DO-178B SERTİFİKASYONUNA UYGUN OLARAK GELİŞTİRİLEN İNSANSIZ HAVA ARACI(İHA) UÇUŞ KONTROL BİLGİSAYARI YAZILIMININ DOĞRULAMA YAKLAŞIMI

İbrahim Seyfullah BABAARSLAN[1]

TUSAŞ Türk Havacılık ve Uzay Sanayii A.Ş., Ankara

ÖZET

Bu bildiride, İHA Uçuş Kontrol Bilgisayarına ait yazılımın DO-178B sertifikasyon isterlerine uygun olarak geliştirilen doğrulama yaklaşımı ile bu yaklaşımdan öğrenilmiş dersler sunulmaktadır. Yazılım doğrulama süreci, hava aracının uçuş kontrol bilgisayarının belirlenen gereksinimlere uygun davranış sergilediğini ve beklenmeyen sonuçlar üretmediğini doğrulamak için kullanılan bir süreçtir. Bildiride ilk önce yazılım mimarisi anlatılacak, daha sonra yazılım mimarisine uygun olarak geliştirilen test türlerinden bahsedilecektir. Sonrasında test durumlarının yönetimi anlatılacak, izlenebilirlik kontrollerinin nasıl yapıldığı belirtilecek, yazılım kapsama analizi konusunda yapılan çalışmalar açıklanacaktır. Son olarak da DO-178B sertifikasyon sürecinden kazanılmış dersler sunulacak ve test seviyelerinin farklı parametrelere göre karşılaştırılması yapılacaktır.

GİRİŞ

Sabit Kanatlı Askeri İHA'lar için geçerli olan STANAG 4671'e göre yazılım ve kompleks donanımların uyması gereken geliştirme seviyeleri Tablo 1 : STANAG 4671 Geliştirme Güvence Seviyesi Hedefleri Tablosunda verilmiştir. İlgili yazılımların ve birimlerin SAE-ARP-4761 "Guidelines and Methods for conducting the Safety Assessment Process on Civil Airborne Systems and Equipment" dokümanı referans alınarak gerçekleştirilen emniyet analizlerine göre yazılımlara geliştirme seviyesi hedefi atanır. Uçuş Kontrol Bilgisayarı Yazılımı yapılan emniyet analizlerine göre "Ölümcül" kategoridedir ve bu sebepten ötürü İHA'lar için en yüksek Geliştirme Seviyesi olan DAL-B uyumu göstermesi gerekmektedir.

Tablo 1: STANAG 4671 Geliştirme Güvence Seviyesi Hedefleri Tablosu

Sistem ve Mimarinin her bir birimi için Gerekli Geliştirme Güvence Seviyesi Atanması		Gerekli Geliştirme Güvence Seviyesi
Kritiklik Seviyesi	Ölümcül	DAL B
	Tehlikeli	DAL C

[1] İbrahim Seyfullah BABAARSLAN, İHA Sistemleri, Yazılım Doğrulama., E-posta: isbabaarslan@tai.com.tr

	Önemli	DAL D
	Az Önemli	DAL E
	Emniyet Etkisi Yok	DAL E

DO-178B standardı bir sistemde meydana gelen hatanın sonuçlarının kritiklik seviyesine göre yapılması gereken gereklilikleri tanımlar. Bu seviyeler, oluşan bir hatanın can kaybı ve ciddi maddi zararlar doğurması ile hiçbir etkisinin olmaması arasında değişmektedir. Her bir seviye için beklenen isterler Tablo 2'de belirtilmiştir:

Tablo 2 : DO-178B Kritiklik Seviyelerine Göre İsterler Tablosu

DO-178 Özelliği	Seviye A	Seviye B	Seviye C	Seviye D
Bağımsızlık Seviyesi	Yüksek	Orta	Az	Çok Az
Yazılım Planları	Evet	Evet	Evet	Evet
Yazılım Standartları	Evet	Evet	Evet	Hayır
Satır Yapısal Kapsama	Evet	Evet	Evet	Hayır
Karar Yapısal Kapsama	Evet	Evet	Hayır	Hayır
UKK Yapısal Kapsama	Evet	Hayır	Hayır	Hayır
Doğrulanabilir ÜS Gereksinim	Evet	Evet	Evet	Hayır
Doğrulanabilir AS Gereksinim/Kod	Evet	Evet	Hayır	Hayır
ÜS-AS ve AS-Kod İzlenebilirliği	Evet	Evet	Evet	Hayır
AS Gereksinim Test Kapsaması	Evet	Evet	Evet	Hayır
Kod Gözden Geçirmeleri	Evet	Evet	Evet	Hayır
Konfigürasyon Yönetimi	Yüksek	Yüksek	Orta	Az
YKG Geçiş Kriterleri	Evet	Evet	Evet	Hayır

