BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MODEL TABANLI GELİŞTİRME TEKNOLOJİSİNİN HAVA ARACI YAZILIMLARINDA KULLANIMI VE SERTİFİKASYONU

TUĞBA SARAÇ

YÜKSEK LİSANS TEZİ
2019

MODEL TABANLI GELİŞTİRME TEKNOLOJİSİNİN HAVA ARACI YAZILIMLARINDA KULLANIMI VE SERTİFİKASYONU

CERTIFICATION ASPECTS OF MODEL BASED DEVELOPMENT FOR AIRBORNE SOFTWARE

TUĞBA SARAÇ

Başkent Üniversitesi
Lisansüstü Eğitim Öğretim ve Sınav Yönetmeliğinin
BİLGİSAYAR Mühendisliği Anabilim Dalı İçin Öngördüğü
YÜKSEK LİSANS TEZİ
olarak hazırlanmıştır.

2019

“Model Tabanlı Geliştirme Teknolojisinin Hava Aracı Yazılımlarında Kullanımı ve Sertifikasyonu” başlıklı bu çalışma, jürimiz tarafından, 27/06/2019 tarihinde, **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI 'nda YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan (Danışman) : Prof. Dr. Mehmet Reşit TOLUN

Üye : Doç. Dr. İhsan Tolga MEDENİ

Üye : Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ

ONAY

/ /2019

Prof. Dr. Faruk ELALDI

Fen Bilimleri Enstitüsü Müdürü

BAŞKENT ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZ ÇALIŞMASI ORİJİNALLİK RAPORU

Tarih: 24/06/2019

Öğrencinin Adı, Soyadı : Tuğba SARAÇ

Öğrencinin Numarası : 21710287

Anabilim Dalı : Bilgisayar Mühendisliği

Programı : Tezli Yüksek Lisans

Danışmanın Unvanı/Adı, Soyadı : Prof Dr. Mehmet Reşit TOLUN

Tez Başlığı : Model Tabanlı Geliştirme Teknolojisinin Hava Aracı Yazılımlarında Kullanımı ve Sertifikasyonu

Yukarıda başlığı belirtilen Yüksek Lisans tez çalışmamın; Giriş, Ana Bölümler ve Sonuç Bölümünden oluşan, toplam 91 sayfalık kısmına ilişkin, 24/06/2019 tarihinde tez danışmanım tarafından Turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 3'dir.

Uygulanan filtrelemeler:

1. Kaynakça hariç
2. Alıntılar hariç
3. Beş (5) kelimeden daha az örtüşme içeren metin kısımları hariç

"Başkent Üniversitesi Enstitüleri Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Usul ve Esaslarını" inceledim ve bu uygulama esaslarında belirtilen azami benzerlik oranlarına tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrenci İmzası:…….

Onay

24 / 06 / 2019

Prof. Dr. Mehmet Reşit TOLUN

TEŞEKKÜR

Beni her zaman destekleyen sevgili anneme ve babama, bu programa başlamama vesile olan sevgili hocam Sn. Prof. Dr. Berna DENGİZ'e, tez danışmanım Sn. Prof. Dr. Mehmet Reşit TOLUN'a, bu tez çalışmamın Savunma Sanayii Başkanlığı, Başkent Üniversitesi ve TUSAŞ arasında "Savunma Sanayii için Araştırmacı Yetiştirme Programı" kapsamında kabul edilmesi konusundaki öneri ve destekleri için Sn. Ayşe Şolpan TEMİZ'e, Sn. Mehmet Yetiş UYSAL'a, Sn. Mine TEMEL'e ve Sn. Cahit MAVİLİ'ye, teknik paylaşımlarından ötürü, konu ile ilgili derin bir tecrübeye sahip olan Sn. Devrim KORKMAZ'a ve benden yardımlarını esirgemeyen çok sevgili arkadaşlarıma gönülden teşekkürlerimi sunuyorum.

Bu çalışmayı, bana her konuda sonsuz destek veren sevgili eşime armağan ediyorum.

ÖZ

MODEL TABANLI GELİŞTİRME TEKNOLOJİSİNİN HAVA ARACI YAZILIMLARINDA KULLANIMI VE SERTİFİKASYONU

Tuğba SARAÇ

Başkent Üniversitesi Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

Teknolojik gelişmelerin hızla ilerlemesi pek çok sektörde olduğu gibi havacılık sektöründe de ihtiyaç duyulan ürün ve hizmetler ile ilgili alternatiflerin artmasına imkân vermiştir. Bu durum üreticiler arasında sıkı bir rekabet ortamının oluşmasına neden olmuştur. Artan rekabet koşullarında hava aracı üreticileri, müşteri isteklerine daha kısa zamanda ve daha az maliyetle cevap vermek için yeni teknoloji arayışlarına girmişlerdir. Bu teknolojilerden birisi de, üreticilere sağladığı takvim ve maliyet avantajı nedeni ile son yıllarda havacılık sektöründe oldukça popüler hale gelen Model Tabanlı Geliştirme (MTG) teknolojisidir. MTG teknolojisi, geleneksel yazılım geliştirme yaklaşımına alternatif olarak geliştirilmiş, sistem ve yazılım seviyesindeki faaliyetlerin iç içe geçtiği yeni bir yazılım geliştirme yaklaşımıdır. Bu çalışmada, MTG teknolojisi ile geliştirilen hava aracı yazılımlarının yaşam döngüsü boyunca tamamlanması gereken faaliyetler, üretilmesi gereken veriler, hava aracı sertifikasyon süreci ve uçuş emniyeti kapsamında dikkat edilmesi gereken konular, bu konulara yönelik öneriler ve MTG teknoloji ile ilgili önemli noktaların sorgulanmasında havacılık sektörüne fayda sağlayabilecek soru listeleri sunulmuştur. Uygulama bölümünde ise, MTG teknolojisinde yapılması gereken model kapsama analizi ile yapısal kapsama analizi karşılaştırması yapılmış ve çalışmada anlatılan diğer önemli konulardan örnekler verilmiştir.

ANAHTAR SÖZCÜKLER: Model tabanlı geliştirme, hava aracı sertifikasyonu, model kapsama analizi.

Danışman: Prof. Dr. Mehmet Reşit TOLUN, Başkent Üniversitesi, Bilgisayar Mühendisliği Bölümü

ABSTRACT

The rapid improvements of technological developments have led to increase alternatives and to enable customers to access these alternatives more easily and quickly. This has led to an increase in competition among companies. Increasing competition conditions have led companies to look for alternative technologies to produce products in less time and less cost. One of these technologies is the model based development technology, which is very popular due to the time and cost advantages it provides to companies. Model based development technology is a new development technology used as an alternative approach to the traditional software development approach. Although the use of this technology in the development of non-safety critical software is older, the use of this technology in the development of safety critical software such as aircraft software is relatively new. In this study, the use of this technology in the aviation, the advantages it provides, the effects on the certification and flight safety, the issues that need to be considered in the product development processes, recommendations for these issues and a useful checklist are presented. In addition, information about model coverage analysis that should be done in model based development technology is given and compared with structural coverage analysis on an example.

KEYWORDS: Model based development, airborne software certification, model coverage analysis.

Supervisor: Prof. Dr. Mehmet Reşit TOLUN, Başkent University, Department of Computer Engineering

İÇİNDEKİLER LİSTESİ

Sayfa

ŞEKİLLER LİSTESİ

ÇİZELGELER LİSTESİ

SİMGELER VE KISALTMALAR LİSTESİ

ARP	Aerospace Recommended Practice
CS	Certification Specification
EASA	European Aviation Safety Agency
EFT	Exploration Flight Test
FAR	Federal Acquisition Regulation
FAA	Federal Aviation Administration
IEEE	Institute of Electrical and Electronics Engineers
MKA	Model Kapsama Analizi
NASA	National Aeronautics and Space Administration
PKM	Passenger Kilometer
RTCA	Radio Technical Commission for Aeronautics
YKA	Yapısal Kapsama Analizi

1 GİRİŞ

Bu tez çalışmasında, geleneksel yazılım geliştirme yaklaşımına alternatif olan MTG teknolojisi ile geliştirilen hava aracı yazılımlarının yaşam döngüsü süreçleri, süreçlerde gerçekleştirilmesi gereken aktiviteler ve üretilmesi gereken yazılım yaşam döngüsü verileri anlatılmıştır. Son yıllarda havacılık yazılımlarının geliştirilmesinde sağladığı takvim ve maliyet avantajı ile oldukça popüler olan bu yeni teknoloji, beraberinde uçuş emniyeti ve sertifikasyon sürecini olumsuz etkileyebilecek konuları da getirmiştir. Bu çalışmada, bu konular ile ilgili örnekler verilmiş ve MTG teknolojisinin hava aracı yazılımlarında sağlıklı bir şekilde kullanılabilmesi için dikkat edilmesi gereken konular ve bu konular ile ilgili öneriler sunulmuştur. İlave olarak, bu teknoloji ile yazılım geliştirecek üreticiler için konu ile ilgili önemli noktaların altını çizen, rehber niteliğinde kullanılabilecek bir soru listesi hazırlanmıştır.

Yapılan literatür taramasında, MTG teknolojisinin havacılık sektöründe kullanımı yeni olduğu için havacılık yazılımları ile ilgili çok fazla bir çalışma olmadığı gözlemlenmiştir. Mevcut çalışmalarda, bu teknolojinin kullanımında dikkate alınması gereken konulardan genellikle bir veya bir kaçının anlatıldığı, MTG yaşam döngüsünün başından sonuna kadar her bir süreçte yapılması gereken aktiviteleri, bu teknolojinin hava aracı sertifikasyon ve emniyet sürecine etkilerini açıklayan bir çalışma olmadığı görülmüştür.

Bu çalışma temel motivasyonu, MTG teknolojisi ile geliştirilecek hava aracı yazılımları için, yazılım geliştirme, doğrulama, kalite, konfigürasyon ve sertifikasyon mühendislerine, sistem ve emniyet ekibine, proje yöneticilerine ve müşterilere, uçuş emniyeti ve sertifikasyon sürecini etkileyebilecek konuları örneklerle açıklayarak farkındalık yaratmak ve havacılık sektörüne katkı sağlamaktır.

Bu çalışmanın, yalnızca havacılık sektörüne değil, tren, medikal, nükleer gibi emniyet kritik yazılım geliştirirken MTG teknolojisini kullanmayı planlayan diğer sektör kullanıcılarına da fayda sağlayacağı değerlendirilmektedir.

Türkiye'deki havacılık yazılımı geliştirilen firmalarda MTG teknolojisi ile geliştirilen az sayıda ve küçük ölçekli hava aracı yazılımları olup bu yazılımların geliştirme ve sertifikasyon süreci halen devam etmektedir. MTG teknolojisi ile sağlanan takvim ve maliyet kazancı değerlendirildiğinde, bu teknolojinin önünün çok açık olduğu, yeni başlayacak çok daha büyük ölçekli ve karmaşık yazılımlar içerecek hava aracı yazılımlarında bu teknolojinin kullanılacağı düşünülmektedir.

Ülkemizde de havacılık sektörünün neredeyse yeni tanışacağı bu teknoloji ile ilgili üreticilere yol gösterebilecek, uçuş emniyeti ve sertifikasyon sürecini etkileyebilecek konuların örnekler üzerinden anlatıldığı çalışmanın sektöre fayda sağlayacağı değerlendirilmektedir.

Bu çalışmada, geleneksel yazılım yaşam döngüsü aktivitelerinden farklı olarak MTG teknolojisinin kullanımı ile gelen aşağıdaki konular hakkında detaylı bilgiler ve örnekler verilmiştir:

- MTG teknolojisinde yazılım yaşam döngüsü
- Yapılması gereken ilave yazılım aktiviteleri
- Üretilmesi gereken ilave yaşam döngüsü verileri
- Hava aracı sertifikasyon sürecini ve uçuş emniyetini etkileyebilecek hususlar
- Önemli konuların altının çizildiği bir soru listesi

Çalışmanın birinci bölümünde giriş, ikinci bölümünde MTG teknolojisinin dünyada kullanım örnekleri, karşılaşılan problemler ve çözüm önerilerini görebilmek üzere yapılan literatür taramasına ait özet verilmiştir.

Üçüncü ve dördüncü bölümlerde, konu ile ilgili temel kavramlar ile hava aracı sertifikasyon süreci hakkında genel bilgiler verilmiştir.

Beşinci bölümde, MTG teknolojisinin tanımı, üreticilere sağladığı avantajlar, geleneksel yazılım yaşam döngüsü ile MTG teknolojisi arasındaki farklar ve MTG yaşam döngüsü tipleri açıklanmıştır.

Altıncı bölümde, yazılım geliştirme yaşam döngüsünde MTG teknolojisine özel yapılması gereken ilave aktiviteler süreç bazında tek tek detaylandırılmıştır.

Yedinci bölümde, yazılım geliştirme yaşam döngüsü boyunca üretilen verilere MTG teknolojisi kapsamında dâhil edilmesi gereken ilave içerikler anlatılmıştır. Bu bölümde ilave olarak bu teknolojinin kullanımında faydalı olabilecek sorular verilmiştir.

Sekizinci bölümde, çalışma boyunca anlatılan konular ile ilgili örnekler, geleneksel yazılım yaşam döngüsü yaklaşımında gerçekleştirilen yazılım kapsama analizine ilave olarak bu teknoloji kullanımında yapılması gereken model kapsama analizi anlatılmış ve bu iki analizin farkı örnekler üzerinden açıklanmıştır. İlave olarak, araçlar arasında model transferi durumunda modelin davranışının değişip değişmediğini kontrol etmek amacı ile kullanılabilecek bir yöntem ile ilgili bir örnek verilmiştir.

Ekler bölümünde, Simulink aracında modellenmiş bir modelde eksik model kapsaması ve türetilmiş model elemanının modelde nasıl göründüğünü örneklemek üzere ekran görüntüleri verilmiştir.

Bu tez çalışması, Savunma Sanayii Başkanlığı, TUSAŞ ve Başkent Üniversitesi arasında yapılan 201900447 numaralı sözleşme ile “Savunma Sanayii İçin Araştırmacı Yetiştirme Programı” projesi olarak kabul edilmiştir. Araştırmacı, bu tezini Başkent Üniversitesi Bilgisayar Mühendisliği bölümünde yaptığı yüksek lisans kapsamında hazırlamış olup, aynı zamanda TUSAŞ bünyesinde Uçuşa Elverişlilik ve Emniyet Müdürlüğü bölümünde Yazılım Sertifikasyon Uzmanı olarak çalışmaktadır. Bu çalışma ile havacılık sektöründe yeni bir teknoloji olan MTG teknolojisinin havacılık sektöründe kullanımına yönelik sertifikasyon ve emniyeti etkileyebilecek hususlar, geliştirici tarafından yapılması gereken çalışmalar, üretilmesi gereken veriler ve sertifikasyon uzmanının incelemesi gereken hususlar örnekler üzerinden anlatılarak sektörde farkındalık yaratılması hedeflenmiştir.

Yapılan literatür taramasına ve gözlemlenen mevcut sektör uygulamalarına göre MTG teknolojisi kullanan firmaların genellikle, sistem modellemeleri için Simulink /

MATLAB aracını, otomatik kod geliştirmek için SCADE aracını kullandığı görülmüştür.

Simulink, MathWorks firması tarafından geliştirilen ve MATLAB yazılımı ile entegre edilmiş bir simülasyon aracıdır. Simulink aracında, hiç kod yazmadan sistem tasarlanıp simülasyonlar ile doğrulanabilmektedir.

SCADE, ANSYS firmasına ait, modelden otomatik kod üreten ve MTG teknolojisini destekleyen bir araçtır. SCADE kalifiye bir araç olup, bu araç ile modelden otomatik olarak üretilen kod, emniyet kritik sistemler içeren hava araçlarında, trenlerde, otomotiv sektöründe ve nükleer uygulamalarda kullanılmaktadır.

Çalışma kapsamında bir bildiri hazırlanmış ve IEEE katalog numarası: CFP19RUA-USB ve ISBN:978-1-7281-3322-5'de yayımlanmıştır.

1.1 Literatür Çalışması

MTG teknolojisi ile geliştirilen sistemlerin hava araçlarında kullanımı, geleneksel sistem geliştirme yaklaşımına göre oldukça yenidir. Konu ile ilgili yapılan literatür taramasında dikkat çeken çalışmalar aşağıda özetlenmiştir.

Saraç [1] çalışmasında, MTG teknolojisinin hava aracı yazılımlarında kullanımı ve bu teknolojinin kullanımında uçuş emniyetini ve sertifikasyonu etkileyebilecek konular ve öneriler örnekler üzerinden açıklanmıştır.

Erkkinen ve Conard, [2] çalışmalarında MTG teknolojisi ile otomatik kod üretmenin hızlı prototip üretilebilmesi için önemli bir avantaj sağladığını belirtmişlerdir. Yaptıkları çalışmada, kod üretiminin de yer aldığı bir modelde doğrulama çalışmalarından bahsetmişlerdir. Özellikle gömülü sistemler için bu teknolojinin, model kontrolü ve hedef ortamda kullanılacak işlemci üzerinde doğrulama açısından önemli avantajlar sağladığını belirtmişlerdir.

NASA'da çalışan Pomales [3], MTG teknolojisinin ticari uçaklarda kullanımının avantajlı olup olmadığını sorgulayan bir çalışma yapmıştır. Bu çalışmada, havacılık sektöründe kısıtlı takvim ve bütçe ile geliştirilen karmaşık sistemlerin kullanımının arttığını ve rekabetin artması ile sistemlerin daha da karmaşıklaşacağını belirtmiştir. Bu koşullarda, MTG teknolojisi ve doğru araçlar

kullanılarak kurulan alt yapı ile rahatlıkla çok sayıda simülasyonlar yapılabileceği ve otomatik olarak yapılabilen doğrulama aktivitelerinin önemli avantajlar sağlayacağını ve geliştirime verimliliğini arttıracağını belirtmiştir.

Jaw et al., [4] çalışmalarında Amerika Hava Kuvvetleri Araştırma Laboratuvarı'nda kritik yazılımların doğrulaması üzerinde geliştirilen teknolojilere ait araştırma sonuçlarını sunmuşlardır. Bu çalışmada, model tabanlı bir geliştirme teknolojisinin, geliştirilen modeldeki belirsizlikleri ve modelin doğruluğunu görebilmek amacı son derece faydalı olduğunu belirtilmişlerdir. Çalışmada ilave olarak temel bileşenleri, "model doğrulama araçları", "yazılım doğrulama araçları", "sistem testleri" olan bir doğrulanma süreci önerilmiş ve bir senaryo üzerinden bu metodolojinin kabiliyetleri anlatılmıştır.

Bhatt et al [5], çalışmalarında MTG teknolojisinin Honeywell firmasında geliştirilen hava aracı yazılımlarındaki uygulamasını anlatmışlardır. Bu çalışmada, gereksinim yönetim aracı olan DOORS ile MTG teknolojisinde havacılık sektöründe simülasyon amaçlı kullanılan Simulink ve otomatik kod geliştirme amaçlı kullanılan SCADE araçları arasında veri alış verişi ve bağlantı kurulabilmesinin özellikle izlenebilirlikler kapsamında önemli katkılar sağladığı vurgulanmıştır. MTG teknolojisinin sağladığı otomasyon avantajından ve bu teknoloji ile yapılan geliştirmelerde uygulamada yaşadıkları problemlerden de bahsedilmiştir.

Bouquet et al [6], çalışmalarında MATLAB ve Simulink ile geliştirilen gömülü yazılımlar için yaşam döngüsünün aşamalarına göre dikkate alınmasının fayda yaratacağını değerlendirdikleri model kalite metriklerini (Model düzeni, gereksinimler ile izlenebilirlikler, standarda uyum, model büyüklüğü, model kapsama oranı gibi) paylaşmıştır.

Eisemann [7], çalışmasında MTG teknolojisi ile geliştirilen verilerin (tanımlama modeli, tasarım modeli gibi) geliştirilmesi, model kapsama analizi gibi aktiviteleri, Simulink aracı kullanarak örnekler üzerinden sunmuştur. Bu araç kalifiye olmadığı için üretilen kodun, tasarım modeline uygunluğunun gözden geçirilerek kullanılabileceğini belirtmiştir.

Jackson ve Heny [8], çalışmalarında NASA ve Lockheed Martin çalışanları tarafından EFT-1 hava aracı için ortak geliştirilen bir algoritmada MATLAB / Simulink araçlarını kullanarak MTG teknolojisi ile algoritma geliştirmişlerdir. Bu çalışmada geliştirme boyunca karşılaştıkları sorunları paylaşmışlardır. Bu çalışmaya göre, karşılaştıkları en büyük sorunlar, mühendislerin bu teknolojiyi destekleyen araçları kullanmayı öğrenmeleri için harcanan zaman ve konfigürasyon yönetim sorunlarıdır. MTG teknolojisin sağladığı en önemli avantajın el ile kod ve test prosedürü geliştirmeden, otomatik kod ve test prosedürü üretimi ile sağlanan kazanç ile model standardına uyumluluğu kontrol etmek amacı ile kullanılan araçların (Model Advisor gibi) sağladığı kazançlar olduğu belirtilmiştir.

Pettit ve Mezcciani [9], çalışmalarında MTG teknolojisindeki en önemli zorlukların başında, geliştirme ekibindeki herkesin aynı geliştirme yaklaşımına sahip olamamasının getirdiği zorluk olduğunu belirtmiştir. Böylelikle farklı yazarlar tarafından geliştirilen modellerin, tamlığının, standarda uyumluluğunun, doğruluğunun ve kalite karakteristiklerinin farklı olabileceği belirtilmiştir. İkinci olarak, birden faklı modelleme aracı ile alan uzmanları tarafından geliştirilen modellerin tek bir modelde entegrasyonu esnasında çıkabilecek problemlerden bahsedilmiştir. Son olarak da, otomatik geliştirme aracı kullanılmadığı durumlarda kalifiye olmayan araçtan üretilen kod ile model uyumluluğu sorunlarından bahsedilmiştir.

Kramer ve arkadaşları [10], çalışmalarında Mayıs – Aralık 2016 tarihleri arasında MTG teknolojisi ile geliştirilen yazılımlar kapsamında (hava aracı yazılımı değil) 32 soruluk bir anket düzenlemişlerdir. Bu soruların arasında, MTG teknolojisinden beklentilerin neler olduğu, bu beklentilerin karşılanıp karşılanmadığı, bu teknolojinin en büyük zorlukları, kullanılan test yaklaşımları gibi sorulara verilen cevaplar grafik üzerinden gösterilmiştir.

2 TEMEL TANIM VE KAVRAMLAR

Tezde geçen temel tanım ve kavramlar aşağıda verilmiştir.

Emniyet: Ölüm, yaralanma ve mesleki hastalıklara, cihaz/mal hasarına veya kaybına, çevre hasarına neden olan koşullarından arınmış olma durumudur [11].

Emniyet Kritik Yazılım: Operasyonu veya hatası ölüme yol açabilen veya çevresine önemli ölçüde zarar veren yazılımdır.

Geleneksel Yazılım Geliştirme Yaklaşımı: Planlama fazı ile başlayan, analiz (gereksinim oluşturma), tasarım, kodlama, test ve entegrasyon fazı ile devam eden şelale yaklaşımıdır.

Model: Analiz, doğrulama, simülasyon, kod oluşturma veya bunların kombinasyonu için kullanılmak üzere sistemden yazılıma akacak öğelerin soyut gösterimidir [12].

Model Tabanlı Geliştirme ve Doğrulama: Modelin, yazılım gereksinimleri ve tasarımını temsil ettiği ve geliştirilen model ile yazılım geliştirme ve doğrulama süreçlerinin yürütüldüğü bir teknolojidir [12].

Model Simülasyonu: Bir model simülatörü kullanarak modelin davranışını görme aktivitesidir.

Otomatik Kod Üretimi: Modelleme araçları ile hazırlanan modellerden, C, C++, Ada gibi programlama dillerinde otomatik kod üreten araçlarla kod geliştirilmesidir.

Sertifikasyon: Hava aracının uçuşa elverişliliğinin belirlenmesi amacıyla, tasarlanıp geliştirilmesi veya modifikasyona tabi tutulması sürecinde belirli uçuşa elverişlilik ölçütlerine uyumluluğunun yetkili bir otorite tarafından doğrulandığı inceleme, test ve değerlendirme sürecidir.

Havacılık Kuralları: Tasarlanan hava aracının uçuşa elverişliliğinin ve emniyetinin sağlanması için karşılanması gereken ve otoriteler tarafından tanımlanmış kurallardır.

Model Standardı: Bu standart MTG teknolojisi kullanımına özel bir standart olup, modelleme dili, yöntemler, kullanılacak araca ait kısıtlar, karmaşıklık ölçütleri gibi konuları tanımlayan kurallar bütünüdür. Bu standardın sertifikasyon kapsamında beklenen çerçevesi DO-331 rehber dokümanında belirtilmiştir [11] .

Sistem Emniyeti: Bir sistemin geliştirme ve hizmet hayatının bütün fazları boyunca, operasyonel verimlilik, uygunluk, zaman ve maliyet sınırları içerisinde, kabul edilebilir risk seviyesini elde edebilmek için uygulanan mühendislik ve yönetim ilkeleri, kriterleri ve tekniklerinin bütünüdür.

Tip Sertifikası: Hava aracı tasarımının ilgili havacılık kurallarına uyumluluğu sertifikasyon otoritesine kanıtlandıktan sonra, sertifikasyon otoritesi tarafından ilgili hava aracı tasarımı içim yayımlanan sertifikadır.

Tanımlama Modeli: MTG teknolojisinde geçen bu kavram yazılımın, fonksiyonel, performans, ara yüz veya emniyet karakteristiklerini içeren üst seviye gereksinimleri gösteren modeldir [11].

Tasarım Modeli: MTG teknolojisinde geçen bu kavram, alt seviye yazılım gereksinimlerini, yazılım mimarisini, veri yapılarını, varsa algoritmaları, kontrol ve veri akışını içeren modeldir. Bu teknolojide kaynak kod genellikle tasarım modelinden üretilir [11].

Uçuş Emniyeti: Kişilere ve mallara gelebilecek zararlara ait riskler ile tehlikelerin tanımlanması ve risk yönetim süreci ile sürekli olarak kabul edilebilir seviyeye indirilmesi ve bu seviyede veya altında tutulabilmesi durumudur.

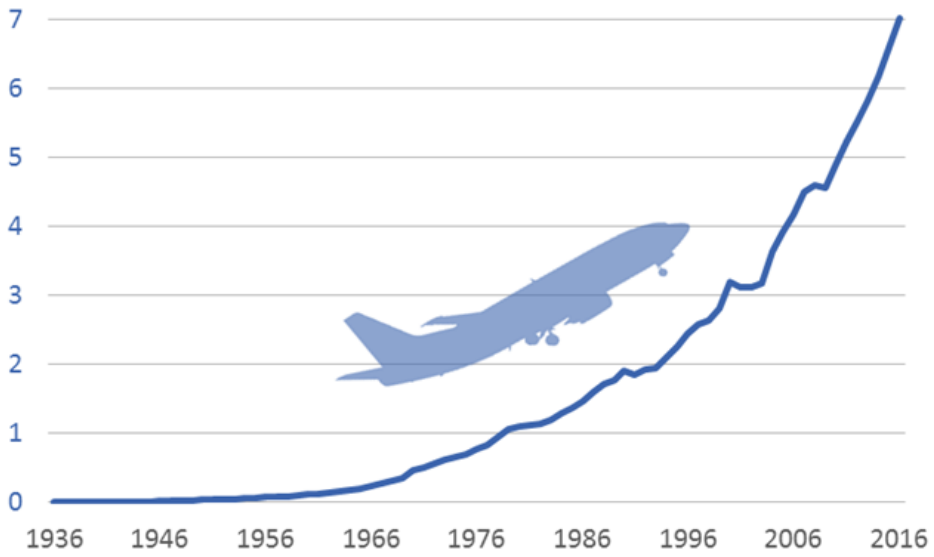
Uçuşa Elverişlilik: Uçuşa elverişlilik, bir hava aracının, uçuş ekibi, yer ekibi, yolcular ve uçuş yapan diğer hava araçlarını tehlike atmadan, onaylanmış kullanım şartları ve sınırlandırmalar içerisinde emniyetle uçuşunu başlatabilme, sürdürebilme ve sonlandırabilmesi durumudur.

3 HAVA ARACI SERTİFİKASYONU HAKKINDA GENEL BİLGİLER

İnsanlığın 19. Yüzyılın sonlarında başlayan uçuş serüveni, 17 Aralık 1903 yılında Orville ve Wilbur Wright kardeşler tarafından gerçekleştirilen ilk uçuş ile tarihe geçmiştir. 1909 tarihinde, Louis Bleriot, Bleriot XI tarafından İngiliz Kanalı geçilmiştir. 1914 yılında 1. Dünya Savaşı ile uçakların endüstriyelleştirilmesi başlamıştır. 1927 yılında 12 kişi kapasiteli ilk yolcu uçağı Ford Trimotor tarafından geliştirilmiştir. 1939 yılında ilk jet uçağı Heinkel He 178 üretilmiştir. Bell X-1 (XS-1) ile ilk kez 1947 yılında ses hızı aşılmıştır. Bu gelişmeleri takiben 1958 yılında, Amerika'da Federal Havacılık Dairesi FAA (Federal Aviation Administration) kurulmuştur.

Dünyaya paralel olarak aynı yıllarda, Türkiye'de de havacılık sektöründe önemli gelişmeler olmuştur. 1912 yılında Sefaköy'de ilk havacılık çalışmaları başlamış ve Yeşilköy'de ilk uçuş okulu açılmıştır. 23 Nisan 1920 tarihinde ilk hava teşkilatı olan Hava Kuvvetleri Şubesi kurulmuştur. Bu gelişmeyi 1925 yılında bugünkü Türk Hava Kurumu'nun kurulması izlemiştir. Aynı sene Vecihi Hürkuş ilk Türk uçağını imal ederek, 28 Ocak 1925 tarihinde uçurmuştur. Uçağın "Uçuşa Elverişlilik" sertifikası ise, uçağın teknik özelliklerinin uçuşa elverişliliğini değerlendirecek kimse olmadığı için Türkiye'de verilememiştir. 1931'de uçağın montajını sökerek tren ile Çekoslavakya'ya gönderen Vecihi Hürkuş, gerekli kontrollerin ardından uçuş iznini almıştır. 1933 yılında Havayolları Devlet İşletme İdaresi, bugünkü adı ile Türk Hava Yolları kurulmuştur. Aynı sene içerisinde, Türk Hava Postaları isimli 5 uçaklı filo ile ilk sivil hava taşımacılığı gerçekleşmiştir. 1936 yılında Nuri Demirağ, şimdiki Atatürk Havalimanı'nın yerinde uçuş sahası ve hangarlar yaptırmış ve pilot yetiştirmek üzere havacılık okulu olan Gök Okulu'nu kurmuştur. İlk uçak mühendislerinden Selahattin Reşit Alan'ın çizdiği uçak ve planörlerin planları ile 1936'da ilk tek motorlu uçak, 1938'de de ilk çift motorlu yolcu uçağı üretilmiştir. 1944 yılında Hava Kuvvetleri Komutanlığı kurulmuştur. THY'nin ilk yurt dışı seferi, 1947 yılında Ankara-İstanbul-Atina olarak tarihe geçmiştir.

Yukarıda verilen kısa tarihçeden görülebileceği gibi havacılık sektöründe kısa sürede önemli gelişmeler yaşanmıştır.

