Araç kalifikasyonu unutulmuş	Test aracının "yalnızca yardımcı" sayılması	Aracın çıktısı bir doğrulama adımının yerine geçiyor mu sorusunu SOI-1'de yanıtlamak

Denetimden sonra

SOI-3 bulguları genellikle ek test, ek analiz ya da yeniden koşu gerektirir ve takvimi doğrudan etkiler. Düzeltici faaliyetler (corrective action), SOI-4'e kadar tamamlanabilecek biçimde planlanmalı; her biri, hangi kanıtın hangi sürümle yeniden üretileceğini açıkça söylemelidir. SOI-4'te önceki aşamaların bulgularının kapanmış olması beklenir; bu yüzden SOI-3'ten kalan her açık madde, son denetimin takvimine doğrudan yazılmış bir iştir.

İlgili bölümler

- [9. Yazılım Doğrulama](#)
- [17. Kapsanmayan Kodlar: Ölü, Gereksiz ve Devre Dışı Bırakılmış Kodlar](#)
- [13. DO-330 ve Yazılım Aracı Kalifikasyonu](#)
- Önceki aşama: [SW SOI-2](#) · Sonraki aşama: [SW SOI-4](#)

SW SOI-4

SOI-4, yazılımın sertifikasyona sunulmadan önceki son denetimidir: tüm faaliyetler tamamlandı mı, önceki aşamaların bulguları kapandı mı, teslim edilen yazılım tam olarak tanımlanmış ve yeniden üretilebilir mi, açık kalan problem raporları kabul edilebilir mi? Bu sayfa, son denetim öncesinde sunum paketini sınamak için bir kontrol listesidir.

Katılım aşaması (Stage of Involvement, SOI) denetimlerinin genel mantığı ve yazılım başarı özetinin (Software Accomplishment Summary, SAS) içeriği [12. Sertifikasyon İrtibatı](#) bölümünde anlatılır. Buradaki maddeler bir otorite listesinin çevirisi değil, sahada işe yarayan hazırlık sorularıdır.

💡 SON DENETİMDE YENİ KONU AÇILMAZ

SOI-4'ün iyi geçmesinin sırrı, SOI-4'te yeni bir şey göstermemektir. Bu aşamada otorite, daha önce gördüğü sürecin sonuna kadar aynı disiplinle işletildiğini ve sunulan belgelerin birbiriyle, kayıtlarla ve gerçek yazılımla tutarlı olduğunu doğrular.

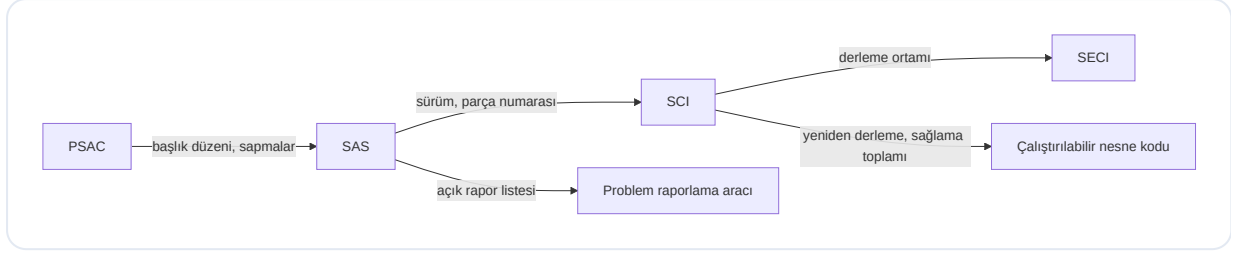
Ne zaman yapılır?