Mimari, Algoritma Doğrulama	Evet	Evet	Evet	Hayır

Bir yazılımda kritiklik seviyesi arttıkça yazılımla ilgili detaylı dokümanlar, testler ve analizler de artmaktadır. Bunların sonucundan maliyet artmaktadır. Bağımsızlık ve gözden geçirmelerin sürece dahil olmasıyla daha kontrollü bir yapı kurulmuş ve daha kaliteli ürünler ortaya çıkmıştır.

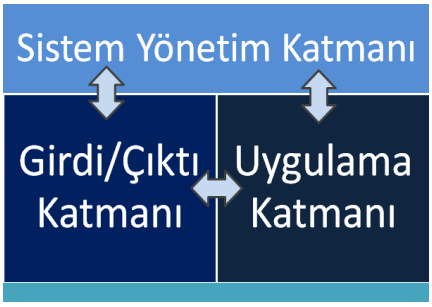
Tablo 2'de verilen DO-178B Kritiklik Seviyelerine Göre İsterler Tablosuna göre Bağımsızlık Seviyesi kriteri "Orta" olarak istenmesine rağmen, geliştirdiğimiz İHA için bu seviye "Yüksek" olarak sağlanmıştır. Geliştirici ve doğrulayıcı ekiplerin tamamen ayrılması, yapılan değişikliklerin hem yazılımcılar hem doğrulamacılar tarafından çapraz doğrulanması sağlanmıştır. Yazılım Planlarının süreçte olması kriteri yazılım geliştirme yaşam döngüsü süreçleri içerisinde yer alan tüm süreçlere ait yazılım plan dokümanlarının yazılmış ve herkesin erişebileceği şekilde açık bulunmasıyla sağlanmaktadır. Bu planlar; yazılım sertifikasyon irtibat süreci planı, yazılım kalite güvence planı, yazılım konfigürasyon yönetim planı, yazılım geliştirme planı ve yazılım doğrulama planıdır. Yazılım standartları olarak yazılım gereksinimleri standardı, yazılım kodlama standardı vs. standartlar bulunmaktadır. Yapısal kapsama analizi için Seviye B için yeterli görünen kriter olarak "Karar Yapısal Kapsama" kullanılmaktadır. Yazılımda doğrulanabilir üst seviye gereksinimler ve bu gereksinimlerle ilişkili doğrulanabilir alt seviye gereksinimler bulunmaktadır. Üst seviye gereksinimlerin alt seviye gereksinimler ile ve alt seviye gereksinimlerin kod ile izlenebilirliği kurulmuştur. Alt seviye gereksinimlerin doğrulandıkları testler ile izlenebilirlikleri kurulmuştur. Kodlar geliştirildikten sonra geliştiren kişi dışında bir kişi tarafından kod geliştirme kontrol tablosuna göre gözden geçirilmektedir. Yazılıma ait tüm ürünler(kod, test, plan dokümanları, gereksinimler vs.) konfigürasyon yönetim araçlarında tutulmakta, yetkisiz erişimleri önlemek için her bir ürün için bu ürünü değiştirme, okuma vs. yetkiler tanımlanmaktadır. Birbirine bağlı olarak ilerleyen süreçlerin kesintisiz devam edebilmesi için bir sürecin başlaması ve tamamlanması için gerekli olan geçiş kriterleri tanımlanmakta ve uygulanmaktadır. Mimari ve algoritma doğrulamaları entegre ve bölüt bazlı testler ve kod gözden geçirmeler ile sağlanmaktadır.

İHA Uçuş Kontrol Bilgisayarına ait yazılımın doğrulama yaklaşımının açıklanacağı bu bildiride öncelikle yazılım mimarisi açıklanacaktır. Ardından hava aracının yazılımına ait bölüt bazlı ve entegre testlerden bahsedilecektir. Sonrasında ilk kez geliştirilen veya güncellenen bir test durumu için test durumu yönetilme yöntemi anlatılacaktır. Yazılım test durumlarının hata ve değişiklik takibinin nasıl yapıldığı açıklanacak, izlenebilirliğin nasıl sağlandığı tanımlanacak, yazılım test durumunun en önemli etkinlik parametrelerinden olan yazılım kapsama analizi çalışmaları anlatılacaktır. Sonrasında yazılım doğrulamaya yardımcı kullanılan araçların kalifikasyon sürecinden bahsedilecektir. Son olarak da DO-178B sertifikasyon sürecinden kazanılan dersler ve yazılım doğrulama sürecinde uygulanan bölüt bazlı ve entegre test yöntemlerinin karşılaştırılması ve birlikte kullanılmasının yazılım doğrulama sürecine katkıları verilecektir.