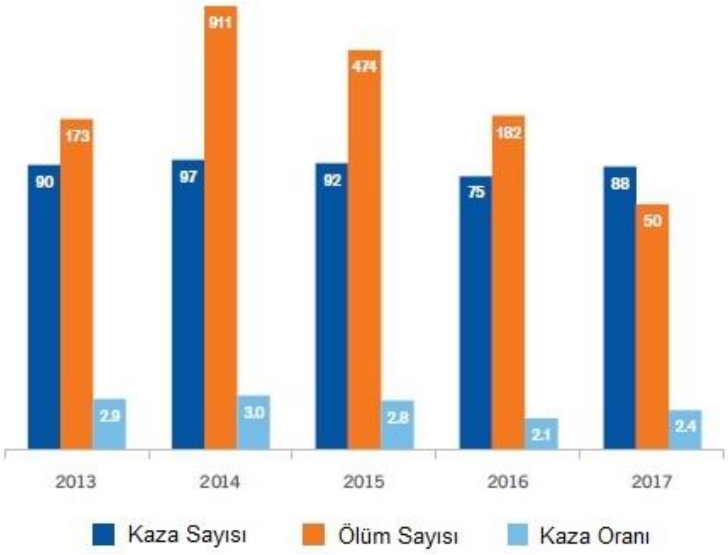


Şekil 3.1 1936-2016 yılları arasında hesaplanan PKM (Trilyon / Yıl) [13]

Teknolojinin de hızla ilerlemesi ile günümüzde son derece kabiliyetli, hızlı, performanslı ve kapasiteli hava araçları geliştirilmiş ve bu araçlar hem taşımacılık hem de savunma sanayiinde yaygın bir şekilde kullanılmaktadır. Şekil 3.1'de seyahat eden yolcu sayısı ve seyahat edilen kilometrenin çarpımı ile elde edilen PKM (Passenger Kilometer) değerindeki ivme, hava araçlarının (uçak, helikopter gibi) günlük hayatta giderek daha fazla kullanıldığının bir göstergesidir [13]. Havacılık sektöründeki ve hava araçlarının kullanımındaki bu ilerlemenin, gelişen teknoloji ve tecrübeli mühendislerin artışına paralel olarak devam edeceği şüphesizdir.

Geçen yüzyıl içerisinde havacılık sektöründeki sevindirici gelişmeler ve hava araçlarının sunduğu hizmetlerin yanında, maalesef ölümlü hava aracı kazaları da yaşanmaktadır. 1908 yılında yaşanan ilk kayıtlı uçak kazasında, Thomas Selfridge hayatını kaybetmiş Orville Wright ise ağır yaralanmıştır [14].

Son dönemlerdeki en ağır bilançolu uçak kazası 10 Mart 2019 tarihinde gerçekleşen ve South East of Addis Ababa Bole hava alanından kalktıktan kısa bir süre sonra düşen Etiyopya Hava yollarına aittir. Bu kazada, Boeing 737-8MAX uçağı düşmüş ve uçakta bulunan 157 kişi yaşamını yitirmiştir [15].



Şekil 3.2 2013-2017 yılları arası ticari uçuşlara ait kaza raporu [15]

Kuralları kan ile yazılan havacılık tarihinde maalesef pek çok hava aracı kazası vardır. Şekil 3.2'de 2013-2017 yılları arasındaki ticari uçuşlara ait kaza raporu görünmektedir [16]. Hava araçlarının emniyetli bir şekilde kullanılması ve uçuşa elverişliliği için sertifikasyon otoriteleri tarafından bir takım havacılık kuralları tanımlanmış ve düzenlemeler yapılmıştır.

3.1 Havacılık Kuralları ve Düzenlemeleri

Hava aracının servis hayatı boyunca emniyetli uçuş gerçekleştirebilmesi ve risklerin kabul edilebilir seviyeye çekilebilmesi için havacılık kurallarına uyulması ve sistematik bir emniyet sürecinin tasarım aşamasından itibaren işletilmesi gerekmektedir.

Havacılık düzenlemelerine ilk olarak 1880 yılında balonlarla ilgili ilk kurallar tanımlanarak başlanmıştır. O tarihten sonra farklı zamanlarda özellikle Amerika'da ve Avrupa'da önemli çalışmalar ve konferanslar yapılmıştır. Bu çalışmaların en önemlileri arasında, 18 Avrupa ülkesinin katılımı ile 1910 yılında Paris'te gerçekleştirilen ilk uluslararası havacılık seyrüsefer konferansı toplantısı, 1938 yılındaki Amerikan CAA (Civil Aeronautics Auhority)'nin kurulması, 1947 yılında ICAO (International Civil Aviation Organisation)'nin kurulması, 1958 yılında Amerika'da FAA'nın kurulması, 1970 yılında Avrupa' da Joint Airworthiness

Authorities'nin kurulması, 2003 yılında Avrupa Birliği tarafından EASA'nın (European Aviation Safety Agency) oluşturulması sayılabilir.

Bu konferans ve çalışmaların temel amacı, sivil havacılık ile ilgili uluslararası standardizasyon ve harmonizasyonu sağlamaktır. Bu çalışmaların sonucunda sivil havacılık otoriteleri tarafından günümüz havacılık kural ve düzenlemeleri oluşturulmuştur. Bu kurallar uçakların kategorilerine göre değişmektedir (Çizelge 3.1). Örneğin, pilot harici 9 yolcu taşıyan, maksimum kalkış ağırlığı 12500 pound (5670 kg) 'dan az normal, hizmete özel, akrobatik uçaklar ve pilot harici 19 veya az yolcu taşıyan maksimum kalkış ağırlığı 19000 pound (8618 kg)'dan az olan uçaklar "Part 23" tipi küçük uçak olarak tanımlanmıştır. Bu kategoride uçak geliştirecek üreticilerin uyması gereken havacılık kuralları, sertifikasyon otoritesi EASA ise CS 23, FAA ise FAR 23'tür.

Benzer şekilde otoriteler tarafından büyük uçaklar, hafif uçaklar, hafif döner kanatlar, küçük döner kanatlar, büyük döner kanatlar, motor ve pervane için de ayrı ayrı havacılık kuralları tanımlanmıştır. Hava aracı kategorileri ve uyulması gereken uçuşa elverişlilik standardı (havacılık kuralı) Çizelge 3.1' de verilmiştir. CS, EASA tarafından tanımlanan kurallar, FAR ise FAA tarafından tanımlanan kurallar olup bu kurullar birbirleri ile aynıdır.

Sivil havacılık talimatına göre 150 kg'dan fazla olan uçangözler (drone) sertifikasyon sürecine tabidirler. Bu kapsamda yeni geliştirilecek uçangözler için emniyet seviyesi atanır ise (örneğin insanlı bir bölgede kullanılacak uçangözün düşmesi insanlara zarar verebilir) ilgili havacılık kurallarının de işletilmesi gerekecektir.

Hava araçları ile ilgili havacılık kuralları belli iken uçangözler için havacılık kuralları henüz tam olarak netleşmemiş olup, FAA, EASA gibi otoriteler konu üzerinde çalışmaları devam etmektedirler. Bu kurallar netleştikten sonra MTG teknolojisi ile geliştirilen uçangöz yazılımları için uyulması gereken hava aracı kuralları getirilirse, bu kurallara uygun geliştirme yapılması ve otoriteden izin alınması gerekebilir.

Çizelge 3.1 Uçuşa elverişlilik standartları

Kategori	**EASA**	**FAA**	**Askeri**
Hafif Uçaklar	CS VLA		MIL-HDBK-516C DEF-STAND
Hafif Döner Kanatlar	CS VLR		
Küçük Uçaklar	CS 23	FAR 23	
Büyük Uçaklar	CS 25	FAR 25	
Küçük Döner Kanatlar	CS 27	FAR 27	
Büyük Döner Kanatlar	CS 29	FAR 29	
Motor	CS E	FAR 33	
Pervane	CS P	FAR 36	

Otoriteler tarafından hazırlanan havacılık kuralları, yaşanan hava aracı kazaları sonucunda yeni kurallar eklenerek güncellenmeye devam etmektedir. Örneğin, ilk ticari üretim jet motorlu uçak Havalland 106 Comet, servisinin birinci yılında iki adedi metal yorgunluğu, kare pencerelerin köşesindeki yüksek gerilimler nedeni ile uçuş esnasında uçak parçalanmıştır. Bu kazadan yola çıkarak pencereler kareden ovale çevrilerek, metal yorgunluğu ile ilgili sağlanması gereken yeni havacılık kuralları eklenmiştir.

Havacılık kuralları, hava aracını oluşturan tüm bileşenler ve hava aracı emniyetini etkileyebilecek tüm süreçler ve organizasyon için ayrı ayrı tanımlanmıştır. Örneğin hava aracının yapısal, aviyonik, hidromekanik, uçuş performansı, emniyet hedefleri, yazılım, donanım gibi ilgili tüm disiplinler için uyması gereken havacılık kuralları olduğu gibi, hava aracı üreticisinin organizasyon ve süreç alt yapısı olarak sağlaması gereken kurallar da vardır.

Havacılık otoriteleri hava aracının geliştirme fazının farklı aşamalarında üretilen verileri (tasarım, gereksinim, test sonuçları, analizler gibi) incelerler. Bu incelemenin amacı, hava aracı tasarımının, üretiminin ve yürütülen süreçlerin ilgili havacılık kurallarına uygun olup olmadığını değerlendirmektir. Otorite üretilen kanıtları inceledikten sonra kurallara uygun tasarım, geliştirme ve üretim

yapıldığını değerlendirilirse, otorite o hava aracına özel tip sertifikasını yayımlar ve hava aracı servis hayatına başlayabilir.

3.2 Yazılım ile İlgili Havacılık Kuralları

Yazılımın hava araçlarında kullanılabilmesi için sağlaması gereken temel havacılık kuralı, yazılımın "beklenen fonksiyonu icra etmesi" ve bu fonksiyonu icra ederken, bir başka sistemin çalışmasını negatif yönde etkilememesidir. Bu iki kural CS / FAR'larda 1301 ve 1309 numaralı havacılık kurallarında yer almaktadır.

Hava aracı yazılımları için bu iki kuralın uyum gösterim yöntemi, (yazılımın hava aracında kullanılabilmesi için otoriteler tarafından kabul edilebilir yöntem) geliştirilecek yazılımın DO-178C rehber dokümanına uyumlu geliştirilmesidir. Bir başka deyişle, yazılım DO-178C dokümanına uygun geliştirirse ve havacılık otoritesi yazılım kapsamında üretilen kanıtları yeterli bulursa geliştirilen yazılımın hava aracında kullanımına izin verir. Bu izin alınamaz ise uçak servis hayatına başlayamaz. Sivil havacılık sektöründe DO-178C rehber dokümanı kullanılsa da, askeri hava araçlarında yazılım uyum gösterim yöntemi için, sistem ve yazılım seviyesinde yapılması gereken aktiviteleri anlatan MIL-HDBK-516C askeri standardı da kullanılabilmektedir.

DO-178C dokümanı, RTCA (Radio Technical Commission for Aeronautics) tarafından yayımlanan ve hava aracında kullanılacak yazılımlar için, yazılım yaşam döngüsü süreçlerini, bu süreçlerin amaçlarını, gerçekleştirilmesi gereken aktiviteleri ve üretilmesi gereken kanıtların beklenen içeriklerini (yazılım planları, gereksinimleri, yazılım tasarımı, kod, test sonuçları gibi) anlatan rehber bir dokümandır [17].

DO-178 rehber dokümanının C sürümü 2011 yılında, B sürümü 1992 yılında, A sürümü 1985 yılında ve ilk sürümü olan 178 sürümü 1980 yılında yayımlanmıştır. Sertifikasyon otoriteleri 1980 yılında beri, yazılım uyum gösterim yöntemi olarak DO-178 rehber dokümanının o dönem geçerli olan sürümünü adreslemektedirler [18].

Sertifikasyon otoriteleri, bir hava aracı yazılım geliştirme sürecinde aşağıdaki fazlarda yürütülen süreç ve üretilen yazılım yaşam döngüsü verileri üzerinden örneklemeler alarak detaylı gözden geçirmeler / denetimler yaparlar.

- Planlama gözden geçirmesi
- Geliştirme gözden geçirmesi
- Doğrulama gözden geçirmesi
- Nihai gözden geçirme

Bu dört önemli aşamanın sonunda yapılan gözden geçirmeler / denetimlerin sonunda, sertifikasyon otoritesi geliştirilen yazılımın havacılık kurallarını sağladığı konusunda ikna olursa yazılım onaylamış olur.

Son yıllarda MTG teknolojisinin havacılık alanında kullanımının artması ile RTCA, DO-178 rehber dokümanının C sürümünü yayımlarken, bu teknolojinin hava aracı yazılımlarında kullanılması durumunda sağlanması gereken amaçları tanımlamak üzere (DO-178C dokümanının eki olarak) DO-331 dokümanını yayımlamıştır [4]. MTG teknoloji kullanılarak geliştirilen yazılımların sertifikasyon otoritesi tarafından onaylanarak hava aracının tip sertifikasının yayımlanabilmesi için, DO-331 dokümanına göre sağlanması gereken tüm amaçların sağlanarak gerekli kanıtların üretilmesi ve sertifikasyon otoritesine sunulması gerekmektedir.

3.3 Yazılım Emniyet Seviyesinin Belirlenmesi

Geliştirilecek bir hava aracının çok sayıda fonksiyonu icra etmesi beklenir. Bu fonksiyonlara örnek olarak, hava aracının konum bilgisinin hesaplanması, hız verisinin gösterimi, kalan yakıtın hesaplanması, uçağın konum bilgisinin hesaplanması, motor göstergelerinin gösterimi fonksiyonları verilebilir.

Yazılım emniyet seviyelerini belirlenmeden önce, tasarlanacak hava aracından beklenen uçak ve sistem fonksiyonları listelenir. Fonksiyonlar ve bu fonksiyonların hata durumları sistematik ve kapsamlı bir şekilde değerlendirilerek fonksiyonel tehlike analizi yapılır. Bu analizin amacı uçak / sistem fonksiyonlarında olabilecek hata durumlarının belirlenmesi ve hata durumlarının hava aracı ve mürettebat

üzerindeki etkilerini sınıflandırmaktır. Bu kapsamda, belirlenen uçak / sistem fonksiyonlarının her biri için ilgili fonksiyonun tamamen kaybı, kısmı kaybı, istemsiz çalışması, hatalı çalışması durumu, hatalı çalışması durumunun tespit edilememesi durumlarını içeren hata senaryolarının uçuş emniyetine etkileri değerlendirilerek emniyet seviyeleri belirlenir.

Örneğin, uçağın motor göstergelerinin pilota gösterilememesi, oto pilot yazılımın yanlış çalışması, hava aracında meydana gelen yangın ile ilgili ikazın pilota verilememesi veya pilot çekmediği halde fırlatma koltuğundaki yazılımın istemsiz bir şekilde aktive olması ile koltuğun uçuş esnasında fırlatılması gibi senaryolarına uçuş emniyetine etkisi değerlendirilerek fonksiyon bazında emniyet seviyesi atanır. Atanabilecek emniyet seviyeleri Çizelge 3.2'de verilmiştir [19].

Çizelge 3.2 Emniyet seviyeleri ve hata durumları arasındaki ilişki [19]

Hata Durumları Sınıfı	**Etkisiz (E)**	**Az önemli (D)**	**Önemli (C)**	**Tehlikeli (B)**	**Ölümcül (A)**
İzin verilen Hata Yapma Olasılığı	Hedef Yok	Olası	Uzak	Son Derece Uzak	Son Derece İhtimal Dışı
Uçağa Etkisi	Emniyet ve operasyonel kabiliyetlere etkisi yok	Fonksiyonel kabiliyetlere ve emniyete çok az etkisi var	Fonksiyonel kabiliyetlere ve emniyete önemli etkisi var	Fonksiyonel kabiliyetlere ve emniyete büyük etkisi var	Uçağın tamamen kaybına yol açar
Yolculara Etkisi	Yolculara fiziksel etkilemeyen rahatsızlık	Sağlığı etkilemeyen, fiziksel rahatsızlık	Hafif yaralanma içeren fiziksel rahatsızlık	Sakatlık veya ölüme neden olan durum	Birden fazla yolcunun ölümüne yol açacak durum
Mürettebata Etkisi	Mürettebata etkisi yok.	İş yükünü çok az arttırır veya acil durum prosedürlerini kullanmayı gerektirebilir.	Fiziksel rahatsızlık verir ve mürettebatın iş yükünü arttırır.	Çok fazla iş yüküne yol açar. Mürettebat istenilen görevi getirememeye başlar.	Mürettebatın ölümüne ve kalıcı sakatlığa yol açar.

Çizelge 3.3'te, hava araçlarında olabilecek hata tanımları ve bunlara ait emniyet seviyelerine ait örnekler verilmiştir. Bunlardan örneğin ilk örnek şu şekilde yorumlanabilir. Hata ağacında hız verisinin hatalı hesaplanması, hava aracı kaybına ve / veya mürettebat ölümlerine yol açabileceği için ölümcül olarak değerlendirilmiş ve bu nedenle hız verisinin hesaplanması fonksiyonuna ait emniyet seviyesi A olarak atanmıştır.

Atanan emniyet seviyesi, operasyon koşullarına veya mimariye bağlı olduğu için bu parametreler değiştiğinde emniyet tehlike değerlendirmesi tekrar yapılmalıdır. En yüksek emniyet seviyesi uçak tipine göre de değişmektedir. Örneğin CS / FAR 23 tipi küçük uçaklar için en yüksek emniyet seviyesi B'den başlar iken CS / FAR 25 tipi büyük uçaklar için seviye A'dan başlar.

Sistem seviyesinde atama yapıldıktan sonra, yazılımın emniyet seviyesi de yazılımın o sistemde icra ettiği fonksiyona göre belirlenir. Örneğin sistem seviyesinde hız verisinin sağlanması fonksiyonu seviye A olarak atanmış olsun. Bu durumda hız verisini sağlayan cihazın emniyet seviyesi A olur. Eğer hız verisini sağlayan fonksiyon cihazın içindeki bir yazılım ise, bu yazılımın seviyesi de A olur. Eğer hız verisi yazılım değil donanım tarafından sağlanıyor ise bu durumda donanımın emniyet seviyesi A olur. Özet olarak yazılım, içinde bulunduğu sistemde icra ettiği fonksiyonun emniyet seviyesi ne ise yazılımın emniyet seviyesi de odur.

Yazılım emniyet seviyesi ile yapılan bu atama, gerçekleştirilecek yazılım doğrulama aktivitelerini etkilemektedir. Örneğin bir yazılım için emniyet seviyesi A (ölümcül) veya B (tehlikeli) olarak belirlendi ise, üretilen tüm verilerin yazarı dışındaki bir kişi tarafından gözden geçirilmesi gerekirken, seviye C ve D için böyle bir şart yoktur. Bir başka örnek olarak yapısal kapsama analizi verilebilir. Seviye A,B ve C için yapısal kapsama analizi yapılması gerekirken, Seviye D için yapılmasına gerek yoktur. Yazılımın seviyesi A'ya yaklaştıkça yapılması gereken doğrulama aktiviteleri de artmaktadır.

Çizelge 3.3 Hata tanımı ve emniyet seviyesi örnekleri

Hata Tanımı	Sınıfı	Emniyet Seviyesi
Hız verisinin hatalı hesaplanması	Ölümcül	A
Seyrüsefer verisinin hatalı gösterimi	Tehlikeli	B
Motor Göstergelerinin kaybı	Ölümcül	A
Konum bilgisinin kaybı	Önemli	C

3.3.1 Uçuş emniyetini olumlu yönde etkileyebilecek öneriler

Yazılım ister geleneksel yazılım geliştirme yaklaşımı ile geliştirilsin isterse de MTG teknolojisi ile geliştirilesin, her durumda “beklenen fonksiyonu” icra etmesi gerekir. Aksi halde ölümlü kaza ile karşılaşılabilir.

Bununla birlikte yazılım beklenen fonksiyonu birebir icra etse bile halen ölümlü kazalarla karşılaşılabilir. Mart 2019 tarihinde gerçekleşen Etiyopya Hava yollarına ait Boeing 737-8MAX kazasında 157 kişi yaşamını yitirmiştir.

Bu kaza ile ilgili olarak yayımlanan pek çok raporda problem sensör ve yazılım kaynaklı gibi görünse de, problemin sistem seviyesinde yapılan tasarım ile de ilgili olabileceği değerlendirilmektedir. İki sensör farklı veriler üretiyor ise pilotların uyarılarak, manevra karakteristiklerini artırma sisteminin devre dış bırakılması belki de bu kazayı önleyebilirdi. Yapılacak detaylı çalışmalardan sonra kazanın sebebinin bu olduğu ortaya çıkarsa, kazanın sebebi olarak yazılımı adreslemek doğru olmaz. Çünkü tasarım sistem seviyesinde yapılmış bir tasarımdır. Bir başka değişle, sensörler farklı veri üretirken pilotların uyarılmaması ve manevra karakteristiklerini artırma sisteminin halen devrede kalması çözümü o uçağın sistem ve emniyet ekibi tarafından geliştirilmiş bir çözümdür. Sistem böyle tasarlandı ise, sensörlerden farklı veri geldiğinde yazılım “kendisinden beklendiği” şekilde uyarı vermez ve manevra karakteristiklerini artırma sistemini de devreden çıkarmaz. Yazılım beklenen fonksiyonu icra ettiği halde hava aracı kaybedilmiş olabilir.

Burada verilmek istenen mesaj, hatalar yazılım kaynaklı olabileceği gibi, sistem tasarımı nedeni ile de olabilir. Emniyet, öncelikle sistem seviyesinde sağlanmalıdır. Emniyetli ve olağan dışı durumları yönetecek bir sistem tasarlanmadı ise, yazılım

beklenen fonksiyonu icra etse bile halen ölümlü kazalar ile karşılaşılabilir. Bu konu ile ilgili sektörde gözlemlenen eksikler ve öneriler aşağıda verilmiştir.

- Sistem seviyesinde geliştirilen veriler (örneğin sistem gereksinimleri) ve işletilen süreçleri sertifikasyon otoritesi tarafından denetimini şart koşan havacılık kurallarının olmaması. Bu kapsamda, havacılık kurallarında açıkça yer almasa bile yazılım ve donanım kapsamında sertifikasyon otoritesi tarafından yapılan çalışmaların sistem seviyesinde de yapılacak şekilde bir mekanizma kurulması önerilmektedir. İlave olarak sistem seviyesi ile ilgili havacılık kurallarının eklenmesi fayda sağlayabilir.

- Hava araçlarında her bileşen (yazılım, donanım) ayrı ayrı doğrulanmaktadır. Bununla birlikte, bu bileşenlerin sistem içinde birlikte çalışmaları durumunda sistem davranışını incelemek son derece önemlidir. Projenin sonlarına doğru yapılması gereken bu aktivitelerin yoğun takvim baskısına maruz kalmadan yapılabilmesi için gerekli kaynaklar projenin başında proje takvimine dahil edilmelidir.

- Sistem gereksinimlerinin ve planlama sürecinin DO-178C rehber dokümanında yer alan Tablo A-1, A-2 ve A-3 amaçları karşılayacak şekilde bir kurgu yapılması önerilmektedir. İlave olarak kalite, konfigürasyon ve değişiklik yönetim süreçlerinin de DO-178C rehber dokümanına uyumlu olması tavsiye edilmektedir.

- Sistem ekibi ile yazılım ekibi arasındaki kopukluk ve “herkes kendi tarafından sorumludur” algısının kırılmasının gerektiği değerlendirilmektedir. Yazılım ekibinin, geliştirilen sistemi olduğu gibi kabul etmeyip sorgulaması, olağandışı senaryoları düşünmesi, varsa tespit ettiği olası açıkları sistem ekibine bildirmesi ve sistem ekibinin de bu geri dönüşleri direnç göstermeden kabul ederek değerlendirmesi çok önemlidir.

- Sistemin, olağan durumda beklene senaryosu ile birlikte olağandışı senaryodaki davranışları da sistem gereksinimi olarak tanımlanmalıdır. Örneğin, iki cihazdan gelen verinin ortalaması alınarak bir değer hesaplanıyor ise, bu cihazlardan birinden veya ikisinden beklenen verinin hiç gelmediği zaman sistemin

davranışı, bir cihazdan [0-100] arasında veri gelmesi bekleniyor ise yazılıma bir şekilde sınır dışı (örneğin -5000) gelmesi durumundaki sistemin davranışı, iki cihaz farklı veri üretiyor ise oto pilot sisteminin devreden çıkarılması, havada cihaz açılıp kapanır ise verilerin varsayılan değere mi döneceği yoksa hafızaya yazılan son değere mi dönmesi gerektiği, sistem gereksinimi olarak tanımlanmalıdır. Sistemin beklediği davranış sistem seviyesinde tanımlanmaz ise, yazılım hatasız da çalışsa bile ölümlü kaza ile karşılaşılabilir.

4 MODEL TABANLI GELİŞTİRME TEKNOLOJİSİ İLE YAZILIM GELİŞTİRME

MTG teknolojisine geçmeden önce geleneksel yazılım geliştirme yaklaşımını kısaca hatırlatmanın faydalı olacağı düşünülmüştür.

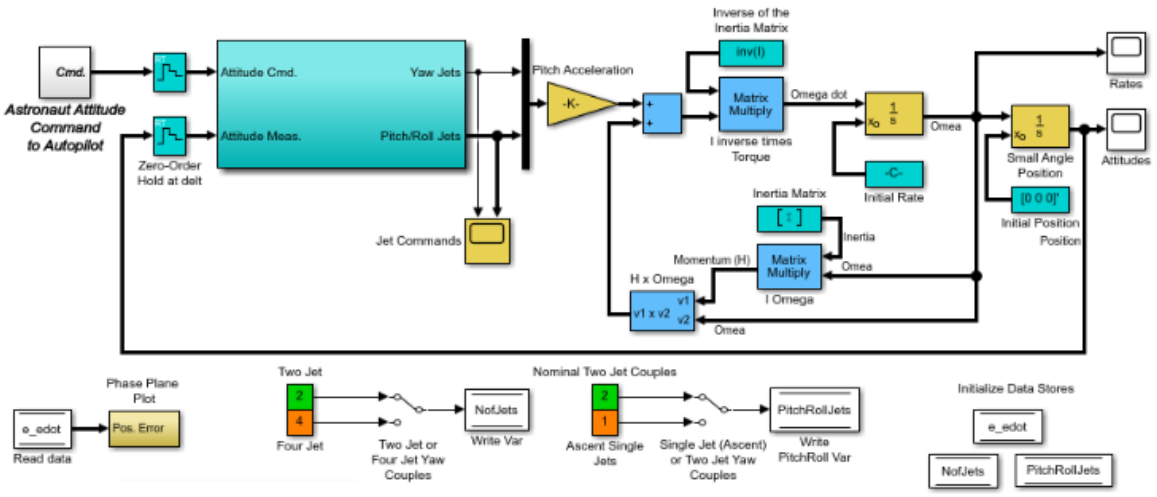
4.1 Geleneksel Yazılım Geliştirme Yaklaşımı

Geleneksel yazılım yaşam geliştirme yaklaşımı ile kast edilen şelale modelidir. Şelale modeli, analiz süreci ile başlayan, yazılım tasarım, kodlama, yazılım test süreçleri ile devam eden ve entegrasyon süreci ile sonlanan ve sıklıkla kullanılan yazılım geliştirme modeldir. Her süreç kendinden önceki sürecin çıktılarını girdi olarak alır ve süreçlerin sonunda ilgili yazılım yaşam döngüsü verisi üretilir.

4.2 Model Tabanlı Geliştirme Teknolojisi Tanımı ve Genel Bilgiler

Günümüzün artan rekabetçi koşullarında üreticiler öne çıkabilmek için daha az maliyetle daha kabiliyetli ürünler üretmek durumunda kalmışlardır. Benzer durum havacılık sektörü için de geçerlidir. Havacılık sektöründeki pek çok hava aracı üreticisinin temel hedefi daha az maliyet ile daha kısa süre içerisinde daha kapasiteli ve kabiliyetli hava araçları geliştirmektir. Daha kapasiteli ve yetenekli hava araçları, daha karmaşık fonksiyonları ve mimarileri de beraberinde getirmektedir. Bu durum, hava araçlarında kullanılan yazılımların artmasına ve bu yazılımların testlerinin ve diğer doğrulama aktivitelerinin karmaşıklaşmasına neden olmaktadır. Kısalan proje takvimleri ile daha az zamanda daha karmaşık sistemler geliştirmek, entegre etmek, doğrulamak ve olası değişiklikler kapsamında etki analizi yapmak zorlaşmaktadır. Bu zorlukları yönetebilmek ve kolaylaştırabilmek üzere firmalar yeni teknoloji arayışlarına gitmişlerdir. Bu teknolojilerden birisi de MTG teknolojisidir.

Model, yazılım geliştirme ve doğrulama aktivitelerini gerçekleştirmek / desteklemek üzere oluşturulan, sistemden yazılıma akacak öğelerin soyut bir gösterimidir. Modeller, yazılım gereksinimleri ve / veya tasarımını temsil ederler. MTG teknolojisi, bu modellerden otomatik olarak yazılım kodunun geliştirilebileceği ve doğrulanabileceği bir teknolojidir. Bu teknoloji ile geliştirilen yazılım, ilgili havacılık kurallarını (DO-331 dokümanına uyum) sağlar ise hava araçlarında kullanılarak sertifiye edilebilir.



Şekil 4.1 Örnek bir oto pilot tasarım modeli [23]

MTG teknolojisi, geleneksel yazılım geliştirme yaklaşımı beraber de kullanılabilir. Geliştirilecek yazılım bazı fonksiyonları MTG teknolojisi ile geliştirilirken, bazıları geleneksel yazılım geliştirme yaklaşımı ile geliştirilerek entegre edilebilirler.

MTG teknolojisi Airbus A380 uçağında, uçuş kontrol sistemleri, oto pilot, uçuş ikaz sistemleri, yakıt yönetimi, iniş takımı, buzdan korunma sistemi, elektriksel yük analizi sistemlerinde kullanılmıştır [20].

Şekilde 4.1'de, Simulink aracında çizilmiş örnek bir oto pilot tasarımına ait model görülmektedir [23].