SOI-4, yazılım yaşam döngüsü faaliyetleri tamamlandığında ve sertifikasyona esas sürüm belirlendiğinde yapılır. Tipik giriş kriterleri:

- ☐ SOI-1, SOI-2 ve SOI-3 bulgularının tamamı kapatılmış ve kapanış kanıtları dosyalanmıştır.
- ☐ Sertifikasyona esas yazılım sürümü son temel çizgiye (baseline) alınmıştır.
- ☐ Doğrulama bu sürüm için tamamlanmış, sonuçları gözden geçirilmiştir.
- ☐ SAS ve yazılım konfigürasyon indeksi (Software Configuration Index, SCI) yayımlanmıştır.
- ☐ Kalite güvencesi yazılım uygunluk gözden geçirmesini (software conformity review) tamamlamıştır.
- ☐ Açık problem raporlarının her biri sınıflandırılmış, emniyet ve işlev etkisi değerlendirilmiştir.

Son paket: otoritenin baktığı noktalar

Son paketin belgeleri birbirine çapraz kontrollerle bağlıdır: yazılım sertifikasyon planında (Plan for Software Aspects of Certification, PSAC) verilen söz SAS'ta karşılığını bulur, SAS'ın andığı sürüm SCI'de tanımlanır, SCI'nin tanımladığı yazılım da yazılım yaşam döngüsü ortam konfigürasyon indeksinde (Software Life Cycle Environment Configuration Index, SECI) kayıtlı ortamda yeniden üretilir. Aşağıdaki sorular bu bağların her birini sınar.



Önceki aşamaların kapanışı

- ☐ Her önceki bulgu için ne yapıldığı, hangi kanıtla kapatıldığı ve kimin onayladığı tek bir listede görülebiliyor mu?
- ☐ Bir bulgunun düzeltilmesi başka bir veriyi değiştirdiyse o veri yeniden doğrulanmış mı?

Doğrulamanın tamamlanması

SOI-3 doğrulamanın bir kesitini görür; SOI-4'te sorulan, o günden sonra yapılan değişikliklerle birlikte doğrulamanın teslim edilen sürüm için hâlâ geçerli olup olmadığıdır.

- ☐ Kredi için sunulan test ve analiz sonuçları son temel çizgideki çalıştırılabilir nesne kodu (executable object code) için mi geçerli; daha eski bir derlemede koşulan testler için değişiklik etki analizine (change impact analysis) dayanan bir yeniden test gerekçesi yazılmış mı?
- ☐ SOI-3'te açık kalan gereksinim kapsamı ve yapısal kapsam boşluklarının hepsi kapanmış mı?
- ☐ En kötü durum yürütme süresi (worst-case execution time, WCET), yığın ve bellek kullanımı analizleri son sürüm için güncel mi; kalan paylar SAS'ta yazılı mı?
- ☐ Kalifiye araçların kalifikasyon verisi tamamlanmış mı; kullanılan araç sürümleri SECI'dekilerle aynı mı; araçların bilinen problemleri ve işlevsel kısıtları yazılım açısından değerlendirilip SAS'ta raporlanmış mı?

Yazılım başarı özeti (SAS)

- ☐ SAS, PSAC'ta verilen sözlerle aynı başlık düzeninde karşılaştırılabilir mi?
- ☐ Plandan yapılan tüm sapmalar, gerekçeleri ve onay kayıtlarıyla birlikte yazılmış mı?
- ☐ Yazılımın karşıladığı hedeflere ilişkin uyum beyanı açık mı; SAS'ta yazılı sapmalar, sınırlamalar ve açık problem raporlarıyla tutarlı mı?
- ☐ SAS, geçerli SCI'yi ve SECI'yi kimlik ve sürümleriyle açıkça anıyor mu; SAS'taki yazılım sürümü, parça numarası ve belge sürümleri SCI ile birebir aynı mı?

Konfigürasyon ve yeniden üretilebilirlik

- ☐ SCI, çalıştırılabilir nesne kodunu ve onu üreten kaynak dosyalarını sürümleriyle tanımlıyor mu?