YAZILIM MİMARİSİ

Uçuş Kontrol Bilgisayarı Yazılımı(UKBY), gerçek zamanlı işletim sistemi üzerinde çalışan, belli işlevleri yerine getirmek üzerine ayrılmış bölütlerin(partition) zaman ve adres bazında çalışmaları üzerine tasarlanmış yazılımdır. UKBY, işlevsel fonksiyonların yönetildiği Uygulama Katmanı, Hava Aracındaki diğer ekipman/altsistemlerle ve Kontrol İstasyonuyla haberleşmeyi sağlayan Girdi/Çıktı

Katmanı ve katmanların sağlık bilgilerini ve çalışma performanslarını izleyen Yönetim Katmanından oluşur. Yazılım mimarisine ait gösterim Şekil 1'de verilmektedir.



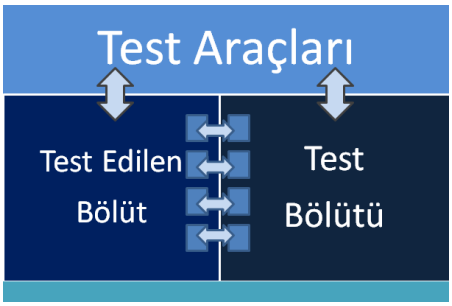
Şekil 1: Uçuş Kontrol Bilgisayarı Yazılım Mimarisi

Her bir katman işlevine göre farklı alt katmanlara ayrılmaktadır. Uygulama Katmanında model tabanlı yazılım geliştirme aracı ile kodlar üretilmektedir. Girdi/Çıktı ve Yönetim Katmanı ise manuel kodlarla yazılmıştır. Her bir katmanın işlevsel gereksinimlerini açıklayan bölüt bazlı gereksinimler ve uçuş kontrol bilgisayarının girdi/çıktı katmanlarından yazılan ve tüm bölütlerin birlikte çalışması durumundaki işlevsel gereksinimler ile uçuş kontrol bilgisayarının fonksiyonalitesi tanımlanmaktadır.

TESTLER

Bölüt Bazlı Testler

Her bir bölüt için o bölütün işlevini anlatan yazılım gereksinimleri bulunmaktadır. Bu yazılım gereksinimleri o bölüt dışındaki bölütleri yok sayarak sadece o bölütün girdisi ve çıktısı cinsinden yazılmıştır. Bölüt bazlı testler de tıpkı gereksinimleri gibi o bölüt dışındaki bölütleri yok sayarak sadece o bölütün girdi ve çıktılarını kontrol etmek üzerine kurulmuştur. Bu nedenle bölüt bazlı testlerde bir test bölütü kullanılır. Test bölütü yardımıyla test edilen bölütün paylaşılan bellek alanlarına girdiler sağlanır ve test edilen bölüt tarafından bu girdilere karşılık üretilen çıktılar kontrol edilir. Bölüt bazlı test yaklaşımı Şekil 2'de gösterilmektedir.

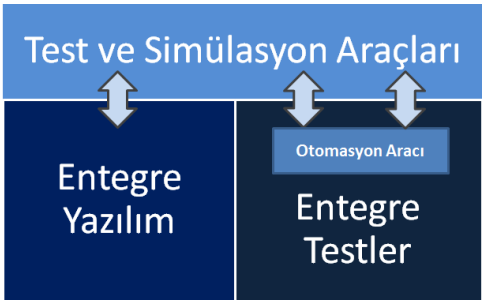


Şekil 2: Bölüt Bazlı Test Yaklaşımı

Entegrasyon Testleri

Bölüt bazlı yazılan gereksinimlerin yanında bütün bölütlerin birlikte çalışmasını sağlayacak entegre seviyede yazılmış gereksinimler de bulunmaktadır. Bu gereksinimlerin testleriyle birlikte tüm bölütlerin beraber doğru olarak çalıştığı ve beklenen sonuçları ürettiği doğrulanmış olur. Entegrasyon testlerinde test ve simülasyon araçları kullanılarak yazılıma girdiler sağlanır ve yazılımın ürettiği çıktılar yine aynı araçlar vasıtasıyla toplanarak geçti/kaldı kararları verilir. Entegre yazılım ile birlikte çalışan simülasyon aracına girdi sağlanması ve çıktıların karşılaştırılması için yazılım doğrulama