4.3 MTG Teknolojisindeki Genel Kavramlar ve Aktiviteler

4.3.1 Türetilmiş Model Elemanı

Türetilmiş (derived) model elemanı, üst seviye sistem gereksinimlerine eklenmeyecek kadar detay olan gereksinimler için modele eklenen elemanlardır. Bir başka deyişle, türetilmiş model elemanlarına ait üst seviye bir gereksinim bulunmaz. Türetilmiş model elemanlarına örnek olarak, kesme işleme (interrupt handling) esnasında yazılım mühendisi tarafından tanımlanan sistem davranışı verilebilir. Örneğin bir aritmetik işlem yapılırken oluşabilecek sıfıra bölme hatasında yazılımın davranışının ne olacağı müşteri sözleşmesi veya sistem seviyesi gereksinimlerde tanımlanmaz. Yazılımın frekansı, zamanlaması veya

örneğin bir sıralama yapılacak ise kullanılması gereken sıralama algoritmasının ne olacağı türetilmiş model elemanına verilebilecek örneklerden bazılarıdır. Türetilmiş model elemanları, tasarım ile ilgili çok detay seviyede kaldıkları için ne sözleşmeye ne de sistem gereksinimlerine eklenirler. Havacılık sektöründe türetilmiş gereksinim / model elemanlarının son derece dikkatli kullanılması gerekir. Türetilmiş gereksinimler / model elemanları uçuş emniyetini son derece olumsuz etkileyebilir. Çünkü bu tip durumlarda sistem davranışı, mimariyi ve hata senaryolarını bilen sistem ve emniyet ekibi tarafından değil, yazılım mühendisi tarafından tanımlanmaktadır. Yazılım mühendisi tanımladığı davranışın uçuş emniyetine etkisini bilemeyebilir. Türetilmiş gereksinimlerin / model elemanlarının uçuş emniyetine etkisini, ancak hava aracı mimarisini ve hata senaryolarını değerlendiren sistem ve emniyet ekibi bilebilir. Örneği cihazdan gelen bir verinin (örneğin hız bilgisi) cihaz bozulduğu için yazılıma gelmemesi durumda, sisteminin davranışı yazılım mühendisi tarafından verinin değerini pilota “sıfır” göstermek şeklinde tanımlanmış (kodlanmış) olsun. Böyle bir durumda, sistem ve emniyet mühendisi bu hatanın hava aracı mimarisinde yol açabileceği durumları bildiği için, bu hata durumunda yazılımın kapatılarak pilota siyah ekran göstermenin daha emniyetli olabileceğini değerlendirebilirler. Bir başka örnek olarak, seçilen sıralama algoritmanın performansı hava aracı yazılımının zamanlama performansı ile uyum sağlamayabileceği konusu verilebilir. Seçilen algoritmaya bağlı olarak, yazılım istenen zamanda kendinden beklenen fonksiyonu icra edemeyebilir ve uçuş emniyeti olumsuz yönde etkilenebilir. Bu nedenle, tüm türetilmiş gereksinimlerin / model elemanlarının etiketlenerek uçuş emniyetine etkisinin değerlendirilebilmesi için ilgili sistem ve emniyet mühendislerine iletilmesi gerekir.

MTG teknolojisi kullanımında tüm türetilmiş model elemanlarının modeli geliştiren kişiler tarafından mutlaka etiketlenmesi ve emniyet bölümüne iletilmesi gerekir. Bir modelde türetilmiş bir model elemanının gösterimine ait bir örnek, Ek 1’de yer alan Şekil 8.1’de verilmiştir.

Türetilmiş model elemanlarının emniyet değerlendirme sürecinde, emniyet mühendisi, sistem mühendisi ve yazılım mühendisi bir araya gelerek bir değerlendirme yaparlar. Bu değerlendirmede, yazılım mühendisi tarafından tanımlanan sistem davranışının uçuş emniyetine etkisi değerlendirilir. Bu

değerlendirme sonucunda türetilmiş model elemanının uçuş emniyetine negatif etkisinin olabileceği değerlendirilirse (yukarıdaki örnekte, hız verisinin sıfır gösterilmesi pilotun dikkatini çekmeyebilir ve pilot cihaz bozulduğu halde uçuşa devam edebilir) model elemanı mimariden çıkarılır ve başka bir çözüme gidilir. Türetilmiş model elemanının emniyet değerlendirmesi sürecinde aşağıdaki sorular kullanılabilir:

- Türetilmiş model elemanı ile tanımlanan fonksiyon sistem davranışını nasıl etkiler? Oluşabilecek emniyetsiz durumlar var mıdır?
- Tanımlanan bu fonksiyon yanlış çalışırsa uçuş emniyetine etkisi ne olur?
- Tanımlanan bu fonksiyon çalışması gerektiği halde çalışmazsa uçuş emniyetine etkisi ne olur?
- Tanımlanan bu fonksiyon istemsiz bir anda çalışırsa (örneğin acil bir durum olmadığı ve pilot kolu çekmediği halde fırlatma koltuğu yazılımın istemsiz bir şekilde sistemi aktive ederek koltuğu fırlatılması) uçuş emniyetine etkisi ne olur?

4.3.2 Fonksiyonel olmayan model elemanları

Modelde fonksiyonel olmayan, bir başka deyişle koda dönüşmeyecek (örneğin açıklama blokları) model elemanları olabilir. Bu elemanların test edilmesi, kapsama analizinde kapsanması ve koda dönüşmesi beklenmez. O nedenle bu tip model elemanlarının geliştirme fazında model standardında belirtilen yönteme göre etiketlenmesi gerekir. Etiketlenmez ise, model kapsama aktivitesinde veya kod ile model arasındaki izlenebilirlikte gereksiz yere sorun şeklinde görülür ve analiz için vakit kaybına neden olabilir.

4.3.3 Standartlara uyum

MTG kullanımında geliştirilen modelin, model standardına uyumu çok önemlidir. Model standardı, MTG teknolojisinde kullanılan araca, seçilen modelleme diline ve projeye göre değişebilen, DO-331 rehber dokümanında çizilen çerçeveyi kapsayacak şekilde proje ekibi tarafından geliştirilen kurallar bütünüdür. Tüm modellerin belirlenen standarda bire bir uyumlu olması, tüm geliştirme ekibinin

aynı dili kullanarak aynı geliştirme yaklaşımını belirlemesi ve tüm ekibin bir modelde aynı davranışı anlaması açısından son derece önemlidir. Projede tanımlama modeli ve tasarım modeli kullanılıyorsa her iki model için ayrı ayrı standartlar geliştirilmelidir. Geliştirilecek tanımlama modellerinin ve tasarım modellerinin model standartlarına göre hazırlanması ve gözden geçirilmesi gerekir. Model standardı ile ilgili detaylar Bölüm 7'de verilmiştir.

4.3.4 Araç kalifikasyonu

MTG teknolojisinin sağladığı en önemli avantajlardan birisi modelden otomatik kod üretilebilmesidir. Daha önce de belirtildiği gibi, havacılık sektöründe MTG teknolojisinin uygulamada kullanımı, genellikle tasarım modelinden otomatik olarak kod üretilmesi şeklinde olmaktadır. Bu şekilde üretilen kodun hava aracında kullanılabilmesi için otomatik kod üreten aracın kalifiye edilmesi gerekir. Aksi halde, üretilen otomatik kod hava aracında kullanılamaz.

Araç kalifikasyonu, bir yazılım aktivitesinin insan yerine bir araç tarafından yapılması durumunda gereklidir. Bir başka deyişle, bir insan tarafından yapılan bir yazılım aktivitesi, araç tarafından yapıldığı için insan tarafından tekrar yapılmıyor ise veya araç aktiviteyi otomatikleştiriyor ise ve aracın ürettiği çıktı gözden geçirilmiyor ise bu aracın kalifiye edilmesi gereklidir. Örneğin bir yazılım testinin geçti / kaldı kararını, beklenen değer ile hesaplanan değeri karşılaştırarak bir araç veriyor ise, bu aracın kalifiye edilmesi gerekir. Aksi halde araca güvenilerek sonraki sürece geçilemez. Eğer aracın ürettiği çıktı insan tarafından gözden geçiriliyor ise (örnekte geçti / kaldı kararını araç verse bile her test adımı için karşılaştırma tekrar bir insan tarafından yapılıyor ise) araç kalifikasyonuna gerek yoktur. Aracın kalifikasyon süreci kısaca şu şekilde açıklanabilir. İhtiyaç duyulan araç için, araçtan beklenen fonksiyonlar gereksinim olarak tanımlanır (bu örnek için yazılımın ürettiği sonuç ile beklenen sonucu karşılaştır. Sonuçlar aynı ise "geçti", değil ise "kaldı" mesajını ekranda göster gibi). Bu gereksinimlere göre araç geliştirilir. Daha sonra bu gereksinimlere göre hazırlanan test prosedürlerine göre geliştirilen araç doğrulanır (araç, beklenen değer ile yazılımın ürettiği veri aynı ise "geçti", farklı ise "kaldı" mesajını ekranda gösterebiliyor mu kapsamında yapılan testler).

Havacılık sektöründe kullanılan ve kalifikasyona tabi olabilecek iki tip araç vardır. Bunlar geliştirme araçları ve doğrulama araçlarıdır. Yukarıda verilen örnekteki araç, yazılım doğrulama süreci olan testlerde kullanıldığı için bir yazılım doğrulama aracıdır. Yazılım geliştirme araçlarına örnek olarak ise, otomatik kod üreten araçlar verilebilir.

Verilen örnekten de görülebileceği gibi yazılım geliştirme araçları koda hatalı bir kod parçası ekleyebilirken, yazılım doğrulama araçları kodun içinde var olan bir hatanın bulunmasını önleyebilir. Bu nedenle yazılım geliştirme araçlarının kalifikasyonu, yazılım doğrulama araçlarının kalifikasyonunda daha zordur.

Havacılık sektöründe firmalar genellikle kendi doğrulama araçlarını (yazılım testlerini icra ettikleri araçlar) geliştirerek kalifiye ederken, kendi yazılım geliştirme araçlarını (otomatik kod üreten araçlar) geliştirerek kalifiye etme yoluna pek gitmemektedirler. Geliştirme araçlarını kalifiye etmek zor olduğu için, genellikle kalifiye edilmiş hazır geliştirme araçları (örneğin SCADE) kullanılmaktadır.

İster hazır olsun, ister yeni geliştirilsin kalifiye bir araç ile yapılan yazılım aktivitesinin (bu örnekte geçti / kaldı kararının verilmesi) tekrar yapılmasına veya verilen kararın gözden geçirilmesine ihtiyaç yoktur. Benzer şekilde kalifiye bir araçtan otomatik üretilen kodun da gözden geçirilme ihtiyacı bulunmamaktadır.

MTG teknolojisinde geliştirilen modeller gözden geçirildikten, model ile üst seviye gereksinim arasında izlenebilirlikler kurulduktan sonra, kalifiye bir araç ile otomatik olarak üretilen kod hava araçlarında kullanılarak sertifiye edilebilir. Bu sayede hem el ile kod geliştirmesine, hem de kodun model ve kodlama standardı ile uyumluluğunun gözden geçirilmesine gerek kalmamış olur.

Kullanılan otomatik kod geliştirme aracı kalifiye değil ise, bu kodun hava aracında kullanılabilmesi için tasarım modeline göre gözden geçirmesi gerekir. Böyle bir durumda, araç tarafından otomatik üretilen kodun insan tarafından anlaşılabilirliği de önemlidir. Üretilen kodu anlamak zor ise, modele uyumluluğunu gözden geçirmek de son derece zor olacaktır.

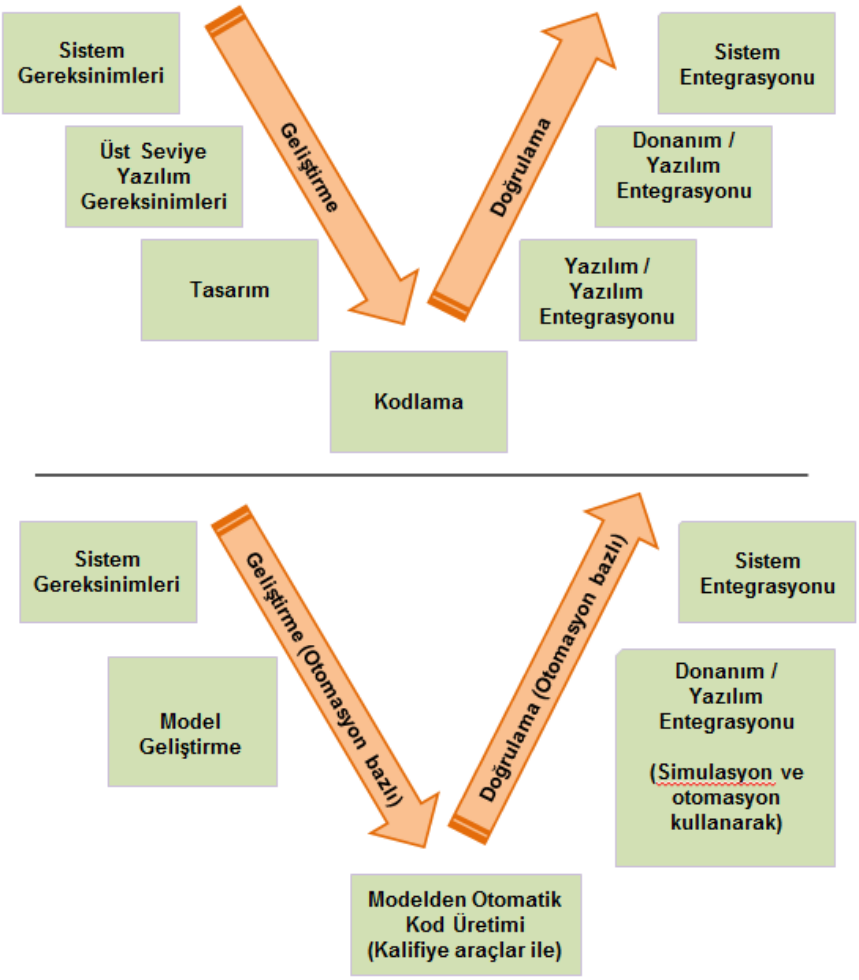
4.3.5 Bağımsızlık

Bir hava aracı sisteminde kullanılacak yazılımın emniyet seviyesinin, yazılımın sistemde icra ettiği fonksiyonlara göre belirlendiği bölüm 4.3'te belirtilmişti. Hava aracı yazılımlarda, emniyet çalışmalarına göre yazılım seviyesi A veya B olarak belirlenmiş ise gözden geçirme aktivitesinin bağımsız yapılması gerekir. Burada bahsedilen bağımsızlık, veriyi (model, gereksinim, simülasyon prosedürü gibi) üreten kişi ile gözden geçiren kişinin eş seviyede farklı kişiler olması zorunluluğudur. Seviye C ve D yazılımlar için veriyi geliştiren kişi ile gözden geçiren kişi olabilirken, seviye A ve B yazılımlar için bu durum havacılık kurallarına aykırıdır.

4.4 Geleneksel Yaklaşım ile MTG Teknolojisinin Karşılaştırması

Geleneksel yazılım geliştirme yaklaşımında, sistem gereksinimleri detaylı olarak hazırlanır ve bu gereksinimlerden, üst seviye yazılım gereksinimleri, üst seviye yazılım gereksinimlerinden yazılım tasarımı (alt seviye yazılım gereksinimi ve yazılımı mimarisi), yazılım tasarımında da el ile kod geliştirilir. Geliştirilen yazılım, yazılım gereksinimlerinden oluşturulan test prosedürlerine göre doğrulanır. Yazılım doğrulandıktan sonra, yazılımın içinde bulunduğu sistem doğrulama ve geçerleme çalışmalarına başlanır. Geleneksel yazılım yaşam döngüsünde süreç, her aşama için ayrı ayrı geliştirilen yaşam döngüsü verileri ve bunların el ile gözden geçirilerek ilerlemesi şeklinde olup sistem doğrulama ve geçerleme faaliyetleri ancak projenin sonuna doğru gerçekleştirilebilmektedir.

MTG teknolojisinde ise, üst seviye sistem gereksinimler tanımlandıktan sonra sistem, bu gereksinimlere göre modelleme araçları kullanılarak modellenmektedir. Bu modellerden, kalifiye araçlar ile otomatik kod üretilebilmekte ve yine modelden otomatik üretilen model simülasyon ve prosedürleri ile yazılım otomatik olarak doğrulanabilmektedir. Bir başka deyişle, geleneksel yazılım geliştirme yaklaşımında üretilen, yazılım üst seviye gereksinimleri, yazılım tasarımı (yazılım alt seviye gereksinimleri ve yazılım mimarisi), yazılım test prosedürleri ve kaynak kod üretilmeden, MTG teknolojisinde sistem modellerinden direkt kaynak kod üretilebilmektedir.



Şekil 4.2 Geleneksel yaklaşım (üstte) ile MTG teknolojisi (altta) karşılaştırması [21]

Şekil 4.2'de görülebileceği gibi geleneksel yazılım geliştirme teknolojisinde her aşamada ayrı ayrı yaşam döngüsü verileri üretilerek aşama aşama doğrulama yapılmaktadır. MTG teknolojisinde ise, model merkezli bir yaşam döngüsü ile ilerlenerek çok daha az veri üretilerek yazılım geliştirme süreci tamamlanabilmektedir.

Geliştirilen model, kod dahi üretilmeden, simülasyon araçları ile sürekli test edilerek erken fazda doğrulanabilmektedir [21]. Böylece sistem doğrulaması ve performans ölçümü geleneksel geliştirme yaklaşımında olduğu gibi geliştirme fazının sonunda değil başında yapılabilmekte, sistemdeki olası hatalar ve performans iyileştirme ihtiyaçları projenin başında görülebilmektedir.

4.5 MTG Teknolojisinin Avantajları

Günümüz dünyasının kronikleşmiş sıkışık takvim ve düşük maliyet hedefi düşünüldüğünde, MTG teknolojisinin sağladığı en önemli avantajların başında, sistemin erken doğrulanabilmesi ile hataların erken tespit edilebilmesi gelmektedir.

Bilindiği gibi yazılım hataların çözülme maliyeti, hatanın tespit edildiği aşamaya doğrudan bağlıdır. Şekil 4.3'te görülebileceği gibi hata ne kadar geç tespit edilirse, hatanın çözülmesi için harcanacak maliyet büyük oranda artmakta, hatta hatanın çözümü imkânsız hale gelebilmektedir [22].

MTG teknolojisi ile otomatik üretilen kod, kalifiye bir araç ile üretildi ise doğrulama aşamasından sonra hava araçlarında kullanılabilir ve sertifiye edilebilir. İlave olarak, geleneksel yazılım geliştirme yaklaşımında yazılım gereksinimlerine göre hazırlanan test prosedürlerini de hazırlamadan, modelden otomatik üretilen simülasyon durum ve prosedürleri ile model henüz kod bile yazılmadan kolaylıkla doğrulanabilmektedir.

Geleneksel yaşam döngüsünde, yazılım üst gereksinimler, yazılım alt seviye gereksinimler, yazılım mimarisi ve yazılım test prosedürleri hazırlanmak zorunda iken, MTG teknolojisinde bu verilerin hazırlanmasına gerek olmaması üreticilere büyük bir takvim ve maliyet avantajı sağlamaktadır.

Şekil 4.3 Hatanın çözülme maliyetinin tespit edildiği sürece göre değişimi [22]

MTG teknolojisinde geleneksel yazılım geliştirme yaklaşımından farklı olarak yazılı gereksinimler yerine gereksinimleri anlatan görsel modeller hazırlanmaktadır. Modellerin görsel olması, karmaşık sistemlerin daha kolay anlaşılmasını sağlamakta ve detaylar arasında boğulmadan büyük resmin daha kolay görülebilmesine imkân vermektedir. Karmaşık sistemlerin tasarımı, emniyet değerlendirmeleri, hata tespit etme ve ayıklama, regresyon analizi gibi çalışmalar için sağlanan bu görsellik geliştiricilere önemli bir avantaj sağlamaktadır.

En önemlileri yukarıda detaylandırılan MTG teknolojisinin avantajları aşağıda özetlenmiştir.

- Karmaşık sistemlerin yönetimi (modelleme, karmaşık sistemlerin kontrolünü ve anlaşılabilirliğini kolaylaştırır)

- Erken fazda doğrulama (model simülasyonu sistem doğrulama, henüz kod bile üretilmeden çok daha erken fazlarda gerçekleştirilebilir)

- Otomatik kod üretimi (modelden otomatik kod ve simülasyon test durum ve prosedürlerinin üretilmesine imkân verir)

- Grafiksel dillerin ve araçların kullanımı (grafiksel gösterimlerin ve modelleme araçlarının kullanımına olanak sağlar)

- Yeniden kullanılabilirlik (geliştirilen modeller, aynı fonksiyonu içeren diğer yazılımlar içinde kullanılabilir)

- Ortak dil ve daha az varsayım (sistem mühendisleri ve yazılım geliştirme ekibi arasında ortak bir dil kullanımına imkân verir. İki farklı alan için ortak bir dil kullanımı, son derece tehlikeli sonuçlara yol açabilecek “varsayımları” azaltır)

- İzlenebilirlik (modelleme araçlarının da desteği ile gereksinimler ve modeller arasında izlenebilirlik kolayca kurulabilir)

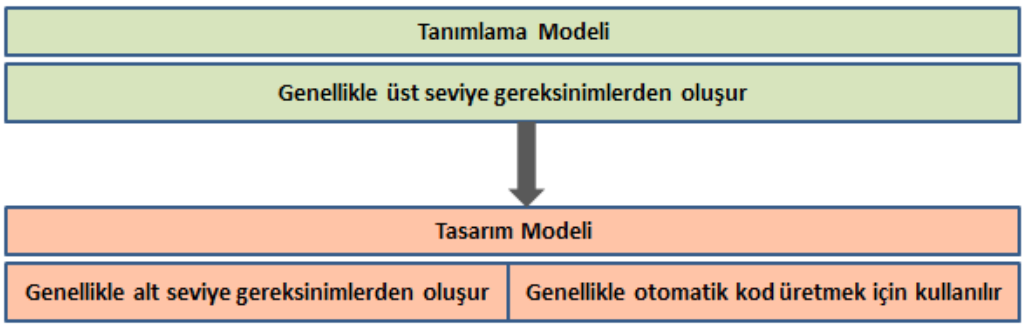
- Daha az veri üretimi (yazılım üst seviye gereksinim, yazılım alt seviye gereksinim, tasarım üretilmeyebilir)

MTG teknolojisinin telekomünikasyon sektörü gibi emniyet kritik yazılım içermeyen sektörlerde kullanımı daha eski olsa da, hava aracı yazılımlarında kullanımı daha yeni olduğu belirtilmişti. Bunun aşağıdaki unsurlardan kaynaklanabileceği değerlendirilmektedir:

- Modelleme araçları ile üretilen otomatik kodun hava araçlarında kullanılabilmesi için gerekli olan kalifikasyon ihtiyacı
- DO-331 rehber dokümanının kullanıcılar tarafından tam anlaşılamaması
- Modelleme araçlarının lisanslarının çok pahalı olması
- Modelleme araçlarının öğrenme ve kullanımının zorluğu
- Geleneksel yazılım geliştirme kültürüne daha yatkın olunması
- MTG teknolojisinin hava araçlarında daha önce kullanılmamış olmasının verdiği çekince
- MTG teknolojisinin ve özellikle modelden otomatik üretilecek kodun hava araçlarında sertifiye edilebileceğine dair duyulan endişe

4.6 Tanımlama Modeli ve Tasarım Modeli

RTCA tarafından MTG teknolojisi ile ilgili sağlanması gereken amaçların anlatıldığı DO-331 rehber dokümanında iki temel yaşam döngüsü verisinden bahsedilmektedir. Bunlar Şekil 4.4'te görüldüğü gibi, tanımlama modeli ve tasarım modelidir.

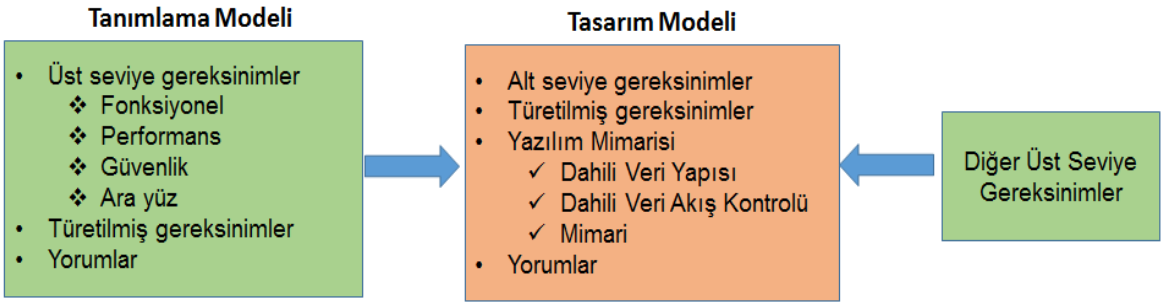


Şekil 4.4 Tanımlama modeli ve tasarım modeli ilişkisi

Tanımlama modeli, yazılım bileşenlerinin fonksiyonel, performans, ara yüz ve emniyet karakteristiklerini içeren üst seviye gereksinimlere ait soyut bir gösterimdir. Yazılım fonksiyonlarının net bir şekilde tanımlanabilmesi için tanımlama modelinin bu karakteristikleri, seçilen modelleme aracı ve belirlenen modelleme gösterim kurallarına uygun olarak net bir şekilde tanımlıyor olması gerekir. Şekil 4.5'te görüldüğü gibi tanımlama modeli, tasarım modelini oluşturacak üst seviye gereksinimleri içermeli, veri yapıları, veri ve kontrol akışı gibi tasarım ile ilgili detayları içermemelidir.

Tasarım modeli Şekil 4.5'te görüldüğü gibi, yazılım bileşenleri, veri yapıları, veri ve kontrol akışı, mimari gibi tasarıma ait bilgileri içerir. Bir başka deyişle, tasarım modelleri tanımlama modelindeki gereksinimleri gerçekleyen alt seviye gereksinimleri içerir. Tasarım modelinin bu bilgileri, seçilen modelleme aracı ve belirlenen modelleme gösterim kurallarına uygun olarak net bir şekilde tanımlıyor olması gerekir. MTG teknolojisinde hazırlanan tasarım modellerinden kalifiye araçlar kullanılarak otomatik kod ve test prosedürleri üretilmektedir.

Tanımlarından ve beklenen içeriklerinden de anlaşılacağı üzere, geliştirilen bir model aynı anda hem tanımlama modeli hem de tasarım modeli olamaz. MTG teknolojisi ile yazılım geliştirilse bile her durumda sistem / yazılımın davranışının ve performansının tanımlandığı mutlaka yazılı gereksinimlerin olması gerekmektedir. Model, bu gereksinimlere göre geliştirilir ve doğrulanır.



Şekil 4.5 Tanımlama modeli ve tasarım modeli içerikleri

4.7 MTG Teknolojisinde Kullanılabilecek Yaşam Döngüsü Tipleri

Endüstrideki uygulamalara bakılarak MTG teknolojisinin genellikle 5 farklı tipte uygulanabileceğini görülmektedir [11]. Seçilecek MTG uygulama tipine göre tanımlama modeli ve tasarım modelinden yalnız biri veya her ikisi birden geliştirilebilir.

MTG teknolojisi kullanıcıları, geliştirecekleri yazılım kapsamında Çizelge 4.1'de belirtilen MTG yaşam döngüsü tiplerden hangisini seçeceğini planlama aşamasında belirlemeli ve yazılım geliştirme planında net bir şekilde ifade etmelidir. Seçilen tipe göre, üretilen verinin uyması gereken kriterler ve yapılması gereken aktiviteler değişmektedir.

Yazılım yaşam döngüsü boyunca tanımlama modeli mi, tasarım modeli mi yoksa her ikisinin birden mi üretileceği açık bir şekilde yazılım geliştirme planında yazmalıdır. Projede geliştirilecek modeller, tasarım modeli veya tanımlama modeli şeklinde geçmiyor ise, bunlara karşılık hangi verilerin üretileceği net bir şekilde yazılım geliştirme planından adreslenmelidir.

MTG teknolojisi, sistem ve yazılım yaşam döngüsü süreç ve verilerini ilgilendirdiği için, sistem seviyesinde üretilecek verilerin ve gerçekleştirilecek aktivitelerin de sistem seviyesinde bir planda (örneğin sistem geliştirme planı, sistem mühendisliği planı gibi) belirtilmesi gerekir.

MTG teknolojisinde kullanılabilecek yaşam döngüsü tipi, yazılımın emniyet seviyesi, şirketin sahip olduğu araçların kalifikasyon durumu, yazılımın tabi olduğu havacılık kuralları, geliştirilecek yazılımın karakteristiği, karmaşıklığı, yazılım ve sistem ekibinin ara yüzü ve iletişimi, sistem ve yazılım ekibinin modelleme araçlarını kullanabilme becerisi gibi ölçütlere göre seçilebilir.

Seçilebilecek yaşam döngüsü tipleri Çizelge 4.1'de verilmiş olup, her bir tipte yapılması gereken aktiviteler aşağıda verilmiştir.

Çizelge 4.1 Model yaşam döngüsü tipleri

<table>
<tr><th>Yaşam Döngüsü Bilgisi Üreten Süreç</th><th>Tip-1</th><th>Tip-2</th><th>Tip-3</th><th>Tip-4
(Bkz: Not 1)</th><th>Tip-5
(Bkz: Not 1)</th></tr>
<tr><td>Sistem Gereksinimleri ve Sistem Tasarım Süreçleri</td><td>Yazılıma atanan gereksinimler</td><td>Modelin geliştirildiği gereksinimler</td><td>Modelin geliştirildiği gereksinimler</td><td>Modelin geliştirildiği gereksinimler</td><td>Modelin geliştirildiği gereksinimler</td></tr>
<tr><td rowspan="2">Yazılım Gereksinimleri ve Yazılım Tasarım Süreçleri</td><td>Modelin geliştirildiği gereksinimler</td><td>Tanımlama modeli
(Bkz: Not 2)</td><td>Tanımlama modeli</td><td rowspan="2">Tasarım modeli</td><td rowspan="2">Tasarım modeli</td></tr>
<tr><td>Tasarım modeli</td><td>Tasarım modeli</td><td>Yazılı yazılım tasarımı
(Bkz: Not 3)</td></tr>
<tr><td>Yazılım Kodlama Süreci</td><td>Kaynak kod</td><td>Kaynak kod</td><td>Kaynak kod</td><td>Kaynak kod</td><td>Kaynak kod</td></tr>
</table>

Not 1: Tip-4 ve Tip-5'te sistem ve yazılım yaşam döngülerinin sınırlarını çizmek ve birbirinden ayırmak kolay değildir. Bununla birlikte, modelin hangi seviyede (sistem veya yazılım) geliştirildiğine bakılmaksızın geliştirilen modeller ile ilgili DO-331 dokümanındaki tüm şartlar sağlanmalıdır. Bu kapsamda, tasarım modelleri için alt seviye gereksinimler ile ilgili DO-178C amaçları, modelin geliştirildiği gereksinimler için de DO-178C'de yer alan üst seviye gereksinimler ile ilgili amaçlar sağlanmalıdır.
Not 2: Tasarım modeli, tanımlama modelinde yer alan gereksinimlerden üretilmiştir.
Not 3: Yazılı gereksinimler, alt seviye yazılım gereksinimlerine ve mimariye karşılık gelmektedir. Bu tipte, geleneksel yazılım geliştirme yaklaşımında olduğu gibi, yazılım tasarım dokümanı hazırlanır.

Geleneksel yazılım geliştirme yaklaşımı ile geliştirilen hava aracı yazılımları için, sistem gereksinimlerinin doğruluğu, tamlığı, yeterliliği gibi konular yazılım seviyesinde olmadıkları için, yazılım ile ilgili havacılık kurallarına (DO-178C rehber dokümanına uyumlu yazılım geliştirme) ve sertifikasyon otoritesi tarafından yapılan sertifikasyon gözden geçirmelerine / denetimlerine tabi değildir. MTG teknolojisi ile geliştirilen hava aracı yazılımlarında ise, sistem ve yazılım verileri ve yaşam döngüleri iç içe geçtiği için sistem gereksinimlerinin de yazılım ile ilgili havacılık kuralı olan DO-178C'ye uymaları gerekebilir (Tip-4 ve Tip-5 gibi). Bu durumda sistem seviyesinde üretilen veriler de sertifikasyon otoriteleri tarafından yapılan gözden geçirmelere / denetimlere tabi olur.