- ☐ SECI, derleyiciyi, bağlayıcıyı, derleme seçeneklerini ve kalifiye araçları sürümleriyle listeliyor mu?
- ☐ Belgelenen talimatlarla temiz bir ortamda yeniden derleme yapıldığında aynı çalıştırılabilir nesne kodu elde ediliyor mu (sağlama toplamı karşılaştırmasıyla); fark varsa kaynağı belirlenip açıklanmış mı?
- ☐ Yükleme talimatları ve yüklenebilir yazılım parçasının (loadable software part, LSP) tanımlaması doğru mu; hedefe yüklenen sürümün doğruluğu nasıl kontrol ediliyor?
- ☐ Yaşam döngüsü verisi arşivlenmiş ve arşivden geri alınabildiği denenmiş mi?

Açık problem raporları

Açık raporların sınıfları ve değerlendirme soruları [9. Yazılım Doğrulama](#) bölümünde tanımlanır; burada yalnızca son pakette aranan tutarlılık sorulur.

- ☐ SAS'taki açık problem raporu listesi, problem raporlama aracındaki kayıtlarla aynı mı; liste SCI'de ya da sürüm notunda da veriliyorsa oradaki de aynı mı?
- ☐ Her açık rapor, planlarda tanımlı ve FAA AC 20-189 / EASA AMC 20-189 ile uyumlu şemaya göre tek bir sınıfa atanmış mı: önemli (significant), işlevsel (functional), süreç (process) ya da yaşam döngüsü verisi (life-cycle data)?
- ☐ Her açık rapor için etkilenen işlev, getirdiği kısıt ve neden bu hâliyle kabul edilebilir olduğu yazılmış mı?
- ☐ "Önemli" sınıfında açık rapor kalmış mı; kaldıysa sistem düzeyindeki hafifletmelere ve işletme kısıtlarına dayanan yeterli bir gerekçesi var mı?
- ☐ Kapatılmış raporların kapanış kanıtı (düzeltme, yeniden doğrulama) izlenebilir mi?

Kalite güvencesi

Yazılım uygunluk gözden geçirmesi, kalite güvencesinin sertifikasyona sunulan sürüm için yaptığı son kontroldür ve Seviye A'dan Seviye D'ye kadar her yazılım seviyesinde aranır. Yeni bir teknik doğrulama değildir; var olan kayıtların eksiksiz ve birbiriyle tutarlı olduğunu üç yönden teyit eder:

Yön	Sorulan
Süreç	Kredi için planlanan faaliyetler tamamlanmış ve kayıtları duruyor mu; kayda geçmemiş ya da onaylanmamış bir gereksinim sapması kalmış mı?
Veri	Yaşam döngüsü verisi planlara ve standartlara göre üretilmiş, kaynaklandığı gereksinimlere izlenebilir ve konfigürasyon kontrolü altında mı; problem raporları (önceki bir uygunluk gözden geçirmesinden ertelenenler dahil) değerlendirilmiş ve durumları kayıtlı mı?
Ürün	Çalıştırılabilir nesne kodu ile varsa parametre verisi ögesi (parameter data item, PDI) dosyaları arşivlenmiş kaynak koddan yeniden üretilebiliyor ve yayımlanmış yükleme talimatıyla yüklenebiliyor mu?

Bu yüzden son temel çizgiden önce yapılan bir uygunluk gözden geçirmesi, sunulan sürüm hakkında bir şey söylemez.

- ☐ Uygunluk gözden geçirmesi son temel çizgi üzerinde mi yapılmış; ondan sonra değişiklik olmuşsa gözden geçirme tekrarlanmış mı?
- ☐ Kalite güvencesinin açtığı uygunsuzlukların hepsi kapanmış mı?
- ☐ Tedarikçilerden gelen veri için gözetim kayıtları tamam mı?
- ☐ Önceden geliştirilmiş yazılımdan (previously developed software, PDS) kredi alınıyorsa, güncel temel çizgi önceki temel çizgiye ve onaylı değişikliklere izlenebiliyor mu?