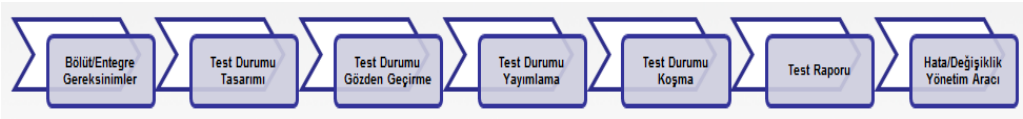
ekibi tarafından geliştirilen bir otomasyon aracı kullanılmaktadır. Entegrasyon testi yaklaşımı Şekil 3'te gösterilmektedir.



Şekil 3: Entegrasyon Testi Yaklaşımı

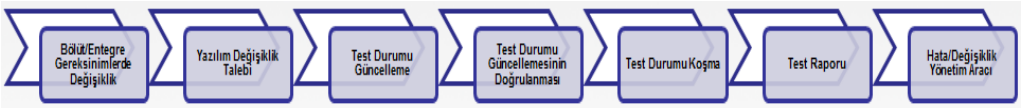
TEST DURUMU YÖNETİMİ

Yazılım geliştirme ekibi tarafından bölüt veya tüm yazılım bölütlerinin birlikte çalışması durumundaki davranışlarını açıklayan entegre seviyede yazılan gereksinimler yazılım test sürecine girdi olarak gelmektedir. Gereksinim seviyesine göre test durumu geliştirilmekte, geliştirilen test durumları eş gözden geçirme aktivitesine girerek test durumu kontrol listelerine göre gözden geçirilmektedir. İlgili test durumuna verilen yorumlar çözüldükten sonra test durumu yayımlanmakta ve bundan sonraki her güncellemede değişikliğin yazılım değişiklik isteği ile yapılması gerekmektedir. Yayımlanan test durumu hedef ortamda koşularak test raporu oluşturulmakta, raporun analizinin ardından kod, gereksinim veya test durumlarıyla ilgili yazılım değişiklik istekleri açılmakta ve hata/istek yönetim aracıyla takibi sağlanmaktadır. İlk kez geliştirilen bir test durumunun yönetimiyle ilgili aşamalar Şekil 4'te gösterilmektedir.



Şekil 4: İlk Kez Geliştirilen Test Durumunun Yönetimi

Yazılım gereksinimlerinde yapılan bir güncelleme sonucunda değişiklik yapılması gerekecek test durumları için de yazılım değişiklik isteği açılmakta ve hata/istek yönetim aracı vasıtasıyla değişikliğin, yapan kişi dışında biri tarafından kontrol edilmesi sağlanmaktadır. Yayımlanmış bir test durumunun yönetimiyle ilgili aşamalar Şekil 5'te gösterilmektedir.