Tip-4 veya Tip-5 yaşam döngüsünde, üreticiler için en büyük sorunun, daha önce DO-178C rehber dokümanı ile çalışmamış olan sistem ekibinin bu rehber dokümana uyumlu gereksinim ve tasarım modeli geliştirme zorluğu olacağı

değerlendirilmektedir. DO-178C rehber dokümanı emniyet kritik hava aracı yazılımları için geliştirildiğinden, yazılım yaşam döngüsü boyunca sağlanması gereken amaçlar, yapılması gereken doğrulama aktiviteleri, analizler ve hazırlanacak veriler son derece detaylı ve zordur. Uzun süredir DO-178C ile çalışan tecrübeli yazılım mühendislerin geliştirdiği yazılım yaşam döngüsü verilerinde bile sertifikasyon gözden geçirmelerinde / denetimlerinde sorunlar yaşanabilmektedir. Bu sebeple daha önce bu rehber doküman ile çalışmamış olan mühendislerin geliştirdiği gereksinimler ve modellerde sağlanamayan DO-178C ve DO-331 amaçları nedeni ile sertifikasyon sürecinde sorunlar yaşanabilir.

MTG teknolojisi ile yazılım geliştirilecek ise, sertifikasyon sürecinin sorunsuz ilerleyebilmesi için projenin başından itibaren DO-178C tecrübesi olan yazılım ekibi ile sistem ekibinin en azından her bir yazılım yaşam sürecinin başında bir süre birlikte çalışması önerilmektedir. Çalışma boyunca, daha önce DO-178C ile uyumlu geliştirilmiş ve doğrulanmış yazılım verileri üzerinden sistem ekibi ile geçmenin faydalı olacağı değerlendirilmektedir.

Aşağıda, Çizelge 4.1'te yer alan MTG yaşam döngüsü tipleri ile ilgili açıklamalar verilmiştir.

4.7.1 Yaşam döngüsüne tipine göre yapılması gereken aktiviteler

Tip-1'de geleneksel yazılım geliştirme yaklaşımıda olduğu gibi sistem gereksinimlerinden yazılım ile icra edilecek olanları, yazılıma atanmaktadır. Yazılıma atanan bu gereksinimler, yine geleneksel yöntemdeki üst seviye gereksinimlere denk gelmektedir. Bu aşamdan sonra geleneksel yöntemde yazılım tasarımı oluşturulurken, MTG teknolojisinde tasarım modeli geliştirilmekte ve bu modellerden de (genellikle otomatik kod üretme aracı ile) kaynak kod üretilmektedir. Tip-1'de geliştirilen veriler için yapılması gereken aktiviteler ve uyulması gereken havacılık kuralı aşağıda verilmiştir.

Tip-1'de yazılıma atanan sistem gereksinimleri ARP 4754A'ye göre doğrulanmalı ve geçerlenmelidir. ARP 4754A, hava aracı ve hava aracı sistemlerinin geliştirilmesini sürecinde uyulması gereken kuralları anlatan bir standarttır [24]. Tip-1 seçildi ise, sistem doğrulama ve geçerleme planında MTG teknolojisinin

kullanılacağı, yaşam döngüsü olarak Tip-1 seçildiği ve sistem gereksinimlerinin ARP 4754A standartına göre doğrulanacağı belirtilmelidir. Geçerleme sürecinde, gözden geçirme, analiz, simülasyon veya test yöntemi kullanılabilir.

Tip-1'de geliştirilen modele ait gereksinimler, üst seviye yazılım gereksinimleri olarak ele alınarak DO-178C'de yer alan üst seviye yazılım gereksinimleri ile ilgili amaçlar (DO-178C, Tablo A-3) sağlanmalıdır.

Tip-1'de tasarım modeli üst seviye gereksinimlere göre tasarım modeli geliştirilir. Geliştirilen modeller, üst seviye gereksinimlere uyumluluklarını doğrulamak üzere gözden geçirilmelidir. Tasarım modelleri, DO-178C, Tablo A-4'te yer alan amaçları karşılamalıdır. Bu amaçların bazıları (1,2,4,7,8,9,11) model simülasyonları ile karşılanabilir. Diğer amaçlar için ilave doğrulama aktiviteleri yapılmalıdır. İlave olarak, tasarım modeli üretildiği için MTG teknolojisi kullanımına özel yapılması gereken model kapsama analizi yapılmalıdır. (Model kapsama analizi için Bkz. Bölüm 4.11.3.1)

Tip-1'de geliştirilen çalıştırılabilir nesne kodunun, tasarım modeli ile uyumlu olduğu doğrlanmalıdır. Kaynak kod ister el ile geliştirilsin ister otomatik üretilsin, üst seviye gereksinimlere göre hazırlanmış yazılım tesrleri ile her durumda gereksinim tabanlı, normal aralık testi ve gürbüzlük (robustness) testilerini içeren yazılım testlerinin yapılması gerekmektedir. Yazılım testlerinin ardından, geleneksel yazılım geliştirme yaklaşımında olduğu gibi yazılım / yazılım ve yazılım / donanım entegrasyon testlerinin hedef donanım üzerinde yapılması gerekir.

Son aşama olarak geliştirilen yazılım kodunu içeren sistem seviyesinde testler gerçekleştirilerek sistem gereksinimleri ARP4754A standardına göre doğrulanmalıdır.

Tip-2 veTip-3 yaşam döngülerinde, sistem seviyesinde hazırlanan üst seviye gereksinimlerinden, yazılım seviyesinde tanımlama modeli oluşturulur. Çizelge 4.1'den görülebileceği gibi, klasik yöntemlerde yer alan üst seviye yazılım gereksinimleri burada oluşturulmamakta, sistem seviyesinde oluşturulan gereksinimlerden tanımlama modeli geliştirilmektedir. Tanımlama modelinden de,

Tip-2'de tasarım modeli geliştirilmekte, Tip-3'te geleneksel yöntemde olduğu gibi yazılım tasarım verisi oluşturulmaktadır.

Tip-2 veTip-3'te tanımlama modelin geliştirildiği gereksinimler ARP 4754A'ya göre oluşturulmalıdır. Sistem doğrulama ve geçerleme planında sistem gereksinimlerinin nasıl geçerleneceği yazılmalıdır.

Tip-2 veTip-3'te**,** sistem seviyesinde hazırlanan üst seviye gereksinimlere göre, yazılım seviyesinde tanımlama modeli geliştirilir. Geliştirilen tanımlama modeli, yazılım üst seviye gereksinimlerini içerir ve DO-178C, Tablo A-'te verilen amaçlar karşılanmalıdır.

Eğer tanımlama modelinden Tip-2'de olduğu gibi tasarım modeli üretilecek ise, tasarım modeli ile tanımlama modeli arasındaki uyumluluk gözden geçirilmelidir. Geliştirilecek tasarım modeli, DO-178C, Tablo A-4'te yer alan amaçları karşılamalıdır. Bu amaçların bazıları (1,2,4,7,8,9,11) model simülasyonları ile karşılanabilir. İlave olarak tasarım modeli gözden geçirilmeli ve model kapsama analizi yapılmalıdır.

Eğer tanımlama modelinden Tip-3'te olduğu gibi geleneksel yazılım geliştirme yaklaşımındaki yazılım tasarım verisi geliştirilecek ise üretilecek yazılım tasarım verisi, DO-178C'de yer alan yazılım tasarım amaçlarına (DO-178C, Tablo A-4) göre geliştirilmeli ve doğrulanmalıdır. Bu durumda bir tasarım modeli geliştirilmediği için model doğrulaması ve model kapsama analizi de yapılmaz.

Tip-2 veTip-3'te, geliştirilen çalıştırılabilir nesne kodunun, Tip-2 ise tasarım modeli ile Tip-3 ise yazılım tasarım dokümanı ile uyumlu olduğu doğrlanmalıdır. Kaynak kod ister el ile geliştirilsin ister otomatik üretilsin her durumda gereksinim tabanlı, normal aralık testi ve gürbüzlük testilerini içeren yazılım testlerinin yapılması gerekmektedir. Yazılım testlerinin ardından, geleneksel yazılım geliştirme yaklaşımında olduğu gibi yazılım / yazılım ve yazılım / donanım entegrasyon testlerinin hedef donanım üzerinde yapılması gerekir.

Son aşama olarak geliştirilen yazılım kodunu içeren sistem seviyesinde testler gerçekleştirilerek sistem gereksinimleri ARP4754A standardına göre doğrulanmalıdır.

Tip-4 ve Tip-5, yukarıda da belirtildiği gibi geleneksel yazılım geliştirme sürecinden oldukça farklıdır. Çizelge 4.1'de de görülebileceği gibi, yazılım tasarım bilgisini içeren tasarım modeli geleneksel yazılım geliştirme yaklaşımından farklı olarak sistem seviyesinde (genellikle sistem ekibi tarafından) hazırlanmaktadır. İlave olarak burada, tanımlama modeli veya geleneksel yazılım gelişitrime yaklaşımında hazırlanan üst seviye yazılım gereksinimleri de yoktur. Hazırlanan tasarım modelinden (genellikle otomatik kod geliştirme aracı ile) kaynak kod üretilir.

Tip-4 ve Tip-5'te modelin geliştirildiği üst seviye gereksinimler, sistem seviyesinde hazırlansa bile, bunların tam ve doğru olduğundan emin olmak DO-178C rehber dokümanında belirtilen üst seviye yazılım gereksinimleri ile ilgili amaçlar (DO-178C, Tablo A-4) sağlanmalıdır. Bu nedenle, bu gereksinimler yazılım gereksinim standardına da uymalıdır.

Tip-4 ve Tip-5'te de üst seviye gereksinimlere göre tasarım modeli geliştirilir. Geliştirilen modeller, üst seviye gereksinimlere uyumluluklarını doğrulamak üzere gözden geçirilmelidir. Tasarım modeli, DO-178C, Tablo A-4'te yer alan amaçları karşılamalıdır. Bu amaçların bazıları (1,2,4,7,8,9,11) model simülasyonları ile karşılanabilir. İlave olarak tasarım modeli gözden geçirilmeli ve model kapsama analizi yapılmalıdır.

Tip-4 ve Tip-5'te geliştirilen çalıştırılabilir nesne kodunun, üst seviye yazılım gereksinimleri ve tasarım modeli ile uyumlu olduğu doğrlanmalıdır. Kaynak kod ister el ile geliştirilsin ister otomatik üretilsin her durumda gereksinim tabanlı, normal aralık testi ve gürbüzlük testilerini içeren yazılım testlerinin yapılması gerekmektedir. Yazılım testlerinin ardından, geleneksel yazılım geliştirme yaklaşımında olduğu gibi yazılım / yazılım ve yazılım / donanım entegrasyon testlerinin hedef donanım üzerinde yapılması gerekir.

Son aşama olarak geliştirilen yazılım kodunu içeren sistem seviyesinde testler gerçekleştirilerek sistem gereksinimleri ARP4754A standardına göre doğrulanmalıdır.

4.7.2 Tüm yaşam döngüleri için yapılması gereken ortak aktiviteler

Belirtilen beş yaşam döngüsü tipinden hangisi seçilirse seçilsin, aşağıdaki rutin aktivitelerin her fazda gerçekleştirilmesi gerekir.

- Gerekli izlenebilirlikler kurulmalı (MTG teknolojsi kullanımında kurulması gereken izlenebilirlikler Bölüm 5.10'da detaylandırılmıştır)

- Türetilmiş model elemanları etiketlenmeli ve emniyet değerlendirme sürecine iletilmelidir.

- Gereksinim kapsama analizi yapılmalı (her gereksinimin en az bir doğrulama yöntemi ile doğrulandığından emin olmak üzere yapılan analiz)

- Geleneksel yazılım yaşam döngüsü yaklaşımı ile yazılım tasarımı ve kod geliştirilecek ise tasarımın ve kodun, tasarım ve kodlama standardına uyumluluğu gözden geçirilmelidir.

- Geliştirilen tanımlama ve tasarım modellerinin ilgili model standartlarına uyumluluğu gözden geçirilmelidir.

- Üretilen kodun veya simülasyon durum ve prosedürleri otomatik üretilerek kullanılcak ise bu araçları kalifikasyon süreçlerinin tamamlanmış olması gerekir.

- Yazılım için atanan emniyet seviyesine göre gerekli olan bağımsızlık koşulunun sağlanması gerekir.

4.8 MTG Yaşam Döngülerinin Karşılaştırması

Bu bölümde Çizelge 4.1'de verilen MTG yaşam döngüleri, birbirine yakın olan yaşam döngüleri ile karşılaştırılarak avantaj ve dezavantajları açıklanmıştır.

İlk paragrafta Tip-1 ve Tip-2'nin karşılaştırması, avantaj ve dezavantajları, ikinci paragrafta Tip-3'ün avantaj ve dezavantajları ve son paragrafta da Tip-4 ve Tip-5'in karşılaştırması, avantaj ve dezavantajları verilmiştir.

Tip 2'de geleneksel yazılım geliştirme yaklaşımında olduğu gibi üst seviye yazılım gereksinimleri geliştirmek yerine MTG teknoloji ile gelen tanımlama modeli

geliştirilmektedir. Tip 1'de ise, geleneksel yazılım geliştirme yaklaşımında olduğu gibi, yazılım üst seviye gereksinimleri yazılı olarak hazırlanmaktadır. Tip-2'de ise üst seviye gereksinimler görsel olarak modeller ile tanımlanmaktadır. Böylece sistem gereksinimlerini geçerlemek, sistem performansını, ara yüzleri, emniyet kritik bileşenleri, tasarım aşamasına geçmeden simülasyonlar ile doğrulamak mümkün olmaktadır. Bir başka deyişle, MTG teknolojisinin avantajları Tip-2'de üst seviye yazılım gereksinimleri seviyesinden kullanılabilirken, Tip-2'de yazılım tasarım seviyesinde kullanılabilmektedir. Her iki tipte de geleneksel yazılım geliştirme yaklaşımında hazırlanan tasarım yerine MTG teknolojisi ile gelen tasarım modeli hazırlandığı için, otomatik kod üretilebilmektedir. İlave olarak Tip-2'de, üst seviye yazılım gereksinimleri yerine modeller oluşturulduğu için tasarım fazına geçmeden yapılan simülasyonlar ile hatalar yakalanabilmekte ve simülasyon test prosedürleri modellerden otomatik olarak gereksinim fazında kolaylıkla üretilebilmektedir. Tip-1'de sistem ve yazılım seviyeleri halen geleneksel yazılım geliştirme yaklaşımında olduğu net bir çizgi ile halen ayrıdır.

Tip-3 geleneksel yazılım geliştirme yaklaşımına en yakın tiptir. Bu nedenle adaptasyonu en kolay MTG yaşam döngüsü tipidir. MTG teknolojisine geçiş yapmak isteyen organizasyonlar için başlangıç aşaması olarak bu tipin seçilmesi faydalı olur. Geleneksel yaşam döngüsünden tek farkı, üst seviye yazılım gereksinimleri yerine tanımlama modeli geliştirilmesidir. Yazılım tasarımı yine el ile geliştirilir. Bu tipin en büyük dezavantajı, bir tasarım modeli geliştirilmediği için kodun da otomatik üretilemeyeceğidir. Kod bu tipte geleneksel yazılım geliştirme yaklaşımında olduğu gibi el ile geliştirilir.

Tip-4 ve Tip-5 genellikle en az rastlanan MTG yaşam döngüsü tipleridir. Tip-5'de tasarım modelleri sistem seviyesinde hazırlanmaya başladığı için sistem ve yazılım seviyeleri iç içe girmektedir. Bu iki tipin en önemli avantajı, konfigürasyon takibi yapılması gereken verilerin az olmasıdır. Çizelge 4.1'de de görülebileceği gibi bu tiplerde tanımlama modeli geliştirilmemektedir. Her iki tipte de hazırlanan tek model, üst seviye gereksinimleri, yazılım gereksinimlerini ve yazılım tasarımını içeren tasarım modelidir. Bu tiplerin en büyük dezavantajı ise, arada tanımlama modeli olmadığı içini üst seviye gereksinim ile kod arasında büyük bir açıklık vardır. Üst seviye yazılım gereksinimi ve yazılım tasarımını içeren tek bir model

çok fazla bilgi içereceği için anlaması güç olabilir. Tip-4 ve Tip-5 yaşam döngülerinde, her seviye bilgiyi içerdiği için oldukça karmaşık olan tasarım modelini yönetmek ve değişiklik ihtiyacı olduğunda etki analizi yapmak da güçleşir. Burada ayrı bir tanımlama modeli olmadığı için, üst seviye gereksinim ile kod arasında büyük bir adım olur. Bu iki tipin başarısında, sistem mühendisliği ile yazılım mühendisliğinin arasındaki iletişim, çalışma kültürleri, kullandıkları araçların benzerliği gibi faktörler önemlidir. Tasarım modeli geliştirildiği için, her iki tipte de otomatik kod üretilebilir.

4.9 Araçlar arası model transferinde dikkat edilmesi gereken konular

MTG teknolojisinin kullanımında, kullanılan modelleme ve otomatik kod geliştirme araçları ile bu araçların kalifikasyonları son derece önemlidir. Sistem mühendislerinin modelleme yaptığı araçlar ile yazılım mühendislerinin otomatik kod üretimi için kullandıkları araçlar genellikle birbirinden farklı olmaktadır.

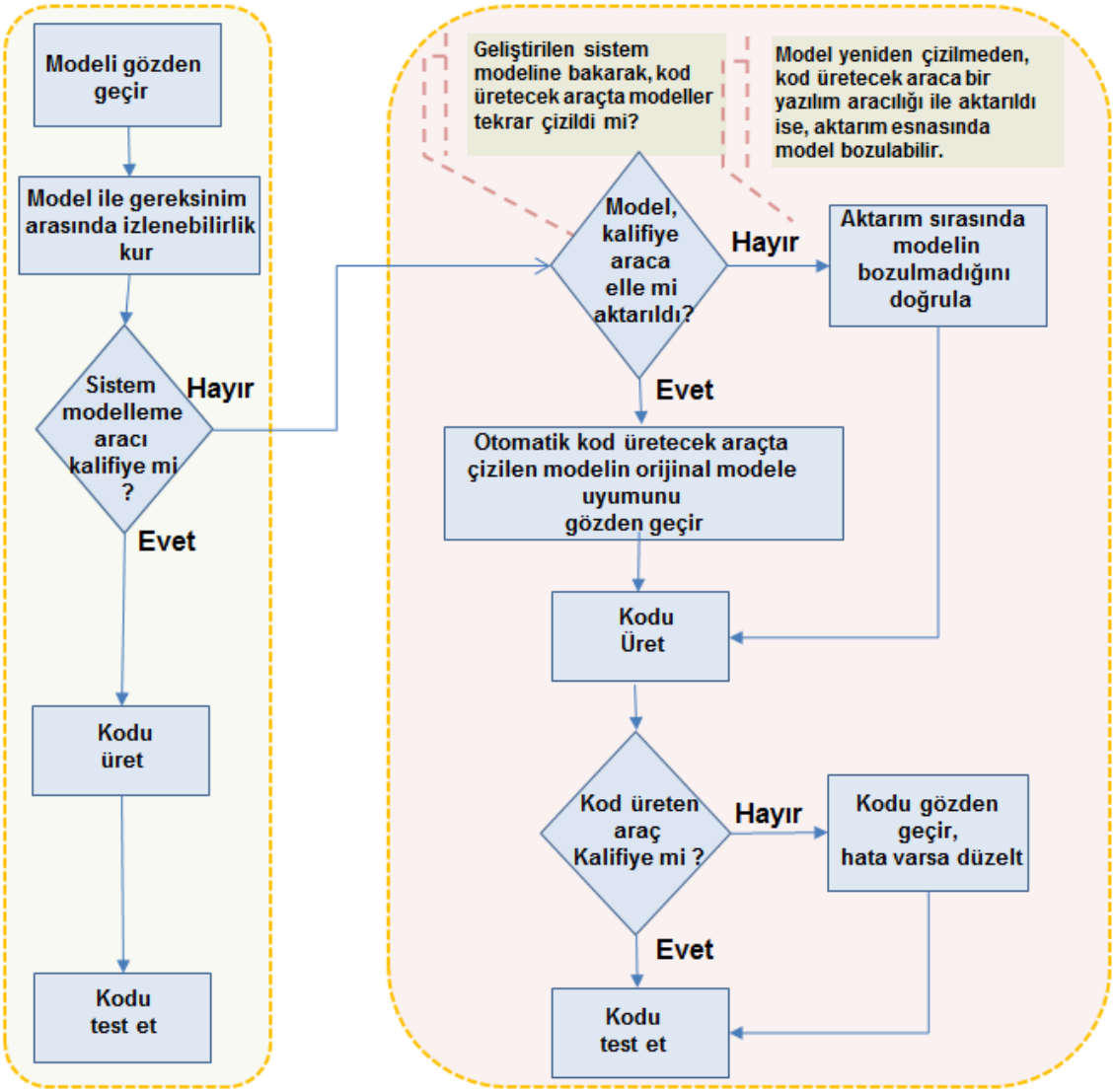
Hava aracı yazılımları gibi emniyet kritik yazılımlar için otomatik kod üreten araçlar genellikle yazılım geliştiricilere kısıtlı bir kütüphane ve geliştirme ortamı sunmaktadır. Sistem mühendisleri ise kısıtların çok olduğu araçlar yerine, geliştirdikleri tasarımlar ile kolaylıkla farklı denemeler yapabilecekleri daha esnek araçları kullanmayı tercih etmektedirler. Bu araçların da genellikle kalifikasyonu zor ve pahalıdır. Bu nedenle, havacılık sektöründe sistem ekibinin tercih ettiği araçlar (örneğin Simulink) genellikle kalifiye araçlar olamamaktadır. Simulink aracı kalifiye olmadığı için Simulink aracında geliştirilen modellerden Simulink aracında otomatik üretilen kod direkt hava aracında kullanılamaz. Özet olarak, sistem ekibi tarafından tercih edilen araçlar ile yazılım ekibi tarafından tercih edilen araçlar genellikle birbirinden farklı olmaktadır.

Uygulamada sistem modelleri bir araçta (havacılık sektöründe genellikle Simulink) geliştirildikten sonra otomatik kod üreten ve kalifiye olan bir başka araçlara (havacılık sektöründe genellikle SCADE) aktarılmaktadır. Bu durum, farklı araçların farklı kütüphaneleri olması, aktarma işini yapan yazılımın kalifiye olmaması gibi nedenlerden dolayı, modellerin bir araçtan diğer araca aktarılırken bozulmasına ve dolayısıyla modelin davranışının değişmesine neden olabilir. Aktarım sırasında modelin bozularak davranışının istemsiz olarak değişmesi, o

modelden üretilecek yazılımın da davranışının istemsiz olarak değişmesine neden olabilir. Bu durum, yazılım ile ilgili temel havacılık kuralını (beklenen fonksiyonu icra etme) ihlal edebilir. Bu nedenle geliştirilerek simülasyonlarla test edilen modelin bir araçtan diğer araca aktarıldıktan sonra bozulmadığının doğrulanması gerekir.

Bu kapsamda kullanılan yöntemlerden birisi Bölüm 8.3'te verilmiştir. Bu yöntem, Simulink ve SCADE araçları arasında kısaca şu şekilde çalışmaktadır. Model sistem ekibi tarafından Simulink aracında geliştirilip doğrulandıktan sonra özel bir ara yüz ile SCADE aracına aktarılarak modelden otomatik kod üretilmektedir. Üretilen kod, s-function ile tekrar Simulink modeline dönüştürülmektedir. S-function MATLAB, C, C++ veya Fortran dilinde yazılan kodun Simulink modeline çeviren bir uygulamadır. Otomatik üretilen koddan elde edilen Simulink modeli ile orijinal Simulink modeli aynı girdiler verilerek beslenir ve her iki modelin ürettiği sonuçlar karşılaştırılır. Bu sonuçlar belirlenen tolerans içinde ise, aktarım esasında modelin davranışının bozulmadığı değerlendirilebilir. Burada bahsedilen toleranslar her bir veri için hava aracına ve veriye göre belirlenmelidir. Toleranslar gereksinim olarak tanımlanmalı ve toleransın doğru belirlenip belirlenmediği alan uzmanları tarafından gözden geçirilmelidir. Bu doğrulama yapıldıktan sonra, üst seviye gereksinimlere göre hazırlanan test prosedürleri ile yazılım test aktivitesinin her durumda yine yapılması gerekir.

Modelin bir araçtan başka bir araca aktarılması söz konusu olan durumlarda, farklı senaryolara göre değerlendirilmesi gereken konular ve yapılması önerilen aktiviteler Şekil 4.6'da verilmiştir. Bu şeklin amacı bir akış şeması vermek değil, MTG teknolojisinde araç kalifikasyonu ve otomatik kod üretimi konuları hakkında dikkate alınması gereken hususların altını çizmektir.



Şekil 4.6 Araç kullanımı kapsamında dikkat edilmesi gereken konular

4.10 İzlenebilirlik

Çizelge 4.1'de anlatılan MTG yaşam döngüsü tipine göre üretilen veriler arasında kurulması gereken izlenebilirlik değişmekte olup aşağıda ayrı ayrı açıklanmıştır.

Tip-1'de aşağıdaki izlenebilirlikler kurulmalı ve gözden geçirilerek doğrulanmalıdır:

- Sistem gereksinimleri ile sistemden yazılıma atanan gereksinimleri arasında
- Yazılıma atanmış gereksinimler ile modelin geliştirildiği gereksinimler arasında

- Modelin geliştirildiği gereksinimler ile tasarım modeli arasında
- Model ile kaynak kod arasında (kaynak kod kalifiye bir araç ile otomatik olarak üretildi ise, model ile kod arasında izlenebilirlik kurmaya gerek yoktur, değil ise kurulmalıdır)

Tip-2'de aşağıdaki izlenebilirlikler kurulmalı ve gözden geçirilerek doğrulanmalıdır:

- Modelin geliştirildiği gereksinimler ile tanımlama modeli arasında
- Tanımlama modeli ile tasarım modeli arasında
- Tasarım modeli ile kaynak kod arasında (kaynak kod kalifiye bir araç ile otomatik olarak üretildi ise, model ile kod arasında izlenebilirlik kurmaya gerek yoktur, değil ise kurulmalıdır)

Tip-3'te aşağıdaki izlenebilirlikler kurulmalı ve gözden geçirilerek doğrulanmalıdır:

- Modelin geliştirildiği gereksinimler ile tanımlama modeli arasında
- Tanımlama modeli ile yazılım tasarımı arasında
- Yazılım tasarımı ile kaynak kod arasında izlenebilirlik kurulmalıdır. (kaynak kod kalifiye bir araç ile otomatik olarak üretildi ise, model ile kod arasında izlenebilirlik kurmaya gerek yoktur)

Tip-4'de aşağıdaki izlenebilirlikler kurulmalı ve gözden geçirilerek doğrulanmalıdır:

- Modelin geliştirildiği gereksinimler ile tasarım modeli arasında
- Tasarım modeli ile kaynak kod arasında (kaynak kod kalifiye bir araç ile otomatik olarak üretildi ise, model ile kod arasında izlenebilirlik kurmaya gerek yoktur)

Tip-5'de aşağıdaki izlenebilirlikler kurulmalı ve gözden geçirilerek doğrulanmalıdır

- Modelin geliştirildiği gereksinimler ile tasarım modeli arasında

- Tasarım modeli ile kaynak kod arasında (kaynak kod kalifiye bir araç ile otomatik olarak üretildi ise, model ile kod arasında izlenebilirlik kurmaya gerek yoktur)

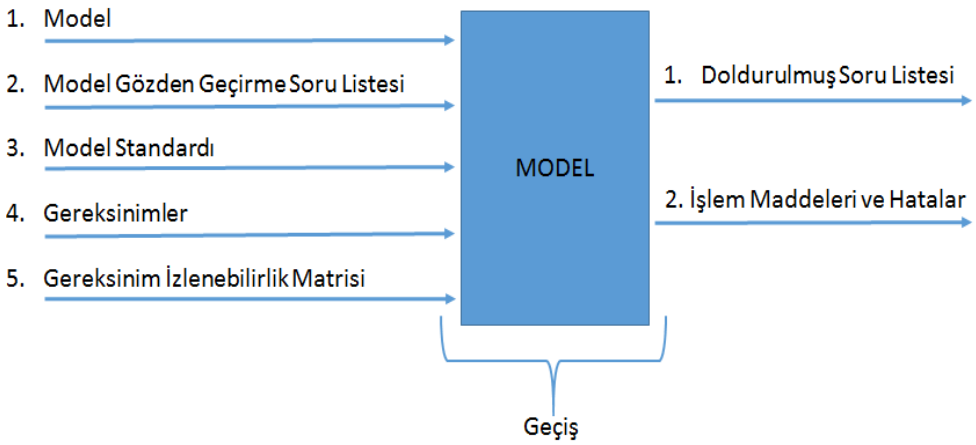
Model ile izlenebilirlik kurulmak istenen üst seviye sistem gereksinim modelleme aracında kurulurken, model elemanının “yorum” bölümüne izlenebilirlik kurulmak istenen gereksinimin numarası verilebilir. Otomatik kod geliştiren araç bu numaraları ilgili kod bloğunun başına otomatik olarak yorum şeklinde getirebilmektedir. Böylelikle kod ile gereksinim arasında el ile izlenebilirlik kurulmasa bile, bu bilgi koda otomatik olarak aktarılabilmektedir. Böylece bir kod bloğuna ait gereksinim kodun içinde görülebilmektedir. Bu bilgi herhangi bir aşamada herhangi bir analiz için fayda sağlayabilir.

4.11 Model Doğrulama

4.11.1 Model Gözden Geçirme

Seçilen MTG yaşam döngüsü tipine göre, tanımlama modeli ve / veya tasarım modeli, seçilen modelleme aracı ve modelleme diline göre geliştirilir. Modeller, ilgili modelleme standardında yer alan kurallara uygun olmalıdır. Model geliştirme sürecinde Bölüm 4.10’da belirtilen izlenebilirliklerin kurulması gerekir. Model geliştirme ve izlenebilirlik kurma aktiviteleri tamamlandıktan sonra, model gözden geçirmeye çıkarılmadan önce konfigürasyon planına göre isimlendirilerek konfigürasyon kontrolü altına alınmalıdır. Model gözden geçirmenin amacı, modelin tam, doğru, geliştirildiği gereksinimler (veya tanımlama modeli) ile uyumlu, model standardına uyumlu, gereksinimlerde olmayan bir fonksiyonun modelde olmadığı, gereksinimlerde olan tüm fonksiyonların da modelde olduğunu gözden geçirmektir.

Şekil 4.7’de model gözden geçirme sürecine ait girdiler ve çıktılar görülmektedir. Bu sürecin girdileri, geliştirilen modeller, gözden geçirme soru listesi, model standardı, üst seviye gereksinimler ve model ile gereksinimler arasında kurulan izlenebilirlik matrisidir. Çıktıları ise, gözden geçirme ekibinde yer alan her bir mühendisin doldurulduğu gözden geçirme soru listesi ve gözden geçirme ekibinin tespit ettiği hatalar ve işlem maddeleridir.



Şekil 4.7 Model gözden geçirme süreci girdi ve çıktıları

Emniyet kritik yazılımlar olan hava aracı yazılımlarında, gözden geçirilen veriler için (bu örnekte model) doldurulan soru listesi grup şeklinde değil, her bir bireyin kendi doldurduğu soru listeleri olmalıdır.