Hazırlanacak veri paketi

Veri	Neden istenir
Yazılım başarı özeti (SAS)	Projenin PSAC'a göre nasıl tamamlandığının özeti
Yazılım konfigürasyon indeksi (SCI) ve SECI	Teslim edilen yazılımın ve yaşam döngüsü ortamının tam tanımı
Araç konfigürasyon indeksi (Tool Configuration Index, TCI) ve araç başarı özeti (Tool Accomplishment Summary, TAS)	Araç kalifikasyon seviyesi (tool qualification level, TQL) TQL-1 ile TQL-4 arasında olan araç varsa: kalifikasyonun planlandığı gibi tamamlandığı; DO-330 bu ikisinin otoriteye sunulmasını bekler
Son temel çizgiye ait doğrulama sonuçları ve kapsam analizi	Doğrulamanın teslim edilen sürüm için tamamlandığı
Açık problem raporları ve değerlendirmeleri	Bilinen kusurların kabul edilebilirliği
Önceki SOI bulgularının kapanış kayıtları	Sürecin tutarlı işletildiği
Uygunluk gözden geçirmesi kaydı	Kalite güvencesinin son kontrolü
Yeniden derleme ve yükleme kanıtı	Teslim edilen yazılımın yeniden üretilebilirliği
Konfigürasyon yönetimi ve arşiv kayıtları	Verinin korunduğu ve geri alınabildiği

Sık görülen bulgular

Bulgu	Tipik nedeni	Önlem
SAS ile SCI arasında sürüm ya da parça numarası farkı	Belgelerin farklı zamanlarda elle güncellenmesi	SCI'yi derleme altyapısından üretmek; SAS'ı SCI'ye başvurarak yazmak
Açık problem raporu listesi SAS'ta ve araçta farklı	Listenin araçtan bir kez çekilip sonra güncellenmemesi	Listeyi tek kaynaktan, son temel çizgi tarihinde üretmek
Açık raporların sınıfı ya da kabul gerekçesi eksik	Değerlendirmenin son haftaya bırakılması	Raporları açıldıkları anda sınıflandırmak; açık kalacakları düzenli gözden geçirmek

Bulgu	Tipik nedeni	Önlem
Zamanlama ya da yığın analizi eski bir sürüme ait	Son değişikliklerden sonra analizin yenilenmemesi	Değişiklik etki analizinde analiz raporlarını da etkilenen veri olarak listelemek
Uygunluk gözden geçirmesi son değişiklikten önce yapılmış	Takvim baskısıyla gözden geçirmenin öne çekilmesi	Son değişiklikten sonra gözden geçirmenin tekrarını giriş kriteri yapmak
Yeniden derleme aynı çalıştırılabilir nesne kodunu üretmiyor	Kayıt dışı derleme seçeneği, ortam farkı, zaman damgası	Yeniden derlemeyi SOI-4'ten önce temiz ortamda denemek
Plan sapması SAS'ta yok	Sapmanın proje sırasında kayda alınmaması	Sapmaları oluştukları anda kalite güvencesi kaydına bağlamak

Denetimden sonra

SOI-4 sonunda otorite, yazılım açısından sertifikasyona engel bir durum kalmadığını kayda geçirir; yazılım onayı, ait olduğu sistemin ya da ekipmanın onay sürecinin parçası olarak tamamlanır. Bundan sonra yapılacak her değişiklik, değişiklik etki analiziyle başlayan ayrı bir süreçtir.

İlgili bölümler

- [9. Yazılım Doğrulama](#)
- [10. Yazılım Konfigürasyon Yönetimi](#)
- [11. Yazılım Kalite Güvencesi](#)
- [12. Sertifikasyon İrtibatı](#)
- [18. Sahada Yüklenebilir Yazılım](#)
- Önceki aşama: [SW SOI-3](#)