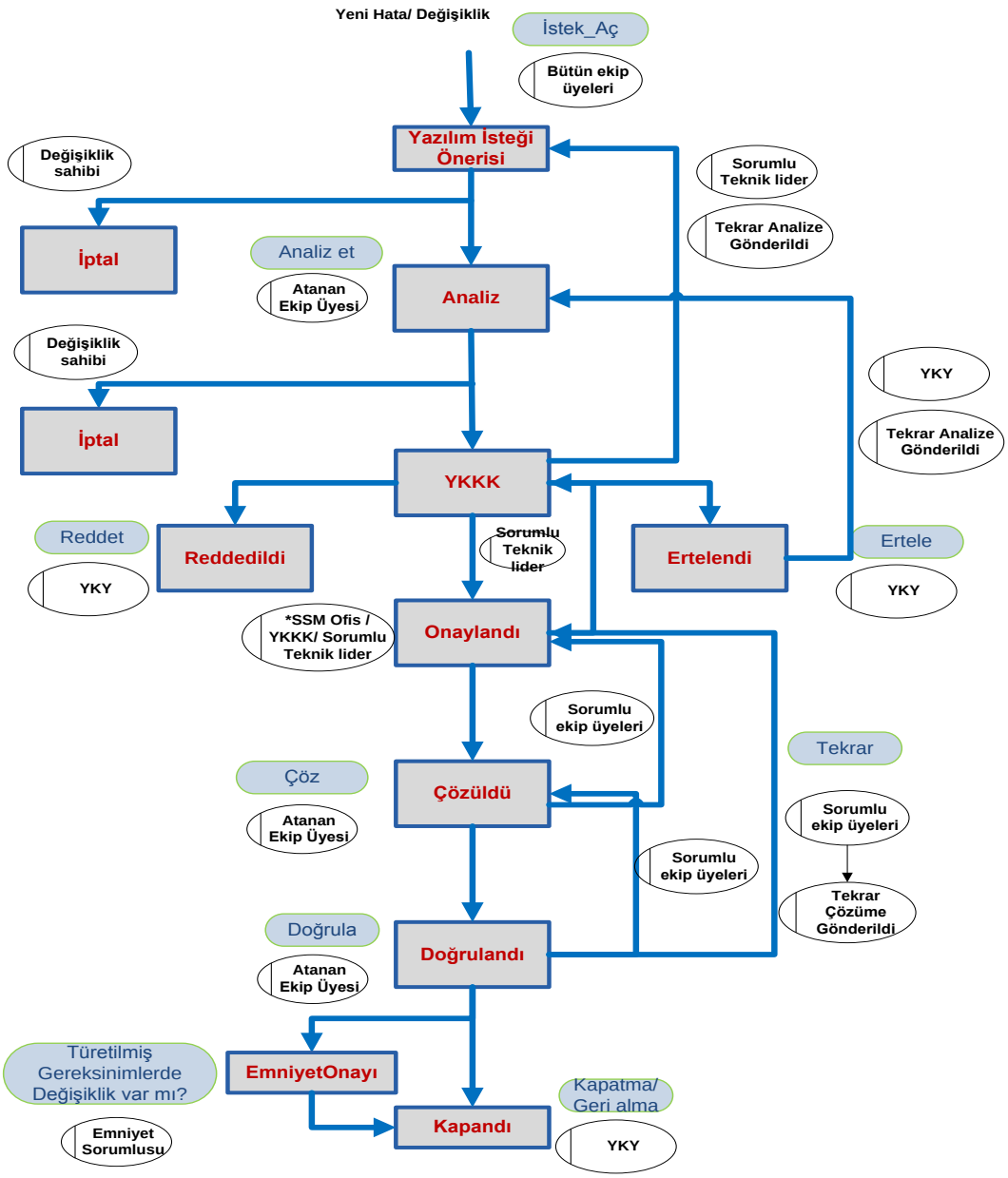


Şekil 5: Yayımlanmış Test Durumunun Yönetimi

DEĞİŞİKLİK VE HATA YÖNETİMİ

Sistem gereksinimlerinin değişmesiyle birlikte ilgili sistem gereksinimlerine karşılık yazılan yazılım gereksinimleri ve kod da değişir. Yazılım gereksinimlerinin değişmesiyle birlikte yazılım test durumlarında güncelleme ihtiyacı doğar. Yazılımın herhangi bir parçasına (kod, test durumu,

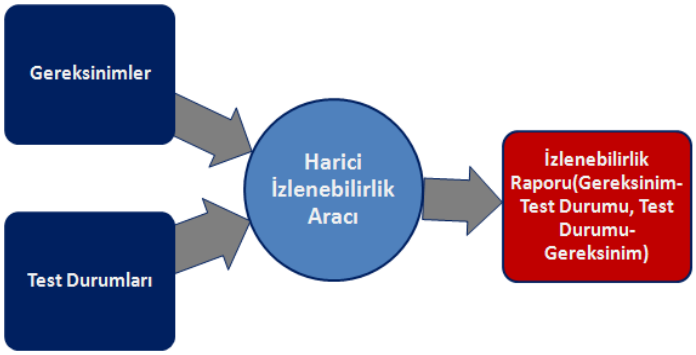
gereksinim vb.) bir değişiklik yapılması gerektiğinde ilgili parça için yazılım değişiklik talebi açılır. Yazılım değişiklik talebi, açıldıktan sonra değişiklik yapacak kişi tarafından analiz edilerek yazılım değişiklik kontrol kuruluna gönderilir veya iptal kararı alınır. Yazılım değişiklik kontrol kurulunda değişikliğin yapılmasının gerekliliği, neye dayandığı vb. konular değerlendirilerek değişikliğin kabul edilmesine veya reddedilmesine karar verilir. Kabul edilen bir değişiklik isteği, sorumlu kişi tarafından işleme alınır ve doğrulanmak üzere bu işlemi yerine getiren kişi dışında bir kişiye atanır. Değişikliğin doğru yapıldığı kontrol edildikten sonra değişiklik isteğinin süreçsel kontrolü de yazılım kalite süreç yöneticileri tarafından yapılarak istek kapatılır. Yayımlanmış bir üründe değişikliklerin kontrollü yapılabilmesi için ürün üzerinde sadece kabul edilmiş bir değişiklik talebi bulunması durumunda değişikliğe izin verilmektedir. Bu kontrol, konfigürasyon yönetim aracı ile değişiklik/hata yönetim aracının entegre çalışması ile sağlanmaktadır. Değişiklik veya hatanın yaşam döngüsü boyunca geçirdiği durumlar Şekil 6'da gösterilmektedir.