MTG teknolojisinin en kritik süreçlerinde birisi model gözden geçirme sürecidir. Daha önce de belirtildiği gibi havacılık alanında bu teknoloji, genellikle kalifiye araçlar ile modelden kodun otomatik olarak üretilmesi şeklinde kullanılmaktadır. O nedenle modelin davranışı ne ise yazılımın davranışı da o şekilde olacaktır. Modelin davranışının tamlığı ve doğruluğu, modele ait üst seviye gereksinimlerin modelde tam olarak kapsanmasına (ne eksik, ne fazla) bağlıdır. Model, gereksinimleri tam olarak yansıtmıyor ise (örneğin eksik veya fazla fonksiyonlar varsa), modelden üretilen kod ve yazılım davranışı da gereksinimleri tam yansıtmayacaktır. Bu durum, yazılım için temel havacılık kuralı olan “beklenen fonksiyonu” yerine getirme kuralını bozacaktır. Çünkü gereksinimlerde tanımlanan beklenen fonksiyon, modele tam ve doğru olarak aktarılamamış olacaktır. Doğal olarak modelden üretilecek kod da, sistemin beklenen fonksiyonunu icra edemeyecektir. Bu durum hava aracı kaybı gibi ölümlü kazalara bile neden olabilir. Oldukça kritik olan model gözden geçire süreci için aşağıdaki hususların dikkate alınması tavsiye edilmektedir.

- Model gözden geçirmelerine model ile modelin geliştirildiği üst seviye gereksinimler arasındaki uyumluluğu sorgulayacak soruların eklenmesi önerilmektedir.

- Tanımlama modeli ve tasarım modeli geliştirilirken dikkat edilmesi gereken önemli hususlardan birisi de geliştirilen modellerin ilgili model standardına uyumluluğu konusudur. Model gözden geçirmelerine geliştirilen modellerin model standardına uyumluluğunu sorgulayan soruların eklenmesi önerilmektedir. Ancak, uygulamada standartlara uyum konusunda genellikle, "model standardına uyulmuş mu?" şeklinde bir soru eklendiği görülmektedir. İçinde pek çok kural ve tavsiye içeren standartlar için soru listesinde bu şekilde tek bir soru eklemenin yeterli olmayacağı değerlendirilmektedir. Onun yerine model gözden geçirme soru listesine, en azından standartta yer alan kritik kuralların, önemli tavsiyelerin ayrı ayrı bir soru olarak eklenmesinin çok daha faydalı olacağı ve gözden geçirme sürecinin verimliliğini arttıracağı değerlendirilmektedir.

- Model, üst seviye gereksinimde yer alan tüm gereksinimlerin tamamını (eksik veya fazla bir fonksiyon olmadan) içermelidir. Bunu teminat altına alabilmek için yapıl ilk aktivite, model ile üst seviye gereksinim arasında izlenebilirlik kurmaktadır. Modelin sistem davranışını eksiksiz yansıtması, yazılımın davranışını direkt etkileyeceği için bu izlenebilirliklerin doğruluğu ve tamlığı da son derece önemlidir. Model gözden geçirmelerine model ile üst seviye gereksinim arasındaki izlenebilirliği sorgulayacak soruların eklenmesi önerilmektedir.

- Modelin, üst seviye gereksinimler ile uyumluluğu kapsamında yapılan bir başka bir aktivite de model simülasyonudur. Simülasyon kullanımında dikkat edilmesi gereken hususlar Bölüm 5.11.2'de verilmiştir.

- MTG teknolojisinin en önemli avantajlarından birisinin, geleneksek yazılım geliştirme yaklaşımda üretilmesi gereken üst seviye yazılım gereksinimleri, alt seviye yazılım gereksinimleri, yazılım mimarisi, yazılım test prosedürü ve kodu üretilmeden (dolayısıyla bu verilere ait herhangi bir gözden geçime de planlamadan) direkt kodun otomatik olarak üretilebileceği belirtilmişti. Bu durum her ne kadar bu durum takvim ve maliyet açısında avantaj olarak görülse de, geleneksel yöntemde var olan ve farklı aşamalarda yapılan pek çok gözden geçirme aktivitesi burada maalesef elimine edilmiş olmaktadır. MTG teknolojisinde geliştirme fazı ile ilgili yapılan tek gözden geçirme model gözden geçirmesidir. Bu nedenle bu gözden geçirmenin sağlıklı yapılması MTG teknolojisi ile geliştirilecek

yazılımlar için son derece elzemdir. Bu teknoloji kullanımında gözden geçirmelerin da azaldığı bilinci ile hareket ederek, model gözden geçirmeleri için gerekli zamanı hesaplayarak proje başında proje takvimine dâhil edilmesi, yukarıda belirtilen konular ve varsa ilave diğer konuları da içeren soru listeleri ile gözden geçirmelerin yapılması ve model gözden geçirmesine tecrübeli sistem ve yazılım ekibinin dâhil edilmesi tavsiye edilmektedir.

- Bölüm 5.9'da anlatıldığı gibi, modellerin geliştirildiği araç ile otomatik kod üreten araçlar farklı ise araçların kütüphaneleri birbirinden farklı olacağı için, aktarım esnasında bozulan model elemanlarının tespit edilerek kullanılmaması sağlanmalıdır. Bir başka deyişle, bir araçtan diğer araca transfer olurken sorun ile karşılaşılan ve karşılaşılmayan model elemanları tespit edilmelidir. Sistem modelleri geliştirilirken, aktarım esnasında sorun ile karşılaşılmayan model elemanlarının kullanılması sağlanmalıdır. Gözden geçirme soru listesine bu konu ile ilgili bir sorunun da eklenmesi önerilmektedir.

4.11.2 Model Simülasyonu

MTG teknolojinin bu kadar popüler olmasının en önemli nedenlerinden birisi olan model simülasyonu kabiliyeti sayesinde sistem davranışı izlenerek hatalar ve olası performans sorunları, yazılım kodu hiç yazılmadan projenin başında tespit edilebilmektedir.

Model simülasyonu ile bulunabilecek hatalara aşağıdaki örnekler verilebilir:

- Olay ve işlemlerin hatalı sıralanması
- Yazılım gereksinimlerinde belirtilen kriterlerin algoritmalar tarafından sağlanamaması
- Hatalı döngü operasyonları
- Mantıksal hatalar
- Doğru giriş koşulları kombinasyonlarının hatalı işlenmesi
- Eksik veya bozulmuş veri için istenmeyen dönüşler

- Hesaplama sıralamasındaki hatalar
- Algoritma performansı, doğruluğu ve kesinliği ile ilgili bilgi
- Hatalı durum geçişleri

Aşağıdaki hatalar / değerlendirmeler ise model simülasyonu ile bulunamayabilirler.

- Aritmetik hesaplama veya dizi sınırlarının ihlali gibi hataları yanlış yönetme
- Veri bozulmaları (özellikle global verilerdeki bozulmalar)
- Zamanlama ile ilgili yazılım gereksinimler
- Donanım kaynak kullanımı ile ilgili problemler
- Donanım kontrol gereksinimleri

Yukarıda da belirtildiği gibi, bazı hatalar model simülasyonu ile tespit edilemeyebileceğinden hava aracı yazılım doğrulama sürecinde simülasyonlar tek başına yeterli olmazlar. Örneğin, yazılımın yükleneceği hedef ortam ile simülasyon ortamı (kullanılacak işlemci, varsa veri alışverişinde bulunulan FPGA, kullanılan gerçek zamanlı işletim sistemi gibi) tamamen aynı olmayabilir. Bu nedenle, model simülasyonu ile bazı doğrulama aktiviteleri gerçekleştirilse bile aşağıdaki doğrulama aktiviteleri model simülasyonu ile gösterilemezler.

- Kaynak kodun hedef bilgisayar ile uyumluluğu
- Bölütleme (partition) bütünlüğü (farklı yazılım bileşenlerinin kaynak kullanımlarının birbirinden ayrılması ile yapılan bölümleme işleminde oluşabilecek, ortak kullanılan kaynaklarında kaynak sorunu, beklenen zaman değerinin aşılması gibi bölümleme ihlallerinin önlenmesi)
- Modellerin model standardına uyumluluğu
- Kurulan izlenebilirliklerin doğruluğu

Model simülasyonunda dikkat edilmesi gereken pek çok husus vardır. Bu hususlar aşağıda maddeler halinde verilmiştir.

Model simülasyonunun amacı, model ile gereksinimler arasındaki uyumunun tekrarlanabilir kanıtını elde etmektir. Doğrulama sürecinde model simülasyonu kullanılacak ise aşağıdaki hususlar dikkate alınmalıdır:

- Simülasyon ile hangi doğrulama amaçlarının karşılanacağının, hangi aktivitelerin gerçekleştirileceğinin ve hangi verilerin üretileceğinin mutlaka yazılım doğrulama planında açıklanmalıdır. İlave olarak, yukarıda belirtildiği gibi model simülasyonu ile yapılamayan doğrulamaların (yazılım / donanım entegrasyonu gibi) neler olduğu ve bu kapsamda hangi ilave doğrulama aktivitelerinin yapılacağı (geleneksel yazılım geliştirme yaklaşımında gerçekleştirilen yazılım donanım entegrasyon testlerinin yapılması gibi) ve hangi kanıtların üretileceği (entegrasyon test sonuçları gibi) yazılım doğrulama planında net bir şekilde belirtilmelidir [24].

- Model simülasyonunda kullanılacak simülasyon test durum ve prosedürlerinin ilgili üst seviye gereksinimlerden geliştirilmesi ve bu gereksinimler ile aralarında izlenebilirlik kurulmalıdır. Simülasyon test durum ve prosedürleri üst seviye gereksinimlerde belirtilen tüm koşulları, sistemde oluşabilecek tüm girdi ve çıktı setini, içerecek şekilde hazırlanmalıdır. Bir başka deyişle modeller, hem normal aralık testi hem de gürbüzlük testi girdilerini oluşturacak şekilde test edilmelidir. Emniyet kritik bir sistemi besleyen bir girdi için olabilecek her durum test edilerek yazılımın davranışı tanımlanmalıdır (gürbüzlük testi). Örneğin yükseklik bilgisini sağlayan bir cihazın veri aralığı [0-53000] feet olsun. Yazlımın yalnızca bu aralık için davranışının tanımlanması emniyet kritik yazılımlar için yeterli değildir. İlave olarak sınır dışı değer gelmesi durumunda da (örneğin -100 veya 60000 gibi) yazılımın davranışının ne olacağının gereksinim olarak tanımlanması gerekir. Bir şekilde verinin bozulması nedeni ile yazılıma hız bilgisi örneğin -50 olarak geldiğinde yazılımın bunu tespit edebilmesi, çökmemesi ve pilota gerekli uyarıyı (örneğin sınır dışı değer geldiğinde ekranda kırmızı çarpı göster) vermesi gerekir. Bu tip olağan dışı durumları yönetilebilmesi için yazılımdan beklenen davranış sistem seviyesinde gereksinim olarak tanımlanmalı ve modele aktarılmalıdır. Sınır dışı durumlardaki beklenen davranış gereksinimlerde tanımlandıktan sonra bu gereksinimler göre test prosedürleri geliştirilmeli ve yazılım doğrulanmalıdır. Verilen örnek için yapılacak yazılım testlerinin yalnızca, 0, sıfırın sınır değerleri, aralık içinde değerler, 53000 ft ve

53000’ ait sınır değerler ile test edilmesi yeterli olmaz. Sınır dışı değerler de verilerek yazılımın beklenen fonksiyonu (verilen örnekte ekranda kırmızı çarpı gösterilmesi) yerine getirip getirmediği doğrulanmalıdır. Doğrulama sonuçları sertifikasyon otoritesine de sunulmak üzere kayıt altına alınmalıdır.

- Organizasyondaki araç alt yapısın bağlı olarak, modelden otomatik kod üretimi gibi kalifiye araçlar kullanılarak modellerden, simülasyon durum ve prosedürleri de otomatik olarak üretilebilir. Böylelikle, test durum ve prosedürlerinin oluşturulması ve gözden geçirilmesi için ayrı bir iş gücü planlamaya gerek olmaz. Ancak burada dikkat edilmesi gereken önemli bir husus vardır. Hem kaynak kod, hem de kodu doğrulayacak simülasyon durum ve prosedürleri aynı modelden üretildiğinde, model gözden geçirmesi esnasına tespit edilemeyen modeldeki bir hata, hatalı modelden üretilen simülasyon durum ve prosedürleri ile de yakalanamayabilir. Örneğin, uçağın hız bilgisini veren cihaz bu bilgiyi ft/sec cinsinden verirken, sistem seviyesinde yapılan tasarım knot cinsinden tasarlanmış olsun. Bu birim dönüşümü modelde unutuldu ve gözden geçirmelerde de tespit edilemedi ise, modelden üretilen kod, yine aynı modelden üretilen simülasyon prosedürü ile test edildiğinde unutulan birim dönüşümünü yakalanamayabilir. Bu basit örnekten de anlaşılabileceği üzere, simülasyon test durum ve prosedürleri modelden otomatik olarak üretildi ise bu prosedürler yazılı gereksinimlere göre gözden geçirilmeli ve gerekli izlenebilirlikler mutlaka kurulmalıdır. Aksi halde, modeldeki hata, o modele göre üretilmiş simülasyon durum ve prosedürü ile yakalanamayabilir. Böyle bir durum, uçuş emniyetini etkileyebileceği için sertifikasyon otoritesi tarafından da sorgulanacaktır. Çünkü üst seviye gereksinimdeki davranış doğrulanmamış olursa uçuş emniyeti de olumsuz etkilenebilir. Bu konu ile ilgili başka bir örnek Bölüm 4.11.3.1’de verilmiştir.

- Model simülasyonu ile doğrulama yapılacak ise, simülasyonda kullanılan model ile kod üretirken kullanılan model aynı olmalıdır. Benzer şekilde, simülasyonda kullanılan kod ile gerçek donanıma yüklenecek kod da aynı kod olmalıdır. Bu kontrolün hangi aşamada hangi rol tarafından nasıl takip edileceği yazılım planlarında açık bir şekilde ifade edilmelidir.

- Model simülasyon ortamı, yazılım doğrulama planı veya eşdeğer bir dokümanda tanımlanmalı ve simülasyon ortamı tanımlandığı şekilde kurularak kontrol edilmelidir. Bu kontrolün hangi aşamada hangi rol tarafından nasıl kontrol edileceği yazılım planlarında net bir şekilde ifade edilmelidir. İlave olarak, simülasyon ortamı ile gerçek donanım arasında fark olup olmadığı da analiz edilmelidir. Farklılıklar varsa bunlar tespit edilmeli ve bu farklılıkların doğrulama aktivitelerine etkileri değerlendirilerek kayıt altına alınmalıdır.

Model simülasyonu tasarım modellerinin doğrulanmasında aşağıdaki amaçlar için kullanılabilir:

- Tanımlama modelinin üst seviye sistem gereksinimleri ile uyumluluğu
- Tasarım modelleri ile üst seviye gereksinimler arasındaki uyumluluk
- Sistemin doğruluğu ve uyumluluğu
- Sistemin doğrulanabilirliği
- Algoritmaların doğruluğu

Model simülasyonu, yukarıda da belirtildiği gibi hedef bilgisayar ile uyumluluk, standartlara uyum, izlenebilirlik ve bölütleme bütünlüğü doğrulamalarında kullanılamaz. Bu doğrulamalar için ilave doğrulama aktiviteleri planlanmalı ve gerçekleştirilmelidir.

Model simülasyonu test ve kapsama objektifleri kapsamında kullanılacak ise, simülasyon ile bu objektiflerin nasıl tam olarak sağlanabileceğini ait detaylı bir analiz yapılması gerekmektedir. Bu analizde, simülasyon yaklaşımının hata bulma ve bu hatayı düzeltme konusunda çalıştırılabilir nesne kodu ile aynı kabiliyette olduğunun açıklanması gerekir. Bu kapsamda model simülasyon ortamı ile hedef bilgisayar ortamı karşılaştırılmalı ve varsa farklar (örneğin işlemci farklılığı gibi) kayıt altına alınarak etkisi analiz edilmelidir. Akabinde belli bir gereksinim seti, hem hedef bilgisayarda hem de model simülatöründe çalıştırılmalı ve sonuçlar karşılaştırılmalıdır. Model simülasyonunun çalıştırılabilir nesne kodunun doğrulanmasında kullanımı oldukça sınırlıdır ve tavsiye edilmemektedir. Örneğin

Tip-3'te belirtildiği gibi yazılım alt seviye gereksinimleri geleneksel geliştirme yönteminde olduğu gibi el ile yapılacak ise, model simülasyonu ile yapılan testler yeterli olmaz. İlave olarak, yazılım alt seviye gereksinimlere göre yazılım test prosedürleri hazırlanarak bu testlerin de ilave olarak koşturması gerekir.

MTG teknolojisi ile geliştirilen yazılımlarda model simülasyonları model doğrulama amacı ile yaygın olarak kullanılırken, modelden üretilen kodun doğrulanmasında kullanımının çok yaygın olmadığı görülmüştür. Geliştirilen yazılım için böyle bir ihtiyaç var ise, bu konu ile ilgili kurallar ve sınırlamaların yer aldığı DO-331 rehber dokümanının MB.6.8.2 bölümünde yer alan tüm değerlendirilmeler yapılmalı ve gerekli kanıtlar üretilmelidir. Böyle bir planlama var ise, bu konu ile ilgili sertifikasyon otoritesi ile proje başında anlaşma sağlanmalı ve kayıt altına alınmalıdır. MTG teknolojisinde kod üretilse de, çalıştırılabilir kodun doğrulanmasının yine geleneksel yazılım geliştirme yaklaşımında olduğu gibi gereksinim tabanlı testler ile yapılması gerekmektedir.

4.11.3 Model Kapsama Analizi

Model kapsama analizi (MKA), yalnızca tasarım modeli geliştirildi ise yapılan bir analizdir. Buna göre, Tip-3'te tasarım modeli geliştirilmediği için bu analizin yapılmasına gerek yoktur.

Geleneksel hava aracı yazılım geliştirme yaklaşımında yazılım, yazılım gereksinimlerine göre oluşturulan yazılım test prosedürleri çalıştırılarak test edilir. Yapısal kapsama analizi (YKA), koşturulan yazılım testi ile kod içinde çalışmayan kod satırı olup olmadığını tespit edebilmek için kaynak kod üzerinde yapılan bir analizdir. Yazılım testleri esnasında çalışmayan bir kod bölümü var ise, (başka bir deyişle eksik kapsama var ise) bu eksikliğin sebeplerinin analiz edilmesi gerekir. Eksik kapsamanın sebepleri, gereksinim eksikliğinden, test prosedürünün eksikliğinden ve / veya kaynak kod içinde aktive edilmemiş kod, savunmacı programlama veya ölü kod varlığından kaynaklanabilir.

Model kapsama analizi (MKA) ise, modelin geliştirildiği gereksinimlerine göre hazırlanan test prosedürleri ile yapılan simülasyonlar sonucunda, çalıştırılmayan model elemanlarına ait gereksinimlerin olup olmadığını tespit etmek amacı ile

yapılan bir analizidir. Model simülasyonu sonucunda test edilemeyen model elemanları var ise, (başka bir deyişle eksik kapsama var ise) bu eksikliğin sebeplerinin analiz edilmesi gerekir. Eksik kapsama, gereksinim eksikliğinden, simülasyon / test prosedürünün eksikliğinden ve / veya aktive edilmemiş fonksiyona ait model elemanlarının varlığından, üst seviye gereksinimde olmayan fakat modele eklenen türetilmiş model elemanlarından veya istemsiz (örneğin yanlışlıkla) eklenen fonksiyonlardan kaynaklanabilir.

YKA kaynak kodunun testlerle ne ölçüde kapsandığını (test edildiğini) analiz etmek için kaynak kod üzerinden yapılır. MKA ise modele ait gereksinimlerin model simülasyonları ile ne ölçüde kapsandığını (test edildiğini) analiz etmek için model üzerinden yapılır.

MTG teknolojisi kullanımında MKA analizi son derece önemlidir. Üst seviye gereksinimlerde olmadığı halde, yazılım geliştirici tarafından daha iyi bir tasarım yapmak üzere modele eklenen en küçük ilave (yeni bir fonksiyon veya sistem gereksinimlerinde olmayan herhangi bir parametre) modelden yazılım koduna dönüşeceği için yazılımın beklenen fonksiyonunu değiştirerek son derece tehlikeli sonuçlar doğurabilir. Çünkü yazılım geliştirici tarafından (daha iyi bir tasarım olması için bile eklense) modele eklenen en küçük ilave bir parametre / fonksiyon / davranış, uçuş emniyetini negatif yönde etkileyebilir. Örneğin pilota verilecek bir a değeri, sistem gereksinimlerine göre b ve c değerinin toplamı ile hesaplanıyor iken, yazılım mühendisinin daha iyi bir tasarım olacağını düşünerek ilave olarak d değerini eklemesi uçuş emniyetini son derece olumsuz etkileyebilir.

Böyle bir durumda yazılım geliştirici tarafından eklenen ilave parametrenin, sistem gereksinimlerine göre gerçekleştirilen model gözden geçirmesinde tespit edilmesi beklenir. Eğer tespit edilmedi ise, sistem gereksinimlerine göre hazırlanan simülasyon test durum ve prosedürlerine göre yapılan model simülasyonlarında bu durum yakalanacaktır. Bu nedenle simülasyon prosedürlerinin gereksinimlere göre hazırlanması son derece önemlidir. Bu örnekte, a değerini hesaplayan fonksiyonu test etmek üzere hazırlanacak test prosedüründe beklenen çıktı b ve c değerinin toplamı olacaktır. Modelde ise ilave olarak d değeri olduğu için, eklenen d değeri o testin kalmasına neden olur ve model simülasyonunda yakalanır.

Burada bir yukarıda anlatılan bir konunun tekrar açıklanmasında fayda vardır. Bu örnekte yazılım mühendisi tarafından modelde olmadığı halde eklenen ilave d parametresi, modelden otomatik üretilen simülasyon prosedürü ile yakalanamayabilirdi. Çünkü sistem tasarımına göre beklenen çıktı b ve c değerinin toplamıdır. Fakat model, b, c ve d değerini toplamaktadır. Bu durumda da modelden üretilen prosedürde beklenen çıktı b, c ve d değerinin toplamı olacaktır. Model'de b, c ve d toplandı ve modelden üretilen prosedürde de beklenen çıktı b, c ve d toplanarak elde edilmiş oldu. Böylece sistem davranışı öyle olmadığı halde, geliştirilen model simülasyon testini geçecektir. Bu durumda hata yakalanamadan uçağa yüklenmiş olacaktır. Böyle bir hata, otomatik üretilen simülasyon prosedürleri ancak sistem gereksinimlerine göre gözden geçirildiğinde ve izlenebilirlikler kurulduğunda bulunabilir. Bu nedenle üst seviye gereksinimler ile simülasyon prosedürlerinin uyumlu olması, aralarında tam ve doğru bir izlenebilirliğin kurulması ve gözden geçirilmesi son derece önemlidir.

Model simülasyonu, model ile üst seviye gereksinimlerin uyumluluğu görebilmek için son derece güçlü bir araçtır. Ancak model simülasyonu ile tespit edilemeyen uyumsuzluklar da olabilir. Örneğin sistem gereksinimlerinde olmadan modele eklenen ilave fonksiyonlar her zaman model simülasyonu ile yakalanamayabilir. Örneğin, yukarıda verilen irtifa örneğinde sistemde seviyesinde davranış yalnızca [0,53000] aralığı için tanımlanmış, sıfırdan küçük ve 53000'den büyük değerler için yazılımın davranışı (yukarıdaki devrede örnekte ekrana kırmızı çarpı gösterilmesi idi) sistem seviyesinde tanımlanmamış olsun. Yazılım geliştiricinin sistem seviyesinde tanımlanmamış bu eksikliği fark ettiğini ve bu eksikliği gidermek üzere, sınır dışı değerler geldiğinde irtifa bilgisin modelde "0" olarak atadığını varsayalım. Bu atamanın model gözden geçirmesinde yakalanamamış olsun. Yazılım geliştiricinin bu atamayı aslında türetilmiş model elemanı şeklinde etiketlemesi ve emniyet sürecine göndermesi gerekirdi.

Bu örnekte, sınır dışı değerler için sistem seviyesinde bir gereksinim olmadığını ve yazılım mühendisinin türetilmiş model elemanını etiketlemediği varsayılsın. Özellikle karmaşık ve büyük sistemlerde insan faktörünün sisteme hata ekleyebildiği gerçekliği göz ardı edilemez (bu örnekte hata, sistemde olmayan bir davranışı model eklediği halde türetilmiş model elemanı olarak etiketlememesidir).

Model simülasyon prosedürleri, sistem seviyesinde tanımlanan gereksinimlere göre geliştirildiği için simülasyon prosedürlerinde sınır dışı değerlerde yazılımın davranışını doğrulayacak test adımları olmayacaktır. Sonuç olarak model simülasyonu yalnızca sınır içi değerler kullanılarak gerçekleştirilecektir. Sınır içi değeler için yapılan model simülasyonları geçtiği halde, sınır dışı durumlar için yazılım geliştirici tarafından eklenen ilave koşul doğrulanamadan bir sonraki fazda koda dönüşecektir. Böyle bir durum uçuş esnasında bir şekilde oluşur ise, bir başka deyişle oluşan bir hata nedeni ile yazılıma irtifa değeri bir şekilde -50 olarak geldiğinde yazılım, sistem ve emniyet mühendisinin bilmediği ve emniyet değerlendirme çalışmalarında değerlendirmediği bir senaryo ile karşılaşılmış olacaktır. Böyle bir durum oluştuğunda irtifa değeri yazılım mühendisinin kararı gereği sıfır olarak gösterilecektir. Beklide böyle bir durumda sıfır atamak yerine, pilota sesli uyarı vermek veya cihazın ekranını kapatmak veya ekranda kırmızı çarpı göstermek daha emniyetli olabilecektir. Bu örnekten de anlaşılabileceği gibi yazılım mühendisi tarafından verilen kararın sistem ve emniyet mühendisi tarafından geliştirme aşamasında mutlaka değerlendirilmesi gerekir. Pilot çarpı işareti görmediği için bir sorun olmadığını düşünerek uçuşa devam edebilir. Hava şartlarının da kötü olduğu bir gece uçuşu için bu durum son derece tehlikeli sonuçlar doğurabilir. Oysa ekran kararsa veya pilot sesli ikaz ile uyarılsa pilot bu bilgiye güvenmemesi gerektiğini bilir ve olabilecek en yakın piste inebilirdi.

Sistem gereksiniminde olmayan ve yazılım mühendisi tarafından sisteme eklenen davranışın, MKA ile şu şekilde tespit edilmesi beklenir. Yazılım geliştirici tarafından modele eklenen ve sistemde olmayan sınır dışı koşulu ile ilgili davranış, model simülasyonları esnasında çalıştırılamayacağından modelde eksik kapsama oluşacaktır. Eksik kapsama ile ilgili bir örnek, EK 1'de mevcuttur. Bu eksik kapsama analiz edildiğinde, modelde olan fakat sistem gereksiniminde olmayan bu davranış tespit edilmiş olacak ve henüz koda dönüşmeden sistem gereksinimlerinde tanımlanacaktır.

Bu örnek ile MKA analizinin amacı ve nasıl işlediği anlatılmak istenmiştir. Bu tip problemin, ilk olarak model gözden geçirmesinde tespit edilmesi beklenir. Fakat giderek karmaşıklaşan sistem mimarilerinde tasarımcının bazı durumları tanımlamayı gözden kaçırdıkları, yazılım mühendislerinin da bu eksikliği

bildirmeden varsayımlar ile ilerlediği durumlar gözlemlenmiştir. Eksik durumların, ilk olarak model gözden geçirmesinde, orada tespit edilemedi ise model simülasyonlarında ve simülasyon sonuçlarına göre gerçekleştirilen MKA'da tespit edilmesi beklenir. MKA aşamasında da tespit edilemedi ise, bunun son doğrulama olan YKA'da tespit edilmesi beklenir. Fakat örnekteki eksik davranışın YKA aşamasında tespit edilerek düzeltilmesi, MKA aşamasında düzeltilmesinden çok daha maliyetlidir. Çünkü YKA aşaması, yazılım kodu geliştirildikten ve testler tamamlandıktan sonra yapıldığı için projenin sonuna doğru yapılmaktadır.

YKA analizi, MTG teknolojisin özel bir analiz olmadığı için burada detaylı olarak açıklanmamıştır. Fakat bölüm 8.2'de her iki analiz kısaca açıklanarak bir uygulama örneği üzerinden farkları anlatılmıştır.

Projelerde kronikleşen takvim ve maliyet baskısı nedeni ile üreticilerin bu iki analizden birine yönlenme eğiliminde oldukları veya bu iki analizin farkını tam olarak bilmedikleri için birinin yapılmasının yeterli olacağını düşündükleri gözlemlenmiştir. Oysa yukarıda ve Bölüm 8.2'de örneklerle açıklandığı gibi bu iki analiz projenin farklı aşamalarında yapılmakta olup amaçları da birbirinden farklıdır. Her ne kadar bazı hataları her iki analizle de tespit edilebilse ve amaçları benzer görünse de bu analizler birbirinde farklıdır ve ilgili fazlarda yapılmalıdır. Bu nedenle MKA ve YKA çalışmalarının proje başında planlanarak takvime eklenmesi son derece önemlidir.

MTG teknolojisinde her ne kadar kaynak kod modelden üretiliyor ve modelin davranışını sergiliyor olsa da, bu verilerin model ve kaynak kod olmak üzere iki ayrı yazılım yaşam döngüsü verisi olduğu unutulmamalıdır. Aynı modelden iki ayrı otomatik kod geliştirme aracı ayı fonksiyonu icra eden iki farklı kod üretebilir. YKA kod üzerinden yapıldığı için, model kapsama analizi yapmak, yapısal kapsama analizini ihtiyacını ortadan kaldırmaz. Bu durum, Bölüm 8.2'de örnekler üzerinden açıklanmıştır.

Model kapsama kriterleri ve olası eksik kapsama nedenler aşağıda verilmiştir. Tasarım modelinin, model kapsama analizi yapılırken aşağıdaki kriterler kullanılabilir. Yazılım geliştiren firma, model kapsama analizinde hangi kriterleri kullanacağını doğrulama planına yazmalıdır.

- Mantıksal bileşenler ve bu bileşenlerin tüm koşulları üst seviye gereksinimlerde tanımlandığı şekilde modelde yer alıyor mu?

- Tüm denklik sınıflar üst seviye gereksinimlere göre limit içinde ve dışındaki durumlar için tanımlanmış mı? Fonksiyonel bileşenlerin ve algoritmalar için yazılım davranışının değiştiği durumlar üst seviye gereksinimlere uyumlu bir şekilde tanımlanmış mı? (örneğin bir bölme işlemi için, bölenin sıfır olma durumu)

- Sonlu otomatlardaki tüm geçişler ve koşulları üst seviye gereksinimlerde tanımlandığı şekilde modelde tanımlanmış mı?

- Sistemdeki tüm fonksiyonların yalnızca iyi senaryodaki değil, kötü senaryodaki davranışları da (örneğin yazılımın kilitlenmesi durumunda watchdog fonksiyonunun tetiklenmesi gibi) ilgili üst seviye gereksinimlerde tanımlandığı şekilde modelde tanımlanmış mı?

- Üst seviye gereksinimde olmayan, türetilmiş bir model elemanı olarak da etiketlenmemiş bir fonksiyon veya model elemanı var mı?