Şekil 6: Değişiklik/Hata Yaşam Döngüsü

İZLENEBİLİRLİK KONTROLÜ

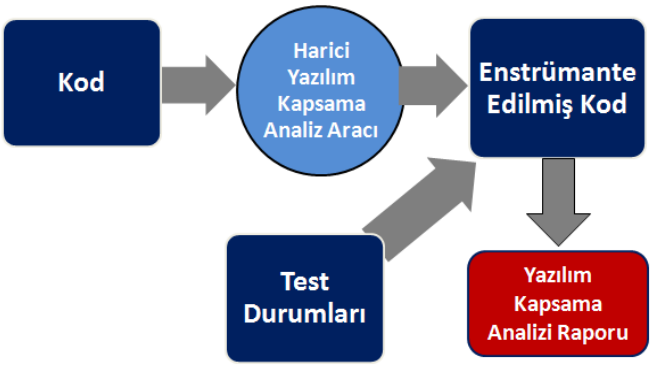
Yazılım gereksinimlerinin tamamının test edildiğinin kontrol edilmesi, test durumları ile yazılım gereksinimleri arasındaki hatalı izlenebilirliklerin saptanması, her bir gereksinimin doğrulandığı test durumunun takibi ve her bir test durumunun doğruladığı yazılım gereksinimlerinin takibi için harici bir izlenebilirlik aracı kullanılır. Bu araca, yazılım gereksinimleri ve testler eklenerek istenen izlenebilirliklerin kurulması sağlanır. Kapsanmayan yazılım gereksinimleri bulunması veya hatalı izlenebilirlik kurulması durumunda, ilgili yazılım gereksinimlerinin testinin eklenmesi veya hatalı izlenebilirliğin düzeltilmesi için yazılım test durumlarına yazılım değişiklik isteği açılması sağlanır. İzlenebilirlik kontrolünün yapısı Şekil 7'de gösterilmektedir.



Şekil 7: İzlenebilirlik Kontrolü

YAZILIM KAPSAMA ANALİZİ

Yazılım test durumlarının etkinliğinin en önemli ölçülerinden biri, yazılım kapsama analizi sonuçlarıdır. Harici bir yazılım kapsama analizi aracı ile kodun enstrümente edilmiş hali üzerinde tüm yazılım testleri koşturulur. Yazılım testleri için ifade kapsama analizi (statement coverage) kullanılmaktadır. Harici yazılım kapsama analizi aracı ile üretilen rapor incelenerek kodun yazılım testlerinin koşumu esnasında hiç koşulmayan satırları belirlenir ve bu satırlar analiz edilir. Yazılım kapsama raporunun üretilmesiyle ilgili gösterim Şekil 8'de gösterilmiştir.



Şekil 8: Yazılım Kapsama Analizi

Kapsanmayan satırların işlevselliği yazılım gereksinimleri tarafından belirtilmemiş ise bu durum yazılım gereksinimlerinde bir eksiklik olduğuna işaret eder ve yazılım gereksinimlerine yazılım değişiklik isteği açılır. Kapsanmayan satırların işlevselliğinin yazılım gereksinimlerinde belirtildiği fakat bu gereksinimleri test eden bir test bulunmadığı durum yazılım testlerinde bir eksiklik olduğuna işaret eder ve ilgili test durumlarına yazılım değişiklik isteği açılarak gereksinimin test edilmesi sağlanır. Kodda hiçbir zaman oluşma ihtimali olmayan kısımlar ölü (dead) kod olarak adlandırılır ve koddan çıkarılması için koda yazılım değişiklik isteği açılır. İlgili kod parçasının çıkarılması sonucunda etkilenebileceği düşünülen kod parçalarının testleri tekrarlanabilir. Bu durumlar dışında aktif olmayan kod, savunma amaçlı kod ve analiz/inceleme ile kapsanabilecek kod gibi sonuçlara da ulaşılır. Aktif olmayan kod, yazılım konfigürasyonuna göre gerektiğine aktif hale getirilecek koddur. Savunma amaçlı kod ise sistemin çökmesini engelleyen, koruma amacıyla yazılmış ve asıl işlevselliği yansıtmayan koddur. Analiz/inceleme ile kapsanabilecek kod, karmaşık algoritmaların bulunduğu yazılım modüllerinde manuel analiz senaryoları ile kontrol edilen koddur. Aktif olmayan kod, savunma amaçlı kod ve analiz/inceleme ile kapsanabilecek kod, gerekçeleriyle birlikte

raporlanarak kodda bırakılır. Tüm değişiklik istekleri çözüldükten sonra kapsanmayan veya açıklanmayan bir kod satırı kalmadığında yazılım kapsama analizi, yapıldığı kod sürümü için sona erer.

ARAÇ KALİFİKASYONU

Yazılım doğrulama için kullanılan araçlardaki bir hata yazılıma hata injekte etmese de yazılımda bulunan bir hatanın bulunmasını önleyebilir. Bu sebeple yazılım doğrulama için kullanılan herhangi yardımcı bir aracın doğru çalıştığını kontrol etmek gerekir. Bu sürece araç kalifikasyonu denir. Bu süreçte yazılımın doğrulanması için kullanılacak aracın gereksinimleri yazılır ve test edilir. Hataları raporlanır ve düzeltmeler yapılır. Bu süreç sonunda yazılımı doğrulamak için kullanılan aracın beklendiği gibi çalıştığından emin olunur ve otorite tarafından da yazılım doğrulama amacıyla kullanılabilir bir araç olarak kabul edilir.

DEĞERLENDİRME VE SONUÇ

Tablo 2'de verilen DO-178B Kritiklik Seviyelerine Göre İsterler Tablosuna göre Bağımsızlık Seviyesi kriteri "Orta" olarak istenmesine rağmen, ANKA-S için bu seviye "Yüksek" olarak sağlanmıştır, bu sayede hataların daha erken tespit edilmesi ve bağımsız bir kişinin diğer kişilerin ürününe bakarak farklı bakış açıları sağlaması mümkün olmuştur. Geliştirici ve doğrulayıcı ekiplerin tamamen ayrılması, yapılan değişikliklerin hem yazılımcılar hem doğrulamacılar tarafından çapraz doğrulanması sağlanmıştır. Yazılım Planlarının süreçte olması kriteri yazılım geliştirme yaşam döngüsü süreçleri içerisinde yer alan tüm süreçlere ait yazılım plan dokümanlarının yazılmış ve herkesin erişebileceği şekilde açık bulunmasıyla sağlanmaktadır. Bu planlar; yazılım sertifikasyon irtibat süreci planı, yazılım kalite güvence planı, yazılım konfigürasyon yönetim planı, yazılım geliştirme planı ve yazılım doğrulama planıdır. Bu planlar sayesinde her bir ekibe yeni başlayan kişilerin iş akışını öğrenmesi mümkün olmuş, her bir sürecin nasıl ilerleyeceğine dair bir standardizasyon sağlanmıştır. Yazılım standartları olarak yazılım gereksinimleri standardı, yazılım kodlama standardı vs. standartlar bulunmaktadır. Bu standartların çoğu bir hatayı önlemek adına yardımcı olmuş, ayrıca belli bir standardizasyon sağlanmıştır. Yapısal kapsama analizi için Seviye B için yeterli görünen kriter olarak "Karar Yapısal Kapsama" kullanılmaktadır. Bu sayede yazılım gereksinimlerinin yazılım testleriyle kapsandığı, herhangi bir uygunsuzluk durumunda düzeltilmesi için aksiyonlar alındığına emin olunmuştur. Yazılımda doğrulanabilir üst seviye gereksinimler ve bu gereksinimlerle ilişkili doğrulanabilir alt seviye gereksinimler bulunmaktadır. Üst seviye gereksinimlerin alt seviye gereksinimler ile ve alt seviye gereksinimlerin kod ile izlenebilirliği kurulmuştur, bu sayede yazılımdan beklenen fonksiyonalitenin alt seviye gereksinimlere doğru şekilde aktığı kontrol edilmiştir. Alt seviye gereksinimlerin doğrulandıkları testler ile izlenebilirlikleri kurulmuştur, böylelikle her bir gereksinimin test edildiği ve tüm gereksinimlerin tüm koşullarının test edildiğine emin olmak için yapılan test gözden geçirmelerine uygun girdinin sağlandığı kontrol edilmiştir. Kodlar geliştirildikten sonra geliştiren kişi dışında bir kişi tarafından kod geliştirme kontrol tablosuna göre gözden geçirilmektedir. Yazılıma ait tüm ürünler(kod, test, plan dokümanları, gereksinimler vs.) konfigürasyon yönetim araçlarında tutulmakta, yetkisiz erişimleri önlemek için her bir ürün için bu ürünü değiştirme, okuma vs. yetkiler tanımlanmaktadır. Birbirine bağlı olarak ilerleyen süreçlerin kesintisiz devam edebilmesi için bir sürecin başlaması ve tamamlanması için gerekli olan

geçiş kriterleri tanımlanmakta ve uygulanmaktadır. Mimari ve algoritma doğrulamaları entegre ve bölüt seviyesi testler ve kod gözden geçirmeler ile sağlanmaktadır.