Hava aracı yazılımlarında model kapsamasının % 100 olması gerekir. %100 kapsama sağlanması demek, her model elemanının ilgili üst seviye gereksiniminde tanımlı olduğu, modelde olan fakat üst seviye gereksinimde olmayan hiç bir fonksiyon olmadığı (türetilmiş model elemanları hariç), modeldeki tüm geçişlerin sistemde tanımlı olduğu şekilde olduğu, modelin geliştirildiği gereksinimlere göre yaratılan testler ile modelin ve modele ait gereksinimlerin tamamının tüm koşullar altında test edilebildiği anlamına gelir. Yapılan model kapsama analizinde genellikle ilk seferde %100 kapsama elde edilememekte, eksik kapsama oluşabilmektedir. Eksik kapsamanın aşağıdaki nedenleri olabilir:

- Gereksinim tabanlı yapılan testlerde eksikler: Hava araçlarında yapılan yazılım testlerinin / model simülasyonlarının, gereksinim tabanlı olması gerekir. Bir başka deyişle, yazılım fonksiyonlarını icra eden tüm gereksinimler doğrulanmalıdır. Doğrulama aşamasında yapılan iş, gereksinimlere göre simülasyon test durum ve prosedürü hazırlamak ve o prosedürü koşturmak şeklindedir. Gereksinim setindeki bazı gereksinimler için simülasyon prosedürü yazılmamış olur ise, o prosedürler koşturulamayacağı için o gereksinime ait model elemanları çalıştırılamamış olur ve

eksik kapsama oluşur. Model elemanının simülasyonla çalıştırılamaması, o modele elemanına ait üst seviye gereksinimin kapsanamadığı anlamına gelir. Bu durumda, kapsanamayan model elemanlarına ait gereksinimlere göre yeni simülasyon prosedürleri oluşturularak ilave simülasyonlar gerçekleştirilmelidir.

- Modelin geliştirildiği gereksinim setindeki yetersizlikler: Bu konu ile ilgili kapsama eksikliği, modelde var olan, fakat üst seviye gereksinimler olması gerektiği halde unutulan gereksinimler nedeni ile oluşur. Alan uzmanı bir mühendis, gereksinimde yer almasa bile olması gereken davranışı modele eklemiş olabilir. Eklenen bu davranışa ait gereksinim olmadığı için, test prosedürü de olmayacağından bu model elemanları simülasyon testlerinde çalıştırılamamış olur. Kapsama analizi yapıldıktan sonra, kapsama eksiğinin eksik gereksinim olduğu belirlenirse, modelde tanımlanan davranış yeni gereksinim olarak üst seviye gereksinimlere eklenmeli, bu gereksinimlere ilave simülasyon prosedürleri oluşturulmalı ve koşturulmalıdır.

- Tasarım modeli ile ifade edilen türetilmiş gereksinimler: Bu konu ile ilgili kapsama eksikliği, modelde olan fakat üst seviye gereksinimde olmayan ve çok detay kalacağı için olması beklenmeyen durumlarda oluşan kapsama eksikliğidir. Üst seviye gereksinimde olmadığı için, testi de olmayacağından eksik kapsama oluşur. Modelde tanımlanan davranış, üst seviye gereksinim olarak eklenebilecek şekilde ise yukarıda belirtildiği gibi yeni gereksinim eklenerek, yeni test oluşturulup yeni prosedürlere göre model simülasyonu yapılır. Modelde tanımlanan davranış, üst seviye gereksinime eklenmeyecek kadar detay ise, bu davranış üst seviye gereksinime ayrıca eklenmez. Bu durumda, bu davranış türetilmiş model elamanı olarak etkilenmeli ve test, muayene veya analiz yöntemlerinden biri ile doğrulanmalıdır [25].

- Tasarım modelinde aktive edilmeyecek fonksiyonlar: Model içinde, başka müşteriler için, başka hava aracı konfigürasyonlarında çalışmak üzere eklenen aktive edilmemiş fonksiyonlar olabilir. Örneğin, A uçağı için X ve Y fonksiyonları, B uçağı için bunlara ilave olarak Z fonksiyonu gerekli olsun. Bu durumda geliştirici, her üç fonksiyonu icra edebilecek bir yazılım geliştirerek iki farklı müşterinin beklentisini tek bir üründe karşılamak isteyecektir. Böyle bir durumda Z fonksiyonu

A uçağı için aktive edilmemiş fonksiyondur. A uçağında Z fonksiyonuna ihtiyaç olmadığı için, model simülasyonunda Z fonksiyonu ile ilgili testler yapılmaz. Bu durumda Z fonksiyonu ile ilgili model elemanları çalışmayacağından eksik kapsama oluşur. Bu durum, MKA raporunda açıklanmalıdır. İlave olarak, hava aracı için aktive edilmemiş olsa da bu fonksiyon mutlaka test edilmeli, bu fonksiyonun aktivasyonunun nasıl engellendiği belirtilmeli ve istemsiz bir şekilde aktive olmayacağı analiz, simülasyon, test vb. yöntemler ile gösterilmelidir.

- Tasarım modelindeki istemsiz fonksiyon: Modele, gereksinimde olmayan bir fonksiyonun yanlışlıkla eklenmesi durumunda oluşan kapsama eksikliğidir. Gereksinimde olmayan bir fonksiyon modele eklenince, gereksinim bazlı yapılan testlerde o fonksiyona ait model elemanları çalışmaz ve eksik kapsama çıkar. İstemsiz bir fonksiyon çok tehlikeli sonuçlara yol açabileceği için modelden silinmelidir.

Yukarıda anlatılan analizlere ilave olarak, yapılması gereke analizlerden birisi de gereksinim kapsama analizidir. Bu analiz, MTG teknolojisi kullanımına özel bir analiz olmayıp geleneksel yazılım geliştirme yaklaşımında da yapılması gereken bir analizdir. Gereksinim kapsama analizi, gereksinimlerin hazırlandığı seviyeden bağımsız olarak (sistem veya yazılım) bütün gereksinimlerin en az bir doğrulama yöntemi (gözden geçirme, analiz veya test) ile doğrulandığından emin olmak için yapılan bir analizdir. Gereksinim kapsama analizi genellikle her bir gereksinim için o gereksinimi test edecek test prosedürü arasında izlenebilirlik kurularak yapılmaktadır.

5 MTG TEKNOLOJİSİNDE YAZILIM YAŞAM DÖNGÜSÜ SÜREÇLERİ

Bölüm 5'de MTG teknolojisi hakkında detaylı bilgiler verilerek yapılması gereken aktiviteler anlatılmıştır. Bu bölümde ise, MTG teknolojisi kullanımında yazılım yaşam döngüsünün her bir sürecinde, geleneksel yazılım geliştirme sürecinden farklı olarak yapılması gereken ilave çalışmalar verilmiştir.

5.1 Yazılım Planlama Süreci

Yazılım planlama süreci, ihtiyaç duyulan yazılımı geliştirmek üzere izlenecek yazılım yaşam döngüsü süreçlerini, bu süreçlerde gerçekleştirilecek aktiviteleri ve üretilecek yazılım yaşam döngüsü verilerini tanımlar.

Geleneksel yazılım geliştirme yaklaşımında olduğu gibi MTG teknolojisinde de yapılması gereken aktiviteler yazılım planlarında açık bir şekilde tanımlanmalı, bu planlama üzerinde sertifikasyon otoritesi ile projenin başında hem fikir kalınması ve geliştirme süreci boyunca bu planların takip edilmesi hava aracının sertifikasyon süreci ve uçuş emniyeti için son derece önemlidir.

Planlama sürecinin başlıca amaçları şunlardır:

- Sistem mimarisine göre emniyet ekibi tarafından yazılıma atanan emniyet seviyesini adreslemek. (Atanacak emniyet seviyesine göre planlama aktivitelerinde değişiklik olabilir. Örneğin seviye D bir yazılım için, yapısal kapsama analizi ile ilgili planlama yapmaya gerek yoktur)

- Yazılım geliştirme süreçlerinde gerçekleştirilecek aktiviteleri, süreçler arasındaki ilişkiyi ve veri alış verişini, yazılım seviyesi ile sistem seviyesi arasındaki geri besleme mekanizmasını ve süreçler arasındaki geçiş kriterlerini adreslemek

- Yazılım yaşam döngüsü boyunca kullanılacak araçları, geliştirme ve doğrulama ortamını adreslemek.

- Gereksinim, tasarım ve kodlama standartlarını yazılım planlarından adreslemek.

MTG teknolojisi ile planlama sürecine gelen ilave konular aşağıda verilmiştir.

- Yazılım geliştirme planında, seçilen MTG yaşam döngüsü süreci tipi (Tip-1, Tip-2, Tip-03, Tip-4, Tip-5) belirtilmelidir. Seçilen tipe göre "Tanımlama Modeli" ve/veya "Tasarım Modeli" olarak hangi yaşam döngüsü verilerinin üretileceği net bir şekilde adreslenmelidir.

- Modelleme araçları ve model gözden geçirme soru listesi gibi kullanılacak soru listeleri de yazılım doğrulama planında adreslenmelidir.

- MTG teknolojisinden farklı bir yöntemle (örneğin geleneksel yazılım geliştirme yaklaşımı) geliştirilecek yazılım bileşenleri var ise bu bileşenler kapsamında gerçekleştirilecek aktiviteler de ayrıca planlarda belirtilmelidir. İlave olarak, üretilen otomatik kodun dış dünya ile ara yüzünü sağlamak gibi ihtiyaçlar nedeniyle el ile geliştirilecek kod ile ilgili geliştirme ve doğrulama faaliyetleri de planlarda tanımlanmalıdır.

- Seçilen model kullanım tipine göre (Bkz Çizelge 4.1) sistem seviyesinde de bazı aktivitelerin gerçekleştirilmesi gerekebilir. Sistem seviyesinde gerçekleştirilecek geliştirme ve doğrulama aktiviteleri de ilgili sistem planlarından adreslenmelidir. MTG teknolojisi kullanımında, sistem gereksinimlerini doğrulamak / geçerlemek için ARP 4754A standardının da proje sözleşmesi ile projeye dâhil olması gerektiği unutulmamalıdır.

- Modeller sistem seviyesinde geliştirilerek doğrulansa bile, bu aktivitelerin anlatıldığı ilgili sistem planları yazılım planlarından adreslenmelidir.

- Model geliştirme ile otomatik kod üretme araçları farklı ise, modellerin araçlar arasındaki transferinin nasıl olacağı ve bu transferde modellerin bozulmadığından nasıl emin olunacağına dair yöntem tanımlanmalıdır. Uygulama bölümünde bu konu ile ilgili bir örnek mevcuttur.

- Modelden üretilen kodun ne şekilde doğrulanacağı, varsa sistem seviyesinden beklenen girdilerin neler olacağı ve doğrulama sonuçlarının sistem seviyesine nasıl sağlanacağı yazılım doğrulama planından adreslenmelidir.

- Model simülasyonu ile yazılım doğrulama aktivitesi gerçekleştirilecek ise, model simülasyonu ile hangi doğrulama aktivitelerinin gerçekleştirileceği, gerçekleştirilemeyen doğrulama aktiviteleri için (yazılım / donanım entegrasyon testi gibi) yapılacak ilave doğrulama aktivitelerinin ve üretilecek doğrulama sonuçlarının neler olacağı yazılım doğrulama planında belirtilmelidir. İlave olarak, model simülasyon ortamı, simülasyon ortamında olabilecek değişiklikler kapsamında izlenecek süreç (varsa ilave doğrulama aktiviteleri) simülasyon araçlarının varsa limitasyonları yazılım doğrulama planından adreslenmelidir.

- Yazılım doğrulama planında, Bölüm 4.11.3.1'te açıklanan model kapsama kriterleri tanımlanmalıdır.

- Model kullanım tipine göre "Tanımlama Modeli" ve "Tasarım Modeli" ayrı ayrı geliştirilecek ise, her ikisi içinde ayrı model standartları tanımlanmalı ve yazılım geliştirme planında adreslenmelidir.

- Yazılım konfigürasyon yönetim planından model konfigürasyon yönetimi ve model değişiklik yönetimi sürecinin nasıl yürütüleceği ve ilgili roller ve sorumluluklar ile birlikte tanımlanmalıdır.

- Geliştirme aşamasında geliştirme veya doğrulama araçları kullanıldı ise bu araçların kalifikasyon ihtiyacının olup olmadığı ve kalifiye edilmesi gerekiyor ise kalifikasyon ile ilgili planlama yapılmalıdır.

- Simülatörün kullanım amacı göz önünde bulundurularak, varsa limitasyonları yazılım doğrulama planlarında adreslenmelidir.

5.2 Yazılım Geliştirme Süreci

Geleneksel yazılım geliştirme süreci, yazılım gereksinim oluşturma, yazılım tasarım, yazılım kodlama ve entegrasyon süreçlerinden oluşmaktadır.

5.2.1 Yazılım Gereksinim Süreci

Yazılım gereksinim sürecinde, sistemde tanımlanan fonksiyonları icra edecek üst seviye yazılım gereksinimleri oluşturulur. Üst seviye yazılım gereksinimlerinin hazırlanıp hazırlanmama durumu, seçilen yaşam döngüsü tipine göre

değişmektedir. Örneğin Tip-1'de üst seviye yazılım gereksinimlerini hazırlanırken, Tip-5'te hazırlanmadan, sistem gereksinimlerine göre geliştirilen tasarım modelinden direkt kod üretilmektedir.

MTG teknolojisi ile yazılım gereksinim sürecine gelen ilave konular aşağıda verilmiştir. Seçilen yaşam döngüsü tipine göre hazırlanacak veri ve bu verinin uyacağı standart değişmektedir.

- Tip-2 ve Tip-3'te üst seviye sistem gereksinimlerine ve tanımlama modeli standardında yer alan kurallara göre tanımlama model oluşturulmalıdır. Tanımlama modeli ile üst seviye sistem gereksinimleri arasında izlenebilirlik kurulmalı ve model gözden geçirme soru listesine göre modeller gözden geçirilmelidir. Tip-4 ve Tip-5'te, üst seviye sistem gereksinimlerinden direkt tasarım modellerine geçildiği için bu süreç bu iki tip için uygulanabilir değildir.

- Üst seviye sistem gereksinimlerinde yer almadığı halde modele eklenen türetilmiş model elemanı tanımlama modeli standardında belirtildiği şekilde etiketlenmeli ve emniyet değerlendirme sürecine iletilmelidir.

- Yazılımın davranışı ile ilgili bilgiler içermeyen model elemanları (örneğin, açıklama içeren model elemanları) modeli standardında belirtildiği şekilde etiketlenmelidir. Çünkü bu elemanların kaynak koda girmemesi gerekir.

5.2.2 Yazılım Tasarım Süreci

Yazılım tasarım sürecinde, tanımlanmış olan yazılım üst seviye gereksinimlerine göre, yazılım mimarisinin ve yazılım alt seviye gereksinimlerin oluşturulur.

MTG teknolojisi ile yazılım tasarım sürecine gelen ilave konular aşağıda verilmiştir. Seçilen yaşam döngüsü tipine göre hazırlanacak veri ve bu verinin uyacağı standart değişmektedir.

- Tip-1'de üst seviye yazılım gereksinimleri ve tasarım modeli standardına göre tasarım modeli hazırlanmalıdır. Hazırlanan tasarım modeli ile üst seviye yazılım gereksinimleri arasında izlenebilirlik kurulmalı ve tasarım modeli gözden geçirme soru listesine göre gözden geçirilmelidir.

- Tip-2'de tanımlama modeli ve tasarım modeli standardına göre tasarım modeli hazırlanmalıdır. Hazırlanan tasarım modeli ile tanımlama modeli arasında izlenebilirlik kurulmalı ve tasarım modeli gözden geçirme soru listesine göre gözden geçirilmelidir.

- Tip-3'te tanımlama modeli ve tasarım standardına göre yazılım tasarım dokümanı hazırlanmalıdır. Bu doküman, geleneksel yazılım geliştirme yaklaşımında hazırlanan, yazılım tasarım (alt seviye yazılım gereksinimleri ve yazılım mimarisi) dokümanıdır. Hazırlanan yazılım tasarımı ile tanımlama modeli arasında izlenebilirlik kurulmalı ve tasarım gözden geçirme soru listesine göre tasarım gözden geçirilmelidir. Tip-4 ve Tip-5'te, üst seviye sistemden gereksinimlerinde direkt tasarım modellerine geçildiği için bu süreç bu iki tip için uygulanabilir değildir.

- Tip-4 ve Tip-5'te üst seviye sistem gereksinimlerine ve tasarım modeli standardına göre tasarım modeli hazırlanmalıdır. Hazırlanan tasarım modeli ile üst seviye sistem gereksinimleri arasında izlenebilirlik kurulmalı ve tasarım modeli gözden geçirme soru listesine göre gözden geçirilmelidir. Tip-4 ile Tip-5'in farkı şekilde Çizelge 4.1'de de görülebileceği gibi tasarım modelinin hangi seviyede (sistem veya yazılım) başladığı ile ilgilidir.

- Üst seviye sistem / yazılım gereksinimlerinde yer almadığı halde modele eklenen türetilmiş model elemanı tanımlama modeli standardında belirtildiği şekilde etiketlenmeli ve emniyet değerlendirme sürecine iletilmelidir.

- Yazılımın davranışı ile ilgili bilgiler içermeyen model elemanları (örneğin, açıklama içeren model elemanları) modeli standardında belirtildiği şekilde etiketlenmelidir. Çünkü bu elemanların kaynak koda girmemesi gerekir.

5.2.3 Yazılım Kodlama Süreci

Yazılım mimarisi ve alt seviye yazılım gereksinimlerinde oluşan yazılım tasarım verisinden, yazılım kaynak kodu geliştirilir. Daha önce de belirtildiği gibi havacılık sektöründe bu teknoloji kullanılırken genellikle tasarım modellerinden kalifiye araçlar ile otomatik kod üretilmektedir.

MTG teknolojisi ile yazılım kodlama sürecine gelen ilave konular aşağıda verilmiştir.

- Tip-3'te tasarım modeli geliştirilmeyip, geleneksel yazılım geliştirme yaklaşımında olduğu gibi yazılım alt seviye gereksinimler ve yazılım tasarımı geliştirildiği belirtilmiştir. Bu tipte bir tasarım modeli geliştirilmediği için modelden otomatik kod üretimi de gerçekleştirilmez. Tip-3'te kodlama standardı ve yazılım tasarımına göre ile kod geliştirilir. Bu tipte geliştirilen kod, yazılım tasarımına ve kodlama standardına uyum kapsamında gözden geçirilmelidir.

- Diğer tiplerde, tasarım modelinden otomatik olarak kod üretilebilmektedir. Modelin geliştirildiği araç ile otomatik kod üreten araç farklı ise, modeller geliştirildikleri araçtan kod geliştirme aracı aktarılırken bozulabileceği, bunun belirlenecek bir yöntemle kontrol edilmesi gerektiği belirtilmişti. Geleneksel yazılım geliştirme yaklaşımı ile geliştirilen hava aracı yazılımları için kod ile tasarım arasında izlenebilirlik kurulması gerekir.

Burada ise, kalifiye araç ile otomatik kod geliştirildiğinde kod ile gereksinim arasında izlenebilirlik kurulmasına ve üretilen kodun gözden geçirilmesine gerek yoktur. Bununla beraber, uygulamada model geliştirilirken model elemanlarına yorum olarak ilgili üst seviye gereksinim numarasının girilebilmekte, bu numaralar otomatik kod üretilirken üretilen kodun ilgili bölümüne otomatik olarak "yorum" şeklinde gelebilmektedir. Böylelikle kod otomatik üretilse bile, koda ait ilgili üst seviye gereksinim kolayca koddan görülebilmektedir.

- Otomatik kod geliştirme aşamasında, araçta yapılan konfigürasyon ayarları kayıt altına alınmalıdır.

- Kod üretme aşamasında, varsa aracın verdiği hata ve uyarıların tek tek incelenerek gerekli düzeltmeler yapılmalıdır. Düzeltmeler yapıldıktan sonra aracın ürettiği, herhangi bir uyarı veya hata durumunun kalmadığını gösteren ekran görüntüsünün doğrulama sonuçlarına raporuna eklenmesi tavsiye edilmektedir.

5.3 Entegrasyon Süreci

Entegrasyon süreci, yazılım / yazılım entegrasyonu ve yazılım donanım entegrasyonu olmak üzere iki aşamadan oluşmaktadır. Yazılım / yazılım entegrasyonunda, doğrulama süreci tamamlanmış yazılım bileşenlerinin bir arada doğru bir şekilde çalışıp çalışmadığı doğrulanır.

Yazılım / donanım entegrasyonunda ise, entegrasyonu tamamlanmış yazılımın, hedef donanım üzerinde doğrulanır. Öncelikle yazılım / yazılım entegrasyonu kapsamında geliştirilen yazılım bileşenlerinin entegrasyonu yapılır. Daha sonra, yazılım / donanım entegrasyonu kapsamında yazılım hedef bilgisayara yüklenir.

MTG teknolojisinde bu süreç ile ilgili yeni bir aktivite yoktur. DO-178C rehber dokümanında yer alan entegrasyon süreci aktivitelerinin gerçekleştirilmesi yeterlidir.

5.4 Yazılım Doğrulama Süreci

Doğrulama, planlama ve geliştirme sürecinde oluşturulan çıktıların teknik olarak değerlendirmesidir. Doğrulama sürecinin temel amacı, o aşamaya kadar geliştirilmiş verilerdeki (kaynak kod, gereksinim gibi) hataları bulmaktır.

MTG teknolojisi ile doğrulama sürecine gelen ilave konular aşağıda verilmiştir.

- Klasik yazılım yaşam döngüsü sürecinde, sistem seviyesinde üretilen gereksinimler yazılım sertifikasyon kapsamına girmez. MTG teknolojisi kullanılıyor ise, Çizelge 4.1'de görüldüğü gibi sistem seviyesinde üretilen veriler yazılım sertifikasyon kapsamına girebilmektedir. Örneğin Tip-5'te görülebileceği gibi yazılım tasarım verisini oluşturan Tasarım Modeli sistem seviyesinde geliştirilmektedir. Bu nedenle sistem seviyesinde yapılan doğrulama aktiviteleri ve buna ait kanıtlar da (örneğin ARP4754A) sertifikasyon süreci kapsamında üretilmeli ve kayıt altına alınmalıdır. Bu kanıtlar sertifikasyon otoritesinin yapacağı gözden geçirme / denetimlerde de sorgulanacaktır [26].

- Geliştirilen tanımlama modeli ve tasarım modelinin ilgili modelleme standardına uyumu kapsamında model standartlarına göre gözden geçirilmesi gerekir.

- Geliştirilen tanımlama modelinin ilgili üst seviye gereksinimlere uyumluluğunun gözden geçirilmesi gerekir. Model gözden geçirme kapsamında kullanılabilecek bir soru listesi Bölüm 7.19'da verilmiştir.

- Geliştirilen tasarım modelinin ilgili tanımlama modeline uyumluluğunun gözden geçirilmesi gerekir.

- Model kapsama analizi yapılmalıdır.

- Simülasyon durum ve prosedürlerinin, ilgili model gereksinimlerine uyumluluğunun, normal aralık ve gürbüzlük testlerini içerip içermediğinin gözden geçirilmesi gerekir.

- Model simülasyonları kullanılacak ise, Bölüm 5.11.2'de detaylı olarak anlatılan konuların dikkate alınması gerekir.

- Model gözden geçirme, MKA ve YKA aktiviteleri için ayrı ayrı iş gücü ve zaman planlaması yapılmalı ve proje takvimine eklenmelidir.

5.5 Yazılım Konfigürasyon Yönetim Süreci

Yazılım yaşam döngüsü boyunca geliştirilen verilerin tanımlı bir ortamda kontrollü bir şekilde saklanması için gerekli alt yapının kurulması ve bu alt yapıya göre verilerin yönetilmesi anlamına gelir. Hava aracı yazılım geliştirme süreçleri arasından en kritik süreçlerden birisi konfigürasyon yönetim sürecidir. Özellikle karmaşık sistemlerde milyonlar seviyesindeki kod sayısı, binlerce gereksinim ve sayfalarca mimariden oluşan yazılımın verilerini yönetmek hiç kolay değildir. Bu kadar yoğun verinin konfigürasyon yönetiminde yapılacak en ufak bir hata, uçağın kaybına dahi neden olabilir. Örneğin, model ile modelden üretilen kod arasındaki konfigürasyonda bir hata olursa, uçağa eski sürümlü bir modele göre üretilmiş kod yüklenebilir. Bu durum, yazılımın temel havacılık kuralı olan beklenen davranış kuralını bozabilir. Bu nedenle, yazılım gereksinimlerinin hangi sürümü, tasarımın hangi sürümü ile uyumlu, tasarımın hangi sürümü hangi kaynak kod sürümü ile uyumlu, yazılım gereksinimlerinin hangi sürümü, yazılım test prosedürlerinin hangi sürümü ile uyumlu gibi birbirileri ile ilişkili verilere ait konfigürasyonların sistematik bir şekilde takip ediliyor olması gerekir.

MTG teknolojisi ile konfigürasyon yönetim sürecine gelen ilave konular aşağıda verilmiştir.

- Yazılım konfigürasyon planında, tasarım ve tanımlama modellerine ne şekilde isim ve numara verileceği tanımlanmalıdır.

- Geliştirilen modellerin nasıl yayımlanacağı ve yayım duyurusunun sistem ve yazılım ekibine nasıl yapılacağı planlarda tanımlanmalıdır.

- Geliştirilen modeller gözden geçirme sürecinden önce ve sonra sürümleri ile birlikte kayıt altına alınmalıdır. İlave olarak, modellerin sürümleri, uyumlu oldukları (geliştirildikleri) üst seviye gereksinimlerin veya tanımlama modeli sürümleri, geliştirme sürecinde takip edilen model standardının sürümü, modelden üretilen kodun sürümü ve model simülasyon durum ve prosedürlerinin uyumlu sürümleri ile birlikte yayımlanması ve takip edilmesi çok kritiktir. Örneğin A sistemine ait bir modelin 1.0 sürümünden üretilen kodun sürümü 1.2 olarak etiketlendi ise, model 2.0 sürümüne yükseldi ve henüz yeni modelden kod üretimi yapılmadı ise, mevcut kodun (1.2) modelin 1.0 sürümü ile uyumlu olduğu, bir başka deyişle 2.0 modeline uyumlu olmadığı net bir şekilde görülebilmelidir. Aksi halde uçağa model ile uyumlu olmayan kod yüklenebilir. Yazılım konfigürasyon planında model ile modele ait otomatik üretilecek kodun sürümünün nasıl takip edileceği belirtilmelidir.

Bu kapsamda, konfigürasyon durum raporunda (örneğin Çizelge 5.1'deki gibi bir tablo), modelin sürümü ile modelden üretilen kodun sürümü bir arada tutularak takip edilebilir. İlave olarak, model simülasyonunda kullanılan model ile kodun geliştirildiği modelin aynı olması hususu da oldukça önemlidir. Bu bilginin de konfigürasyon durum raporunda sistematik bir şekilde tutulması elzemdir.

Planlama sürecinde, takip edilmesi gereken tüm hususların, ne şekilde, hangi roller ile yapılacağı ve nasıl teminat altına alınacağı netleştirilerek kayıt altına alınmalıdır.

Çizelge 5.1 Konfigürasyon durum raporu örneği

Konfigürasyon Maddesi	**No**	**Sürümü**	**Yayın Tarihi**	**Uyumlu olduğu Model Standardı Sürümü**	**Uyumlu Olduğu Tanımlama Modeli ve**	**Üretilen Kodun Sürümü**
Yakıt_Sistemi_ Tasarım_Modeli	YS_Mdl	1.0	29.02.19	1.0	Yakıt_Sistemi _Tanımlama Modeli / 2.0	1.2
Yakıt_Sistemi_ Tasarım_Modeli	YS_Mdl	2.0	19.04.19	1.0	Yakıt_Sistemi _Tanımlama Modeli / 3.0	1.8
Yakıt_Sistemi_ Tasarım_Modeli	YS_Mdl	3.0	27.06.19	2.0	Yakıt_Sistemi _Tanımlama Modeli / 3.0	1.9

- Konfigürasyon durum raporuna veriler yeni sürümleri eklenirken, eski sürüm bilgilerinin de kaybedilmemesi gerekir.

- Modelden direkt kod geliştirildiği / üretildiği için model değişiklik yönetimi son derece kritiktir. Sistem ve yazılım ekibinin bir arada çalıştığı MTG teknolojisinde sistem ve yazılım ekibi arasındaki iletişim çok önemlidir. Bu iletişim yönteminin ne olacağı, bir başka deyişle, yazılım ekibi tarafından sistem seviyesinde tespit edilen bir hatanın sistem seviyesine nasıl iletileceği ve yazılım seviyesindeki bir hatanın sistem ekibi tarafından yazılıma hangi şekilde iletileceği planlarda net bir şekilde tanımlanmalıdır. Örneğin, tanımlama / tasarım modelinde bir değişiklik olduğunda, bu değişiklik ilgili tüm taraflara (sistem ve yazılım ekibine) sistematik bir yöntemle iletilmeli, değişiklik ile ilgili sağlıklı bir şekilde etki analizi yapılmalı ve güncellenmesi kararlaştırılan verilerin güncellenerek doğrulanması gerekir. Yazılıma atanan seviye A veya B ise, değişikliği yapan kişi ile doğrulayan kişilerin farklı olması gerektiği unutulmamalıdır.

- Yazılım konfigürasyon veya doğrulama planında değişiklikler kapsamında etki analizinin anlatıldığı bölümde, MTG teknolojisi kapsamında üretilen verilerdeki (tanımlama modeli, tasarım modeli, otomatik üretilecek kod gibi) değişiklikler için etki analizinin ne şekilde yapılacağı ve doğrulanacağı da açıklanmalıdır. Bu analiz genel olarak, kurulan izlenebilirlikler üzerinden gerçekleştirilmektedir. Özellikle A ve B seviyeler için gerekli değişiklikler ve doğrulamalar yapıldıktan sonra, yazılımın temel fonksiyonlarını doğrulayan küçük bir set ile ilave bir doğrulama yapılması ve bu setin tanımlanması önerilmektedir.

- Modeli güncellerken, yalnızca değişiklik isteğinde belirtilen değişikliğin (ne eksik ne fazla) yapılması hususu da emniyet kritik yazılımlar için son derece önemlidir. Bu kapsamda yayımlamış bir veride herhangi bir değişiklik yapılması gerekiyor ise bu değişiklik isteği için bir kayıt açılmalı ve ilgili kişilere atanmalıdır. Değişiklik yapıldıktan sonra da doğrulanmalıdır. Burada dikkat edilmesi gereken husus, değişikliğin yalnızca açılan değişiklik isteği kaydında yer aldığı kadar olması hususudur. Eğer ilave değişiklikler yapma ihtiyacı var ise, bu değişiklikler de değişiklik isteğine eklenerek kayıt altına alınmalıdır. Aksi halde ilave yapılan bu değişiklikler doğrulanamayabilir. Örneğin modelin a ve b elemanları ile ilgili bir değişiklik isteği kaydı oluşturulmuş olsun. Yazar a ve b elemanını değiştirdikten sonra, ilave olarak konu ile ilgili olan c elemanında da değişiklik ihtiyacı olduğunu değerlendiriyor ise, mevcut değişiklik isteği kaydına c değişikliği ile ilgili gereklilik de eklemelidir. Aksi halde c değişikliğinden, isteği açan kişi haberdar olmayabilir. Bu durumda, a ve b ile ilgili değişiklik isteğini açan kişi, modelde a ve b ile ilgili bu değişikliğin doğru bir şekilde gerçekleştirilip gerçekleştirilmediğini doğrularken, fazladan yapılan c değişikliğini görmeyebilir. C değişikliğinin yapılmaması gerekiyor ise veya yanlış değiştirildi ise bu doğrulama aşamasında yakalanamamış olur ve uçuş emniyetini olumsuz etkileyebilir.

MTG teknolojisinde bu konu ile ilgili özel durum şudur. Modelleme araçları iki sürüm arasındaki farkı görsel olarak gösterme konusunda kabiliyetli olmayabilir. Modelde yapılan değişiklikleri görebilmek, yazılı gereksinimlerdeki değişiklikleri görmekten daha zordur. Yazılı gereksinimlerin iki sürümü arasındaki farklar basit bir karşılaştırma ile görülebilirken, modelin iki sürümü arasındaki değişiklikleri tespit etmek modelleme aracının yeteneğine bağlıdır. O nedenle, modelde

istemsiz (hata raporunda olmayan bir değişiklik) bir değişikliğin olup olmadığının nasıl kontrol edileceği ve bu aktiviteden hangi rolün sorumlu olacağı ilgili yazılım planlarında belirtilmelidir.

5.6 Yazılım Kalite Teminatı Süreci

Yazılım kalite teminatı sürecinin amacı, yazılım geliştirme süreçlerinin onaylanmış plan ve standartlara uygun olarak yürütüldüğünü teminat altına almaktır. Proje başladığında, yazılım geliştirme ekibi kurulurken yazılım kalite sorumlusunun da atanması gerekir. Yazılım kalite sorumlusunun, havacılık yazılımı geliştiren organizasyonlarda geliştirme ekibinden organizasyonel olarak bağımsız olması gerekir.

Yazılım kalite sorumlusu, planlama, geliştirme, doğrulama süreçlerinin belli aşamalarında denetim yaparak, gereksinim, tasarım, kod, test prosedürü gibi ürünlerin gözden geçirmelerine katılarak ve yazılım testlerinin koşulmasına şahitlik ederek yürütülen yazılım süreçlerini proje boyunca izler.

Hava aracı yazılımı geliştiren firmalarda yazılım kalite sorumlusunun diğer sektörlerden farklı olarak yazılım geliştirme aşamasının sonunda doğru yapması gereken “Uygunluk Gözden Geçirmesi” aktivitesi vardır. Uygunluk gözden geçirmesinin amacı, planlanan yazılım yaşam döngüsü süreçlerinin tamamlandığını, yazılım yaşam döngüsü verilerinin tamamının üretildiğini, çalıştırılabilir nesne kodunun konfigürasyon kontrolü altında olduğunu ve her seferinde aynı kodun üretilebildiğini teminat altına almak olan kalite güvence sorumlusu tarafından gerçekleştirilen bir çeşit denetimdir.

MTG teknolojisi ile yazılım kalite teminatı sürecine gelen ilave konular aşağıda verilmiştir.

MTG teknolojisi kullanılarak hava aracı yazılımı geliştirilecek ise yazılım kalite sorumlusu model gözden geçirmelerine katılarak veya denetimler yaparak aşağıdaki aktiviteleri gerçekleştirilmelidir:

- MTG teknolojisi ile ilgili planlarda yer alan tüm geliştirme ve doğrulama aktivitelerinin gerçekleştirildiğinden ve varsa sapmaları değerlendirerek kayıt altına alındığından emin olunmalıdır.

- Modellerin, model standartları ve model gözden geçirme soru listelerine göre ilgili rollerin katılımı ile tecrübeli mühendislerin de yer aldığı bir gözden geçirme ekibi ile gözden geçirildiğinden emin olunmalıdır.

- Modellerin, planlarda belirtilen modelleme araçları ve sürümleri kullanılarak geliştirildiği kontrol edilmelidir.

- Gerekli izlenebilirliklerin planlarda belirtildiği şekilde kurularak gözden geçirildiğinden emin olunmalıdır.

- DO-178C rehber dokümanına göre hava aracı yazılım geliştirme sürecinde herhangi bir yazılım fazına geçebilmek için gerekli şartların (geçiş kriteri) planlarda tanımlanması gerekir. Örneğin, kodlama fazına geçiş için geçiş kriteri, yazılım tasarımının tamamlanmasıdır. MTG teknolojisi ile geliştirilecek yazılımlar için bu teknolojiye özel veriler de (tanımlama modeli, tasarım modeli gibi) ilgili fazlarda geçiş kriterleri olarak tanımlanmalıdır. Örneğin, yazılım geliştirme planında, kodlama süreci geçiş kriterlerinin arasında “modellerin gözden geçirilerek yayımlanması” gibi bir kriter olduğundan emin olunması gerekir. Fazlar arasındaki geçişlerin de planlarda tanımlanan geçiş kriterlerine uygun yapıldığı kontrol edilmelidir.

- Resmi yazılım testlerinde otomatik kod üretildiği durumda, kod geliştirme aracında herhangi bir uyarı veya hata verilmediğinin kontrol edilmesi gerekir.

- Model simülasyonu ile doğrulama yapılacak ise, geliştirme ekibinin doğru simülasyon ortamının planlarda tanımlandığı şekilde olup olmadığından emin olunması gerekir.

- Model simülasyonunda veya kapsama analizinde tespit edilen eksiklerin planlara uygun kayıt altına alınıp alınmadığından emin olunması gerekir.

- Model standartlarından ve / veya planlarda yer alan MTG teknolojisinden sapma varsa bunların onaylı olduğundan ve kayıt altında olduğundan emin olunması gerekir.

5.7 Sertifikasyon Süreci

Sertifikasyon sürecinin amacı, yazılım geliştiren firma ile o yazılımın kullanıldığı uçağın tip sertifikasını yayınlayacak sertifikasyon otoritesi arasındaki iletişimi kurmak ve yazılım uyum dokümanı olarak belirlenen dokümanları otorite ile paylaşarak üzerinde hem fikir olunmasını sağlamaktır. Sertifikasyon otoritesi, hava aracının yazılım uyum dokümanlarını inceleyerek ve yazılım yaşam döngüsünün planlama, geliştirme, doğrulama ve kapanış fazında ayrı ayrı yapacağı detaylı yazılım denetimleri ile geliştirilen yazılımı ve verileri inceleyerek uçuş iznini verir ya da vermez.

MTG teknolojisi ile sertifikasyon sürecine gelen ilave konular aşağıda verilmiştir.

MTG teknolojisi kullanılan projelerde projenin başında yapılacak aktiviteler ve üretilecek veriler üzerinde sertifikasyon otoritesi ile hem fikir kalınması hususu çok önemlidir. Bu teknoloji yeni olduğu için, önceki uygulamalarda sertifikasyon otoritesi çalışılarak kazanılmış çok fazla deneyim bulunmamaktadır. O nedenle her aşamada sertifikasyon otoritesi ile koordineli çalışmak proje takvim ve maliyeti açısından elzemdir. Sertifikasyon otoritesinin bitmiş bir faz ile ilgili geriye dönük isteyeceği ilave bir çalışma projenin sertifikasyon sürecini son derece olumsuz yönde etkileyebilir.

Sertifikasyon otoritesinin, bu çalışmada belirtilen tüm konular ile birlikte özellikle aşağıdaki konuları sorgulayacağı değerlendirilmektedir:

- Otomatik kod / simülasyon durum ve prosedürü üretilecek ise araç kalifikasyonu
- Model ile modelin geliştirildiği gereksinimler arasındaki izlenebilirlik
- Model gözden geçirmeleri

- Üst seviye gereksinimler ile simülasyon/ durum ve prosedürleri arasındaki izlenebilirlik
- Modellerin doğrulama yöntemi
- Modellerin değişiklik yönetimi, sistem ve yazılım ekibi arasındaki iletişim yöntemi
- Modellerin konfigürasyon yönetimi ve uyumlu sürümlerin takip yöntemi
- Model kapsama analizi ve bu analizin sonuçları

Bölüm 7.19'da, sertifikasyon kapsamında sorulabilecek sorular verilmiştir.

6 MTG TEKNOLOJİSİNDE YAZILIM YAŞAM DÖNGÜSÜ VERİLERİ

Hava aracı yazılımının sertifiye edilebilmesi için DO-178C ve MTG teknolojisi kullanılıyor ise DO-331 rehber dokümanında yer alan amaçların sağlanması ve üretilen kanıtların sertifikasyon otoritesine sunulması gerektiği belirtilmişti. Yazılım yaşam döngüsü aktiviteleri harfiyen yerine getirilse bile sertifikasyon otoritesi ancak üretilen kanıtları (üretilen yazılım yaşam döngüsü verileri) görerek uçuş iznine karar verir. Bu nedenle bu kanıtların gerekli içerikler ile birlikte üretilmesi son derece önemlidir.

Aşağıdaki bölümde, MTG teknolojisi ile geliştirilen yazılımların, yazılım yaşam döngüsü verilerine eklenmesi gereken ilave bilgiler verilmiştir.

6.1 Yazılım Sertifikasyon Konuları Planı

Yazılım sertifikasyon konuları planı, sertifikasyon süreci için son derece önemli bir plan olup otorite ile yürütülecek aktiviteler ve üretilecek veriler üzerinde anlaşma sağlanan plandır. Bu planda, MTG teknolojisinin ne şekilde kullanılacağı, bu teknoloji ile geliştirilecek yazılım bileşenleri ve üretilecek veriler belirtilmelidir.

6.2 Yazılım Geliştirme Planı

Yazılım Geliştirme Planı'nda, yazılım geliştirme aşamasında kullanılacak tüm yazılım standartları (gereksinim standardı, tasarım standardı ve kodlama standardı) adreslenmektedir. MTG teknolojisi kullanılacak ise, geliştirilen model standartları, modelleme metotları, modelleme araçları ve model geliştirme ortamı yazılım geliştirme planında belirtilmelidir.

6.3 Yazılım Doğrulama Planı

Yazılım Doğrulama Planı'nda, yazılım yaşam döngüsü boyunca geliştirilen verilerin hangi metotlar ile doğrulanacağını, hangi aktivitelerin gerçekleştirileceği, bağımsızlık kriteri tanımlanır. MTG teknolojisi kullanılacak ise, bu verilere ilave olarak modellerin nasıl gözden geçirileceği, model gözden geçirme soru listesi, izlenebilirlik analizinin nasıl yapılacağı, model kapsama kriterleri ve model kapsama analizi ile ilgili bilgileri tanımlamalıdır. Ayrıca, model simülasyonu kullanılacak ise, model simülasyon ortamı, kullanılacak araçlar, model simülasyon

metotları, üretilecek simülasyon sonuçları, simülasyon ile hangi doğrulama aktivitelerinin gerçekleştirilmiş olacağı ve simülasyon ile hangi doğrulama aktivitelerinin gerçekleştirilemeyeceği açıklanmalıdır. İlave olarak, simülasyon prosedürlerin nasıl hazırlanacağı, bu prosedürler ile kurulacak izlenebilirlikler ve hazırlanacak prosedürlerin gözden geçirmesinde kullanılacak soru listeleri de yazılım doğrulama planından adreslenmelidir.

6.4 Yazılım Konfigürasyon Planı

Yazılım Konfigürasyon Planı'nda, yazılım yaşam döngüsü boyunca geliştirilen verilerin nasıl ve hangi ortamda konfigürasyon kontrolü altına alınacağı tanımlanır. MTG teknolojisinin kullanımı durumunda, modellerin nasıl isimlendirileceği ve numaralandırılacağı, proje ekibine nasıl duyurulacağı, modellerin hangi ortamda konfigürasyon kontrolü altına alınacağı ve değişiklik yönetim sürecinin nasıl olacağı belirtilmelidir. İlave olarak, yazılım konfigürasyon planından model ve model ile ilgili diğer verilerin (gereksinim, kaynak kod, test / simülasyon prosedürü ve sonuçları) uyumlu olan sürümlerinin takip edilebileceği konfigürasyon durum raporunun adreslenmesi gerekir.

6.5 Yazılım Kalite Güvence Planı

Yazılım Kalite Güvence Planı'nda, yazılım plan ve standartlarına uyumluluğun nasıl teminat altına alınacağı, yazılım yaşam döngüsü fazlarında yapılması planlanan yazılım kalite faaliyetleri ve üretilecek kalite kayıtları adreslenir. MTG teknolojisinin kullanılacak ise, model geliştirme, model doğrulama, model konfigürasyon yönetimi süreçleri ve model simülasyonu süreçleri boyunca yürütülen aktivitelerin planlara ve standartlara uygunluğunu teminat altına almak için yazılım kalite güvence sorumlusu tarafından gözen geçirme veya denetimler yapılmalıdır. İlave olarak, planlarda yer alan geçiş kriterlerinde modeller ve seçilen MTG teknolojisi yaşam döngüsünde üretilen verilerin de yer aldığının ve süreçler arasındaki geçişlerin bu kriterle göre sağlandığının teminat altına alınması gerekir. Yazılım kalite sorumlusu tarafından DO-178C dokümanının Bölüm 8.3 başlığı altında tanımlanan uygunluk gözden geçirmesi çalışmasına, ayrı bir madde olarak MTG teknolojisi süreçleri ve çıktılarının da dâhil etmesi önerilmektedir. Modeller için model standardından sapma var ise, bu sapmaların onaylı olduğundan ve

konfigürasyon kontrolü altında olduğundan emin olunmalıdır. İlave olarak, model simülasyonu kullanılacak ise simülasyona başlamadan önce, simülasyon ortamının planlandığı şekilde olup olmadığı kontrol edilmelidir. Yazılım kalite güvence planı, belirtilen tüm yazılım kalite güvence aktiviteleri zamanları ile birlikte (şu fazdan önce şu aktiviteler tamamlanmalıdır şeklinde) tanımlanmalıdır.

6.6 Yazılım Gereksinim Standardı

Bu standart, yazılım üst seviye gereksinimleri geliştirirken kullanılacak metotları, kuralları ve araçları tanımlar. MTG teknolojisi kullanımı durumunda, üst seviye gereksinimlerin yazımı ile ilgili kuralların tanımlanması gerekir. Çizelge 4.1'de yer alan Tip-4 ve Tip-5 için modelin geliştirildiği gereksinimler (üst seviye sistem gereksinimleri) sistem seviyesinde geliştirilse de, bu gereksinimlerin DO-178C rehber dokümanında yer alan üst seviye yazılım gereksinimleri gibi ele alınması gerektiği, o nedenle de bu gereksinimlerin yazılım gereksinim standardına uyması gerektiği unutulmamalıdır.

6.7 Yazılım Tasarım Standardı

Bu standart, yazılım alt seviye gereksinimleri ve mimariyi geliştirirken kullanılacak metotları, kuralları ve araçları tanımlar. MTG teknolojisi kullanılacak ise, geliştirilen tasarım modelleri model standardına uymalıdır. MTG yaşam döngüsü olarak Çizelge 4.1'de yer alan Tip-3 seçildi ise, yazılım tasarımı modelleme aracı ile değil, geleneksel yazılım yaşam döngüsü sürecine göre geliştirilecek demektir. Bu kapsamda DO-178C rehber dokümanında yer alan yazılım tasarım amaçları sağlanmalıdır ve yazılım tasarım standardında yer alan kurallara uyulmalıdır.

6.8 Kodlama Standardı

Bu standart, yazılım kodlama sürecinde uyulması gereken kuralları tanımlar. MTG teknolojisi kullanılıyor ise, kod genellikle modelden kalifiye bir araç ile otomatik olarak üretilmektedir. Bu nedenle kodlama standardı MTG teknoloji kullanımında ilave bir içerik ihtiyacı bulunmamaktadır. Kod, araç yerine el ile geliştirilecek ise bu kodun kodlama standardında yer alan kurallara uyması gerekir.

6.9 Model Standardı

Model standardı, kullanılacak modelleme tekniğini, araçları ve modelleme ile ilgili kuralları içerir. Modelleme standardı geliştirme ekibi tarafından yüksek kalitede ve birbiri ile uyumlu modeller geliştirmesi için son derece önemlidir. MTG teknolojisinde ileride yapılacak düzeltmeler veya ilave müşteri istekleri için, otomatik üretilmiş kod güncellenmez. İlgili gereksinim ve model güncellenerek kod tekrar üretilir. Her ne kadar bu teknolojide geliştirilen kodun karmaşıklığı ve bakım maliyeti söz konusu olmasa da, bu hususlar geliştirilen modeller için geçerlidir. Örneğin çok karmaşık bir modelin testi ve bakım maliyeti de oldukça yüksek olur. İlave olarak, özellikle emniyet kritik sistemlerde karmaşık bir modeldeki güncellemelerin regresyon analizini yapmak da zor olur.

MTG teknolojisinde üretilen kod modele bağlı olduğu için, bu teknolojide kod için Modelin bakım kolaylığını sağlamak, bu teknolojide kod yerine modellerin kalite özelliklerinin tanımlanması gereklidir. Kalite özelliklerinin arasında, bakım kolaylığı, modülarite, karmaşıklık, test edilebilirlik, tutarlılık, tekrar kullanılabilirlik sayılabilir. MTG teknolojisinde bu özelliklerin tanımlandığı ve kurallara bağlandığı yer model standardıdır. Örneğin modelden kod üretme kapsamında SCADE aracı kullanılacak ise, model standardında hem bu araç ve varsa aracın kısıtları adreslenmeli hem de hem de hedeflenen model kalite özellikleri tanımlanmalıdır.

Model standardına uyum ve standarttaki kuralların tüm geliştirme ekibi tarafından aynı şekilde anlaşılması çok önemlidir. Aksi halde, geliştirilen modelin davranışı ile ilgili algı kişiden kişiye farklılık gösterebilir. Model standardının daha iyi anlaşılabilmesi için, hem sistem hem de yazılım ekibinin bir araya gelerek standardın üzerinden tek tek geçmesi çok önemlidir.

Uygulamada standartlarda yer alan bazı kuralların tüm proje ekibi tarafından aynı şekilde anlaşılmadığı durumlar gözlemlenmiştir. Bu durumun önüne geçmek için tanımlanan her bir kuralın altına bir örnek verilmesi ve proje başında tüm kuralların üzerinden proje ekibi ile geçilmesi tavsiye edilmektedir. Bu çalışma, kuralların herkes tarafından aynı şekilde anlaşılmasına katkı sağlayacağı için son derece faydalı olacaktır. Model standardında en az aşağıdaki bilgiler olmalıdır.

- Model geliştirirken kullanılacak araç ve yöntemler
- Kullanılacak modelleme dili
- Dilin karmaşıklık kısıtları ile ilgili bilgiler (örneğin, isimlendirme kuralı, kullanılabilir model elemanları, semboller listesi, modellerin içinde bulunduğu diyagram düzeni, maksimum iç içe geçme sayısı, her bir diyagramdaki en fazla izin verilen model elemanı sayısı, en fazla izin verilen katman sayısı gibi)
- Modelleme aracındaki kısıtlar ve (örneğin zamanlama metotları, global veri kullanımı gibi) model eleman kütüphanesi
- Model içinde gereksinimleri etiketleme ve model ile ilgili yazılım yaşam döngüsü elemanları arasında kurulacak izlenebilirlik yöntemi
- Model kalite özellikleri (bakım kolaylığı, modülarite, karmaşıklık, test edilebilirlik, tutarlılık, tekrar kullanılabilirlik gibi)
- Modelde yer alan türetilmiş gereksinimleri etiketleme yöntemi ve bu gereksinimlerin emniyet değerlendirmesine aktarılma süreci
- Modelin içinde, sonraki aşamaya girdi olmayacak elemanların (örneğin yorum bloğu) nasıl belirleneceğine ait yöntem

Aşağıda tipik bir model standardında olabilecek kurallardan örnekle verilmiştir:

- Tüm değişkenler, sabitler, etiketler aynı formatta olmalı
- Değişken isimleri veri tipini içermeli
- Uzun isimlendirmeler için “_” ayrım yapılmalı
- Sinyal akışı soldan sağa doğru olmalı
- Sinyal isimleri 16 karakter ile sınırlı olmalı
- Türetilmiş model elemanları için yorum alanına “türetilmiş” ifadesi yazılmalı
- Girdilerin veri tipleri aynı olmalıdır

6.10 Yazılım Gereksinim Spesifikasyonu

Sistem gereksinimlerinden yazılıma atanan gereksinimlere göre, yazılımın icra edeceği fonksiyonlar, performans kriterleri, yazılım-donanım ara yüzleri gibi bilgileri içeren gereksinimlerdir.

MTG teknolojisinin kullanılacak ise, yazılım gereksinim spesifikasyonu, tanımlama modelinde yer alan gereksinimler de olabilir veya tasarım modelinin geliştirileceği ayrı bir gereksinim seti de olabilir.

6.11 Tasarım Tanımı

Üst seviye yazılım gereksinimlerinin nasıl gerçekleştirileceği, yazılım mimarisi, veri yapıları, veri ve kontrol akışı gibi bilgiler içeren dokümandır. MTG teknolojisinin kullanımı durumunda, geliştirilen tasarım modeli (ayrı bir yazılım tasarım tanımlama dokümanı olmayacak ise) yazılım tasarımını (alt seviye yazılım gereksinimleri ve yazılım mimarisi) içerebilir.

6.12 Kaynak Kod ve Çalıştırılabilir Nesne Kodu

MTG teknoloji kullanımında kaynak kod ve çalıştırılabilir nesne kodu ile ilgili ilave bir amaç yoktur.

6.13 Yazılım Doğrulama Durum ve Prosedürleri

Yazılım doğrulama süreci aktivitelerinin nasıl gerçekleştirileceğinin anlatıldığı ana dokümandır. Bu dokümandan, adım adım testlerin nasıl icra edileceğini anlatan test prosedürleri adreslenir.

MTG teknolojisi kullanılacak ise, model simülasyonu ile yapılacak doğrulamalar için, simülasyon durumunun amacı, modele verilecek girdiler, koşullar, beklenen sonuç ve geçti / kaldı kriteri simülasyon durumlarında tanımlanmalıdır. Simülasyon prosedürlerinde ise, simülasyon durumlarının nasıl çalıştırılacağı, adım adım her bir simülasyon durumu, simülasyon sonuçlarının nasıl değerlendirileceği ve kullanılacak simülasyon ortamı tanımlanmalıdır. Bu dokümanda simülasyon durum ve prosedürleri ile üst seviye gereksinimler arasında izlenebilirlik içermeli veya adreslemelidir.

6.14 Yazılım Doğrulama Sonuçları

Yazılım test, gözden geçirme ve analiz sonuçlarının derlendiği dokümandır. MTG teknolojisinin kullanımı durumunda, model simülasyon sonuçlarının yer aldığı dokümandır. Bu dokümanda simülasyon durum ve prosedürleri ile sonuçları arasında izlenebilirlik kurulmalıdır. İlave olarak, test sonuçları ile bu sonuçların ait olduğu model sürümü de sonuç raporunda yer almalıdır.

6.15 Yazılım Yaşam Döngüsü Ortam Konfigürasyon İndeksi

Yazılım yaşam döngüsü konfigürasyon indeksi, yazılım geliştirme ve doğrulama süreçlerinde kullanılacak araçların ne olduğunu ve sürüm bilgilerini içerir. MTG kullanılacak ise, bu indekste ilave olarak model geliştirme, doğrulama veya simülasyon süreçlerinde geliştirilecek araçlar sürümleri ile birlikte yer almalıdır.

6.16 Yazılım Konfigürasyon İndeksi

Bu doküman, yazılım ürününe ait tüm verileri sürümleri ile birlikte adresler. Bu kapsamda, yazılım planları, standartlar, gereksinimler, tasarım, kaynak kod, test prosedürleri gibi, yazılım ürününe ait tüm veriler yazılım geliştirme fazının sonunda yayınlanarak adreslenir. MTG teknolojisinin kullanımı durumunda, tanımlama modeli, tasarım modeli, model simülasyon durum ve prosedürleri, doğrulama sonuçları dokümanları, bu modellerden üretilen kod sürümü ile birlikte bu dokümandan adreslenmelidir. Bu dokümanda, yaşam döngüsünde çıkan verilen her sürümü değil, yalnızca birbiri ile uyumlu olan (modelin sürümü ve o sürüme ait üst seviye gereksinimlerin / modelin sürümü, o modelden üretilen kodun sürümü, o modele ait üst seviye gereksinimlerden geliştirilen simülasyon durum ve prosedürlerinin sürümü, simülasyon sonuçlarının sürümü) sürümlerinin yer alması gerekir.

6.17 Yazılım Başarım Özeti

Bu doküman yazılım geliştirme ve doğrulama süreci bittiğinde yazılım sorumlusu tarafından hazırlanan bir dokümandır. Bu doküman temel olarak, geliştirilen yazılımın ilgili havacılık kurallarına uygun geliştirildiğini, üretilen kanıtları varsa planlardan sapmaları adresler. MTG teknolojisi kapsamında, planlardan veya model standartlarından sapıldı ise bunların bu dokümanda adreslenmesi gerekir.

Yazılım başarım özeti dokümanı hazırlandıktan sonra onay için sertifikasyon otoritesine iletilir. Otorite bu dokümanı ve proje boyunca yaptığı gözden geçirme / denetimlerde incelediği verilerin havacılık kurallarına uyduğu konusunda ikna olur ise, yazılımın üretildiği hava aracında kullanılabilirliğini onaylar.

6.18 Sertifikasyon Kapsamında Sorulabilecek Sorular

MTG teknolojisi ile geliştirilecek yazılımların hava araçlarında kullanılabilmesi için yürütülmesi gereken bir sertifikasyon süreci olduğu belirtilmişti. Bu kapsamda otorite tarafından yapılacak denetimlerden önce aşağıda verilen tüm soruların sorularak işletilen sürecin ve üretilen verilerin sorgulanması tavsiye edilmektedir. Benzer sorular sertifikasyon otoritesi tarafından sorulacağı için bu denetimlere hazırlık yapabilmek ve varsa hataları otoriteden önce tespit edebilmek için sorular ve kanıtlar üzerinden yapılacak çalışma son derece faydalıdır. Sorular proje ekibi tarafından arttırılabilir veya o proje için uygulanabilir olmayan sorular listeden elenebilir.

Bu sorular kapsamında Çizelge 6.1'de verilen şablon kullanılabilir. Sertifikasyon içeren projelerde aşağıda verilen, DO-331 dokümanında yer alan amaçlar ve varsa otorite tarafından iletilen soru listelerinde yer alan sorular ilgili kanıt dokümanlar adreslenerek yanıtlanmalıdır. Sertifikasyon gözden geçirmesi / denetimi boyunca otorite, bu kanıtlar üzerinden MTG yaşam döngüsü verilerini inceleyecektir.

Çizelge 6.1 Örnek bir soru listesi şablonu

No	Soru	Evet/Hayır	Kanıt/Açıklama
1	Planlarda Çizelge 4.1'de belirtilen MTG yaşam döngülerinden hangisinin seçileceği belirtilmiş mi?		
2	Tanımlama modeli ve tasarım modeli kapsamında, projeye özel üretilecek veriler planlardan adreslenmiş mi?		
3	Model standardı hazırlanmış ve gözden geçirilmiş mi?		

MTG teknolojisinin sağlıklı bir şekilde işletilerek kullanabilmek için sorulabilecek sorulardan bazı örnekler aşağıda verilmiştir.

- Planlarda Çizelge 4.1'de belirtilen MTG yaşam döngülerinden hangisinin seçileceği belirtilmiş mi?

- Tanımlama modeli ve tasarım modeli kapsamında, projeye özel üretilecek veriler planlardan adreslenmiş mi?

- Model standardı hazırlanmış ve gözden geçirilmiş mi?

- Model standardında model kalite özellikleri tanımlı mı? Model bu özelliklere göre geliştirilmiş mi?

- Model gözden geçirmesinde kullanılacak soru listesi yazılım doğrulama planından adreslenmiş mi?

- Model, yazılım ve sistem mühendislerinin katıldığı tecrübeli bir ekip ile yapılmış mı? Model gözden geçirmeleri için ne kadar zaman ayrılmış?

- Otomatik kod üretimi dışında, bu kodun dış dünya ile ara yüzünü sağlamak üzere el ile kod üretilecek mi? Üretilecek ise bu kod nasıl doğrulanacak? Gözden geçirme sürecinde kalite sorumlusu hangi aktiviteleri gerçekleştirmiş?

- MTG teknolojisi kapsamında sistem seviyesinde yapılacak aktiviteler ve uyum sağlanacak standartlar (örneğin ARP 4754, DO-178C gibi), sistem planlarından adreslenmiş mi?

- Türetilmiş model elemanı var mı? Etiketlenerek emniyet değerlendirme sürecine iletilmiş mi?

- Model gözden geçirmeye girmeden ve gözden geçirmeden sonra konfigürasyon kontrolü altına alınmış mı?

- Model güncellemelerinde, modelin yeni sürümü ile önceki sürümü nasıl karşılaştırılıyor? Bunun için planlarda bir yöntem belirlenmiş mi? Planlarda, modellerde olabilecek fazladan değişikliklerin (hata raporunda olmadığı halde

yazarın yaptığı ilave değişiklik) olmadığından sorumlu rol kim? Bu kapsamda süreçte kontroller yapmış mı?

- Model değişikliklerinin etki analizi nasıl gerçekleştirilmiş? Değişikliği değerlendirmesi gereken tüm roller incelemiş mi?
- Kapsama analizi kriterleri yazılım doğrulama planında verilmiş mi?
- Kurulacak izlenebilirlikler yazılım geliştirme planından adreslenmiş mi?
- Modeller ile üst seviye gereksinimler arasında izlenebilirlik kurulmuş mu?
- Modelin geliştirileceği üst seviye gereksinimler yazılım ve sistem ekibinin katılımı ile belli bir soru listesine göre gözden geçirilmiş mi?
- Model simülasyon durum ve prosedürleri ile üst seviye gereksinimler arasında izlenebilirlik kurulmuş mu?
- Modellerden otomatik kod üretilecek mi? Üretilecek ise otomatik kod üretme aracı kalifiye mi?
- Kod üretirken kullanılan konfigürasyon kayıt altına alınmış mı? Kod üreten aracın ürettiği hata ve uyarılar resmi testlerden önce gideriliyor mu? Kalite sorumlusu bunu kontrol ediyor mu?
- Model bir araçtan başka bir araca aktarılıyor ise modelin davranışından bir değişiklik olup olmadığı kontrol edilmiş mi? Bu kontrol nasıl yapılmış ve hangi kanıtlar üretilmiş? Bu kapsamda s-function ile üretilen model ile orijinal modelin ürettiği veri karşılaştırılacak ise, kabul edilir tolarans değerleri nasıl belirlenmiş ve bu değerler gereksinim olarak tanımlanmış mı?
- Model simülasyonları ile doğrulama yapılacak mı? Simülasyon ortamı ve kullanılacak araçlar sürümleri ile birlikte planlardan adreslenmiş mi?
- Model kapsama analizi yapılmış mı? %100 model kapsaması elde edilmiş mi? Eksik kapsama kapsamında gerekli işlemler tamamlanmış mı?

- Simülasyon prosedürleri modelden direkt üretildi ise, üst seviye gereksinimlere göre gözden geçirilerek gerekli izlenebilirlikler kurulmuş mu?

- Model simülasyonundan bağımsız olarak üst seviye gereksinimlere göre üretilen test prosedürleri ile yazılım testleri yapılmış ve hatalar çözülmüş mü?

- Gözden geçirme ekibi tarafından verilen cevaplar birbiri ile uyumlu mu? Örneğin gözden geçirme soru listesinde yer alan bir soru için (örneğin, karmaşıklık kriteri sağlanmış mı?), gözden geçirme ekibi "Evet", "Hayır", "Uygulanabilir Değil" gibi çelişkili cevaplar verilmiş mi? (verilmiş ise bunun nedeni analiz edilmelidir. Büyük ihtimalle soru herkes tarafından farklı anlaşılmaktadır, soru anlaşılır hale getirilerek soru kapsamında model tekrar incelenmelidir)

- Modelden üretilen ve doğrulanan kod ile yazılım / donanım entegrasyonun yapıldığı ve uçağa yüklenecek kod aynı mı?

- Konfigürasyon durum raporu sistematik bir şekilde tutuluyor mu? Modelin sürümü, modeli üretildiği üst seviye gereksinim sürümü, otomatik üretilen kod sürümü ve simülasyon test durum ve prosedürlerinin uyumlu sürümü bu raporda sistematik bir şekilde kayıt altına alınıyor mu?