Geliştirilen test ortamlarında, yazılım geliştirme ortamına modülerlik sağlayan bölütlerin testleri ve bütün bölütlerin birlikte çalışmasını doğrulayan entegre testler gerçekleştirilmektedir. Test durumu geliştirme süresi olarak bu iki yaklaşım karşılaştırıldığında, entegre seviye testleri geliştirme süresinin bölüt bazlı testlere göre daha az olduğu görülmüştür. Bu durum her bir bölütte açıklanabilecek gereksinimlerin tek bir seviyede anlatılmış olmasından kaynaklanır. Benzer şekilde, entegre testlerde bölüt bazlı testlere göre daha az sayıda gereksinim ve test adım sayısı bulunduğundan güncelleme süre ve eforu daha kısadır.

Bölüt bazlı testler, simülasyon ortamları, gerçek ortamdaki donanım kartları vb. dış faktörlerden etkilenmediğinden otomatize test durumları geliştirmeye daha elverişli olmuş, bu sebeple test koşma süresi bakımından bölüt bazlı testler entegre testlere göre daha iyi sonuçlar vermiştir.

Bölüt bazlı testler spesifik bir alt birime yönelik kurgulandığı için bulunan hata direk olarak test edilen alt birime, bu birimle ilişkili olan diğer alt birimlere ve testlerin koşulduğu bölüte odaklanmayı sağlamaktadır. Bu da hata analizinde ve yapılması gereken iyileştirmenin yazılımın hangi bölümünde olduğunun bulunmasında kolaylık sağlamaktadır. Entegre testlerde ise bir hata durumunda hatanın hangi bölütten kaynaklandığının, simülasyon vb. ortam sorunlarının bu hataya sebep olup olmadığının belirlenememesinden dolayı analizi uzun süren bir süreç oluşmaktadır. Bu bağlamda bölüt bazlı testlerin hataya odaklanma konusunda entegre testlerden daha iyi olduğu söylenebilir.

Entegre bazlı testler kullanıcı isterlerine daha yakın olduğu için bölüt bazlı tasarımda farkedilemeyecek hataları bulma konusunda daha başarılıdır.

Sistemimizde, hem her bir bölütün işlevselliğini açıklayan bölüt bazlı gereksinimlerin bölüt bazlı testleri, hem de bütün bölütlerin birlikte çalışması durumlarının anlatıldığı entegre gereksinimlerin entegre testleri yer almaktadır. Bölüt bazlı testler ile yazılımı oluşturan parçalar tek tek ve detay seviye test edilmekte ve gizli hatalardan arındırılmaktadır. Entegre testlerde ise bölüt bazlı testlerden geçerek olgunlaşmış bileşenlerin bir bütün olarak da beklendiği şekilde çalıştığı kontrol edilmektedir.

Yapılan testlerden elde edilen bulgular Tablo 3'te her bir yaklaşımın avantajlarını ve dezavantajlarını gösterecek şekilde özetlenmiştir.

Tablo 3: Bölüt Bazlı Testler ve Entegrasyon Testleri Karşılaştırma Tablosu

Bölüt Bazlı Testler	**Entegrasyon Testleri**
Avantajlar • Otomasyonu kolaydır. • Test durumlarının koşulma süresi otomasyon yardımıyla kısadır. • Meydana gelen bir hatanın kaynağını bulmak kolaydır.	**Avantajlar** • Test durumu geliştirme süresi kısadır. • Kullanıcı isterlerine daha yakın doğrulama sağladığından muhtemel tasarım hatalarını bulmayı sağlar. • Güncelleme eforu ve süresi kısadır.
Dezavantajlar • Test durumu geliştirme süresi uzundur.	**Dezavantajlar** • Otomasyonu zordur.

• Her bir bölütün doğru çalışmasını doğrular fakat kullanıcı isterlerinin her bir bölütte doğru yapıldığını doğrulayamaz. • Güncelleme eforu ve süresi uzundur.	• Test durumlarının koşulma süresi otomasyonun daha az olması sebebiyle uzundur. • Meydana gelen bir hatanın kaynağını bulmak detaylı bir analiz gerektirebilir.