- Model simülasyonlarından önce, simülasyon ortamının planlara uygun olduğu kalite sorumlusu tarafından kontrol edilmiş mi?

- Süreçler arasındaki geçişlerin, planlarda tanımlanan geçiş kriterlerine göre yapıldığı kalite sorumlusu tarafından kontrol edilmiş mi?

- Model standartlından sapma var mı? Var ise bu sapmayı takım lideri ve kalite onaylamış mı?

- Model simülasyonlarında, yazılım testlerinde veya gözden geçirmelerde açılan hatalar veya kalan test adımlarının tamamı kapatılmış mı?

- Yazılım ekibinin kapsama analizinde tespit ettiği eksik kapsamaların tamamı sistem ekibine iletilmiş mi? Eksik kapsama giderilerek %100 MKA ve %100 YKA elde edilmiş ve doğrulama sonuçları raporunda adreslenmiş mi?

6.19 Model Gözden Geçirmelerinde Kullanılabilecek Sorular

MTG teknolojisinde geliştirilen modellerin geliştirme ekibi tarafında gözden geçirildiği belirtilmişti. Bu gözden geçirme kapsamında, model standardında yer alan kurallara ve üst seviye gereksinimlere uyum başta olmak üzere dikkate alınması gereken pek çok husus vardır. Bu kapsamda kullanılabilecek sorulardan aşağıda örnekler verilmiştir. Bu sorular proje ekibi tarafından arttırılabilir veya o proje için uygulanabilir olmayan sorular listeden elenebilir. Burada da, Çizelge 6.1'de verilen çizelge yapısı kullanılabilir.

- Model, üst seviye gereksinimler ile uyumlu mu, üst seviye gereksinimleri tam olarak yansıtıyor mu?

- Model ile üst seviye gereksinimler arasında izlenebilirlikler kurulmuş mu?

- Model, model standardına uygun mu? (yukarıda da belirtildiği gibi, bu soruya ilave olarak, standardın içinden en azından kritik olan kuralların, hataya açık konuların model gözden geçirme soru listesine ayrı ayrı yazılması gerekir)

- Geliştirilen modellerin yazılım konfigürasyon planına göre verilmiş tekil bir numaraları var mı?

- Model yazılım konfigürasyon planında belirtilen sürece göre tekil bir numara almış ve gözden geçirmeden önce konfigürasyon kontrolü altına alınmış mı?

- Modelin sürüm bilgisi mevcut mu? Gözden geçirme ekibi ile paylaşılmış mı?

- Model standardında model kalite özellikleri tanımlı mı?

- Geliştirilen modeller kalite özelliklerini sağlıyor mu? Sapma var mı?

- Model gözden geçirme girdileri (model gözden geçirme soru listesi, sistem gereksinimleri ve varsa diğer dokümanlar) sürümleri ve adres bilgileri ile birlikte gözden geçirme ekibi ile en başta paylaşılmış mı?

- Gözden geçirme ekibi sistem ve yazılım mühendislerinden oluşan, konu ile ilgili alan uzmanlarından ve tecrübeli mühendislerden oluşuyor mu?

- Model gözden geçirme ekibinin tamamı model gözden geçirme soru listelerini bireysel olarak doldurmuş mu?

- Gözden geçirme ekibi tarafından verilen cevaplar birbiri ile uyumlu mu? Örneğin gözden geçirme soru listesinde yer alan bir soru için (örneğin, karmaşıklık kriteri sağlanmış mı?), gözden geçirme ekibi “Evet”, “Hayır”, “Uygulanabilir Değil” gibi çelişkili cevaplar verilmiş mi? (verilmiş ise bunun nedeni analiz edilmelidir. Büyük ihtimalle soru herkes tarafından farklı anlaşılmaktadır, soru anlaşılır hale getirilerek soru kapsamında model tekrar incelenmelidir)

- Model karmaşıklık kriterleri açısından değerlendirilmiş mi?

- Modele gözden geçirme soru listesine projenin ilerleyen fazlarında yeni bir soru eklendi ise, bu soru kapsamında yayımlanmış modeller tekrar incelenmiş mi?

- Model gözden geçirme aktivitesi için yeterli süre ayrılmış mı?

- Verilen görüşler kapatılmış mı? Sonraki faza ertelenen görüşler için değişiklik isteği açılmış mı?

- Türetilmiş model elemanı var mı? Model standardına göre etiketlenmiş mi? Emniyet sürecine iletilmiş mi?

- Koda dönüşmesi beklenmeyen model elemanları (yorum blokları gibi) var mı? Model standardına göre etiketlenmiş mi?

- Modelden, gözden geçirme sürecinde verilen görüşler dışında bir değişiklik var mı? Böyle bir değişiklik olmadığını hangi rol hangi yöntem ile incelemiştir? Kanıtlar nelerdir?

- Model gözden geçirme ve görüşlere göre güncelleme süreci tamamlandığında modelin yeni sürümü konfigürasyon kontrolü altına alınmış mı?

- Model geliştirme sürecinde, model geliştirici tarafından sistem gereksinimlerinde bir hata / eksik tespit edilmiş mi? Edildi ise bunlar sistem ekibine hata raporu açarak resmi bir şekilde iletildi mi?

- Model gözden geçirme sürecinde, gözden geçirme ekibi tarafından sistem gereksinimlerinde bir hata / eksik tespit edilmiş mi? Edildi ise bunlar sistem ekibine hata raporu açarak resmi bir şekilde iletildi mi?

- Model geliştirme ve modelden otomatik kod üretme farklı araçlarda yapılacak ise, geliştirilen model otomatik kod geliştirme aracına aktarılırken sorun yaşanmaması için model geliştirme aracında belirlenmiş bir "kullanılabilecek model elemanları" listesi var mı? var ise, bu liste dışında bir model elemanı kullanılmış mı? (Örneğin Simulink aracından SCADE aracına modeller aktarılırken, Simulink'te kullanılan ve SCADE'e aktarım sırasında bozulan model elemanları var ise, bunların Simulink modelinde de kullanılmıyor olması gerekir.)

- Modelin gözden geçirme sürecinden sonraki sürümü yazılım konfigürasyon planına uygun şekilde yayımlanmış mı? Modelin sürümü ile modelin uyumlu olduğu modelin geliştirildiği üst seviye gereksinimler / tanımlama modelinin sürümü bir arada kayıt altına alınmış mı ve konfigürasyon durum raporuna eklenmiş mi?

- Modelin yeni bir sürümü yayımlandığında, ilgili model simülasyon adımları da güncellenmiş mi? Modelin sürümü ile uyumlu olduğu simülasyon durum ve prosedürleri bir arada kayıt altına alınmış mı?

- Kalite sorumlusu sürecin, planlara uygun olarak yürütülmesini teminat altına almak için sürece dahil olmuş mu ve konfigürasyon durum raporuna eklenmiş mi?

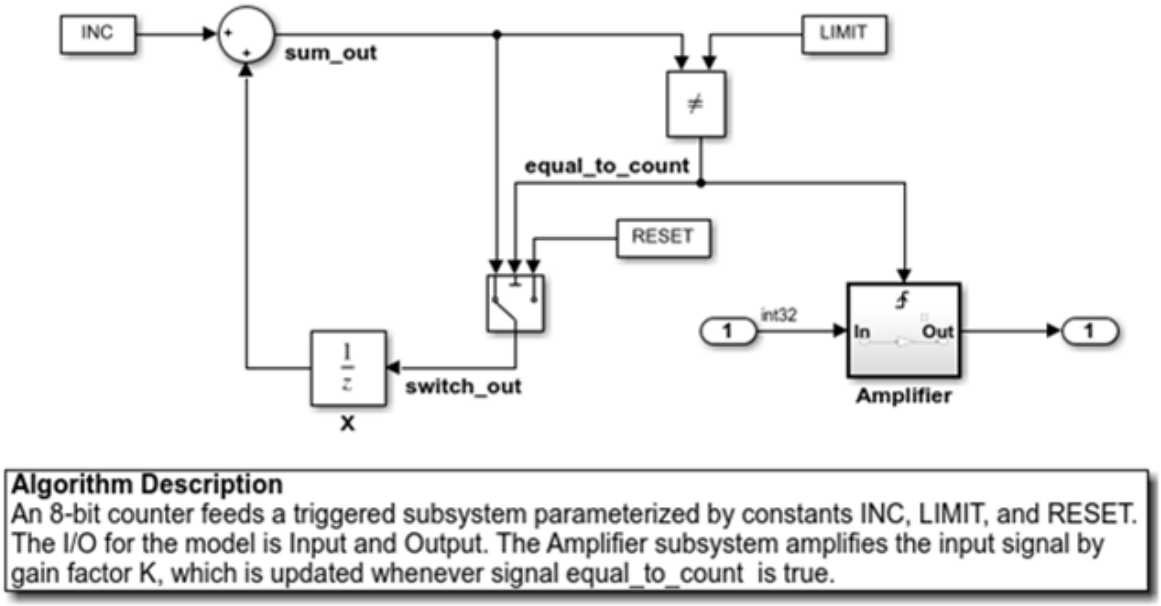
7 UYGULAMA ÖRNEĞİ

Bu bölümde, yukarıda anlatılan konulardan örnek ile anlatılmasının daha faydalı olacağı değerlendirilen konular için örnekler verilmiştir. Bu örnekler aşağıdaki üç konu ile ilgili örnekleri içermektedir:

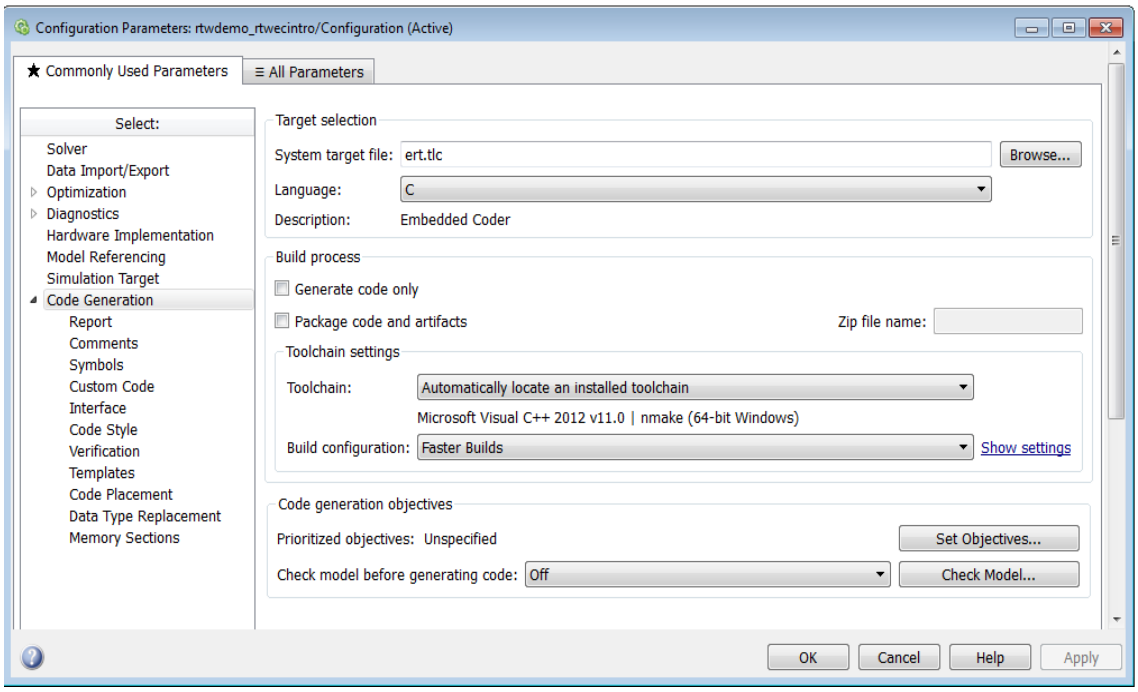
- Modelden otomatik kod üretme örneği
- Yapısal Model Kapsama Analizi ve Yazılım Kapsama Analizi Karşılaştırması
- Kalifiye olmayan bir araçtan, kalifiye olan bir araca transfer edilen bir modelin davranışındaki değişikliğin kontrolü

7.1 Modelden Otomatik Kod Üretme Örneği

MTG teknolojisi kullanımında modellerden otomatik kod geliştirildiği belirtilmişti. Şekil 7.1'de, 8 haneli ve belli bir limite kadar bir k katsayısı ile çarpılarak arttırılan bir sayaç tasarımına ait bir Simulink modeli görülmektedir [27].



Şekil 7.1 Sayaç tasarım modeli [27]



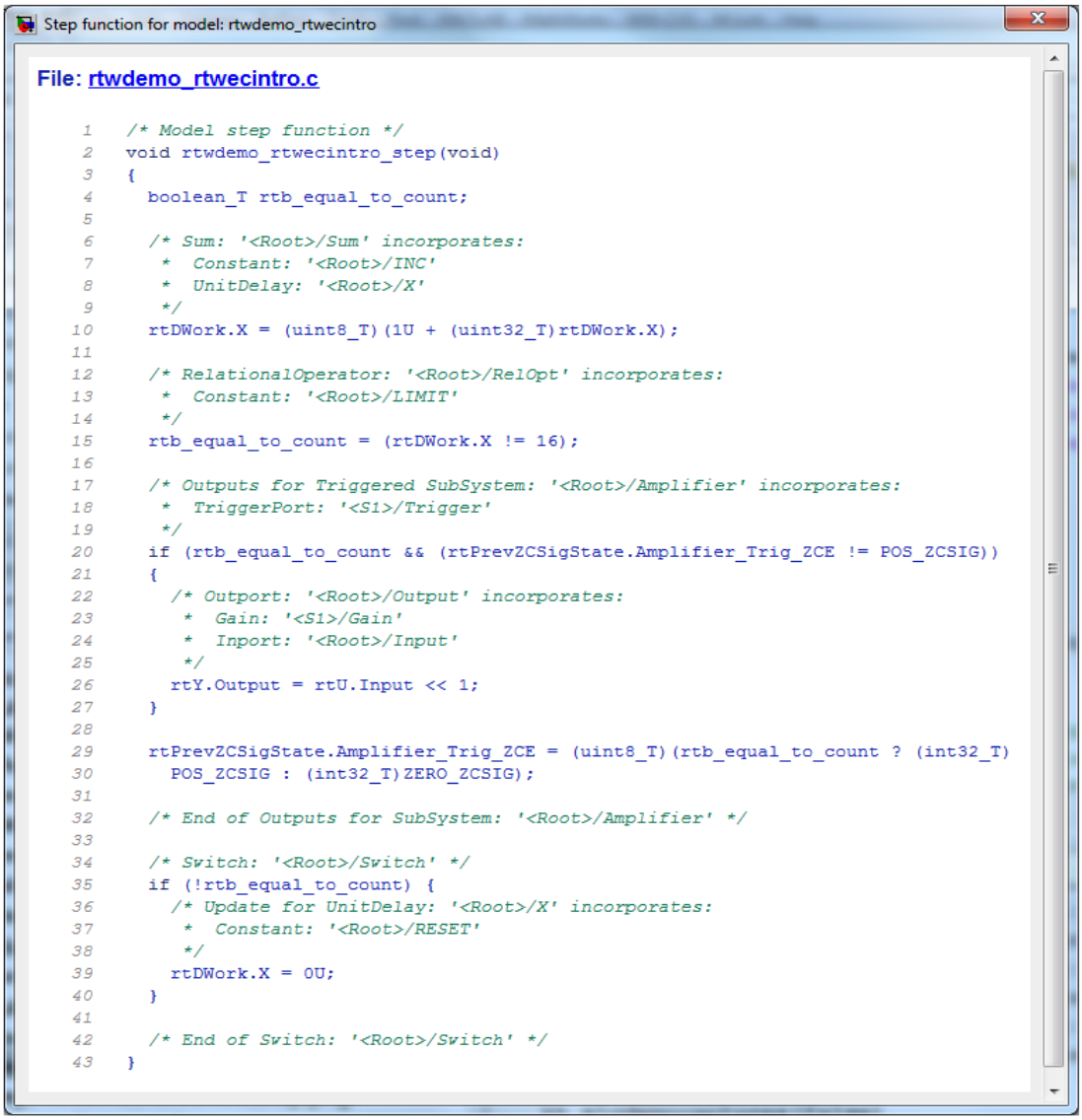
Şekil 7.2 Modelden kod üretme ekranı [27]

Model Simulink aracında çizildikten sonra Şekil 7.2'de görülen ekran üzerinden seçilen dilde otomatik kod üretimi yapılabilir.

7.2 Model Kapsama Analizi ve Yazılım Kapsama Analizi Karşılaştırması

Bu örnekte yapısal kapsama analizi ile model kapsama analizi bir örnek üzerinden karşılaştırılmıştır. Karşılaştırmaya geçmeden önce her iki analizin amacı ve içeriği aşağıda tekrar özetlenmiştir.

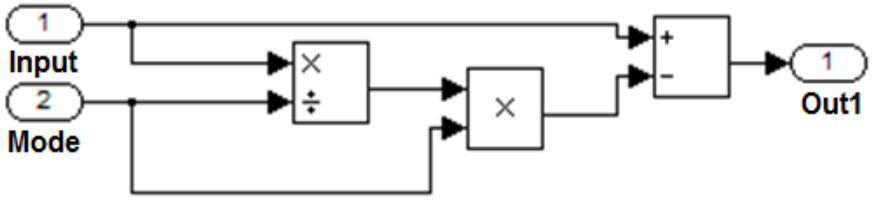
Yazılım test koşularından sonra kod içinde çalışmayan kod satırı olup olmadığını tespit edebilmek için kaynak kod üzerinde yapılan analize yapısal kapsama analizi denir. Eksik kapsama, gereksinim eksikliğinden, test prosedürünün eksikliğinden ve / veya kaynak kod içinde aktive edilmemiş kod, savunmacı programlama veya ölü kod varlığından kaynaklanabilir. Model kapsama analizi (MKA) ise, modelin geliştirildiği gereksinimlerine göre hazırlanan test prosedürleri ile yapılan simülasyon veya testler sonucunda, çalıştırılmayan model elemanlarına ait gereksinimlerin olup olmadığını tespit etmek için yapılır.



Şekil 7.3 Modelden otomatik üretilen kod [27]

Şekil 7.3'te, yukarıda verilen modelden otomatik olarak üretilmiş kod görülmektedir. Hava aracı yazılımı geliştirilecek ise, her iki kapsamanın da %100 olması gerekmektedir.

Şekil 7.4'de modelde 1. girdi olarak verilen değer modu alınacak sayı, 2. girdi ise mod değerini içermektedir. Verilen Simulink modeli, yalnızca mod alma fonksiyonunu içermekte ve yalnızca beklendiği gibi mod fonksiyonu için gerekli olan iki girdi ile beslemektedir. Bir başka deyişle, modelde ne eksik ne de fazla bir fonksiyon bulunmamaktadır. Bu model, Çizelge 7.1'de verilen test seti ile doğrulandığında %100 kapsama elde edilir.



Şekil 7.4 Mod fonksiyonuna ait Simulink modeli

YKA'nın kod üzerinden yapıldığı önceki bölülerde söylenmişti. Şekil 7.5'de mod alma fonksiyonuna ait iki farklı kod uygulaması görülmektedir.

Şekil 7.5'in alt bölümünde yer alan 2. uygulama (implementation 2), Çizelge 7.1'de verilen test girdileri ile test edildiğinde tüm kod satırları çalışacaktır. 1. Uygulamada (implementation 1) ise, modu alınacak sayı ile mod değeri sıfır kalanını verdiği sürece, uygulamanın "else" bloğu hiç bir zaman çalışmayacaktır. Çizelge 7.1'de yer alan test girdileri sıfır kalanı verdiği için, bu test seti ile yapılan testte "else" bloğu hiç bir zaman çalışmaz.

Koşturulan yazılım testi ile kodun her satırı test edilememiş olduğu için eksik yapısal kapsama oluşmuş olur. Kapsanamayan, dolayısı ile test edilemeyen kod bloğunda bir hata var ise bu hata tespit edilememiş olur. Böylelikle hava aracına, test edilemediği için fark edilemeyen ve dolayısıyla hata içeren bir yazılım yüklenmiş olur. Bu durum da yazılımın beklenen fonksiyonunu yapmamasına ve ölümlü kazalara yol açabilir.

MKA ve YKA analizini farkını basitçe anlatabilmek için kolay bir örnek seçilmiştir. Bu örnekte bu test girdilerinin "else" bloğunu çalıştırmayacağını görmek elbette çok kolaydır. Fakat giderek karmaşıklaşan hava aracı sistemlerinde test girdileri yetersiz olduğu için test edilemeyen bir bloğu göz ile görmek mümkün değildir.

Çizelge 7.1 Mod fonksiyonu için test seti

Girdi	Modu Alınacak Değer
20	5
18	3
16	4

```
/* function calculates mode - implementation 1*/
unsigned int func_mode(unsigned int dividend, unsigned int divisior)
{
   unsigned int tempData = 0; /* define a template data for division 1A */

   tempData = dividend / divisior; /* find division 1B */

   if (dividend == divisior * tempData) /* compare dividend and division 1C */
   {
      return 0; /* return 0 if dividend and division are equal 1D */
   }

   else
   {
      tempData = dividend - divisior * (dividend / divisior); /* calculate remainder 1E */
      return tempData; /* return remainder 1F */
   }
}

/* function calculates mode - implementation 2 */
unsigned int func_mode(unsigned int dividend, unsigned int divisior)
{
   return dividend % divisior; /* return mode 2A */
}
```

Şekil 7.5 Mod fonksiyonuna ait iki farklı kodlama

Bu örnekten de görülebileceği gibi %100 model kapsaması sağlana bile %100 yapısal kapsama sağlanamayabilir. Yapısal kapsama kodun yapısına ve verilen test setine bağlı olarak değişebilir.

YKA ve MKA aktiviteleri geliştirilen yazılıma ve yazılıma atanan seviyeye göre değişmekle beraber uzun süreli aktivitelerdir. Havacılık sektöründeki yoğun takvim ve maliyet baskısı ile üreticilerin bu analizlerden yalnızca birini yapmak gibi bir eğilimleri olabilmektedir. Bu durum, hatalı kodun hava aracına yüklenmesine ve uçuş emniyetini etkileyebilecek sonuçlara dahi neden olabilir.

7.3 Model Davranışındaki Değişikliğin Kontrolü

Uygulama örneğinde aynı zamanda kalifiye olmayan bir araçtan (Simulink) kalifiye olan bir araca (SCADE) aktarılan modelin, aktarım esnasında davranışında bir değişiklik olup olmadığını görmek üzere kullanılan bir yöntem örnek üzerinden açıklanmıştır.

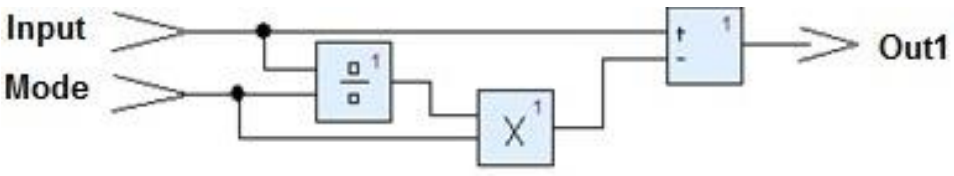
Uygulama örneği kapsamında, Şekil 7.6'da yer alan Simulink modeli, otomatik kod üreten SCADE aracına aktarılmıştır. SCADE aracı, havacılık yazılımlarında

modelden otomatik kod üretmek üzere kullanılan ve ürettiği kod sertifikasyon otoritesi tarafından kabul edilen kalifiye bir araçtır. Bu örnekte model, Simulink aracından özel bir ara yüz ile SCADE aracına otomatik kod üretilmek üzere aktarılmıştır.

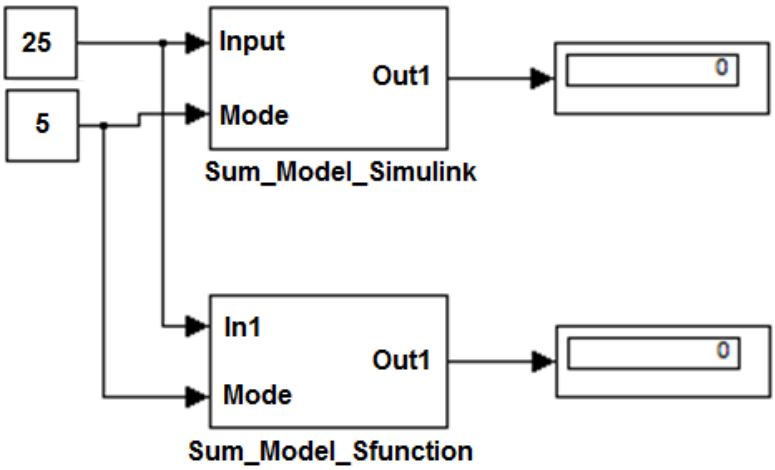
SCADE aracı ile üretilen kod S-function ile tekrar Simulink bloğuna dönüştürülmüştür. S-function (system function), MATLAB, C, C++ veya Fortran dilinde yazılan kodun Simulink bloğuna çeviren bir uygulamadır. Şekil 7.7'de görüldüğü gibi, dönüştürülen model ile orijinal model aynı girdiler ile beslenerek sonuçlar karşılaştırılmıştır. Her iki modelde belirlenen aynı sonucu üretti ise veya üretilen sonuçların farkı belirlenen tolerans değerler arasında ise, aktarım esnasında modelin bozulmadığı söylenebilir.

MKA ve YKA karşılaştırması için başka bir örnek daha verilmiştir. Bir hava aracındaki en önemli fonksiyonlardan birisi, pilota hava aracının yön ve durum bilgisini gösterme fonksiyondur. Kritik fonksiyonlar genellikle olası cihaz arızası, sensör arızası gibi arızalara karşı sistem seviyesinde yedeklenir. Bu yedeklemeye örnek olarak, aynı veriyi sağlayan farklı cihazlar ve bunların sağladığı verilerin başka bir yazılım tarafından karşılaştırılması, oylanması veya ortalamasının alınması gibi farklı tasarım çözümleri verilebilir.

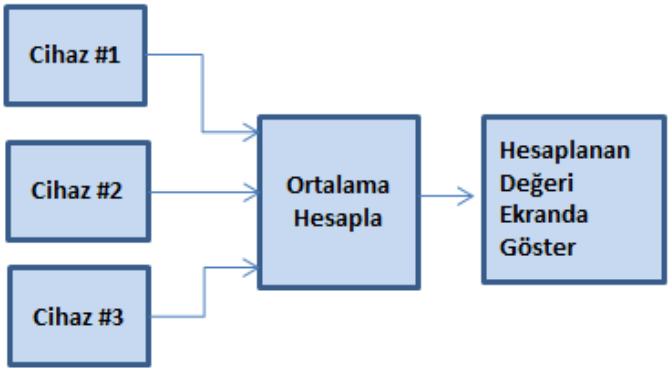
Bu örnekte, hava aracının hız bilgisini sağlayan 3 farklı cihaz olduğu ve bu cihazlardan gelen ve limit içinde olan hız verisinin ortalamasının alınarak hava aracının hızının hesaplandığı bir tasarım düşünülmüştür. Bu tasarıma ait model, modelin girdileri ve çıktıları Şekil 7.8'e benzer bir şekilde olacaktır.



Şekil 7.6 Mod fonksiyonuna ait SCADE'e transfer edilen model



Şekil 7.7 Orijinal model ile üretilen modelin karşılaştırması



Şekil 7.8 Üç cihazın ürettiği değerlerin ortalamasını alan model

Modele ait yazılı gereksinimlerden birisinin aşağıdaki gibi olduğu varsayılmaktadır.

Aşağıdaki koşullar sağlandığında;

a. Cihaz #1 geçerli bir veri sağladı ise

b. Cihaz #2 geçerli bir veri sağladı ise

c. Cihaz #3 geçerli bir veri sağladı ise

aşağıdaki işlemler yapılacaktır;

a. PFD_VALUE değerine, geçerli veri sağlayan cihazların sağladığı verilerin ortalamasını ata

b. Hesaplanan PFD_VALUE değerini ekranda göster

Çizelge 7.2 Ortalama alma fonksiyonu için test seti

Test Durumları	1.Cihaz	2.Cihaz	3. Cihaz
Test Durumu #1	50	100	0
Test Durumu #2	100	50	100
Test Durumu #3	0	0	50

Bu cihazın [0,100] değerleri arasında veri gönderdiği ve bu aralıkta sağlanan verilerin "geçerli", bu aralık dışında sağlanan verilerin ise, (cihazın sensörünün bozulması gibi nedenlerden dolayı) "geçersiz" olduğu varsayılmıştır. Şekil 7.8'de verilen çizimin, gereksinimde belirtilen fonksiyonu, girdi ve çıktıları tam olarak karşıladığı, eksik veya fazla bir fonksiyon / girdi olmadığı görülmektedir. Model, aşağıdaki "geçerli" aralıkta verilen girdiler ile doğrulandığında da tüm model elemanlarının ve fonksiyonlarının çalışarak verilen gereksinimin doğrulandığı, yani model kapsamasının %100 olarak sağlandığı görülür.

Bununla birlikte bu modele ait yazılım geliştirici tarafından Şekil 7.9'da görülen aşağıdaki kod yazılmış olsun. Kod incelendiğinde "else" bölümünün, sensörlerin tamamından geçerli bir değer gelmediği durumlar için hız değerinin sıfır olarak atandığı görülmektedir. Bu kod, yazılım mühendisi tarafından sensörlerden geçerli bir veri gelmediği senaryo durumunda yazılımın davranışını tanımlamak üzere eklenmiş bir kod olup, bu davranış gereksinimde ve modelde tanımlı değildir.

Gereksinimlerde bu davranış tanımlı olmadığından, gereksinim bazlı yapılan testlerde de bu senaryoya yönelik test de olmaz. Dolayısıyla Çizelge 7.2' de verilen test girdileri ile yapılan testler ile "else" bloğu test edilememiş, yani eksik kapsama oluşmuş olur. Else bloğu yazılım geliştirici tarafından bir eksik davranışı tanımlamak için yazılmış olsa bile, bu davranışın sistem ve emniyete etkisini yazılım geliştirici bilemeyebilir ve uçuş emniyeti olumsuz yönde etkileyebilecek bir davranış tanımlayabilir.

Bu senaryoda, tanımlanan davranış doğru olsa bile else bloğunda kodda bir hata da olabilir. Bu durumda hava aracına, hatalı bir kod yüklenmiş olur. Bu tip eksikler görülebileceği gibi MKA ile değil YKA ile tespit edilebilmektedir.

Örnekteki süreç devam ettirildiğinde, YKA'da bu bloğun çalışmadığı tespit edilmiş olacaktır. Yapılan analizde, sistemin böyle bir senaryoda davranışının tanımlı olmadığı ortaya çıkmış olacak ve bu eksiklik sistem ekibine iletilecektir. Sistem ekibi cihazların geçersiz veri göndermesi durumunda sistem davranışını tanımlayan yeni gereksinimler ekleyerek modeli güncelleyecektir. Yeni eklenen gereksinimlere göre yeni testler oluşturulacak ve böylelikle verilerin geçerli olmadığı durumlarda yazılımın davranışı test edilebilecektir.

Bu örnekte ideal senaryo şu olurdu. Sistem sorumlusunun gereksinimleri tanımlarken, aralık dışı değer gelmesi durumunda sistemin davranışını tanımlamalı idi. Bu tanımlama unutuldu ise, bu eksikliğin gereksinim ve / veya model gözden geçirmelerinde tespit edilmesi beklenirdi. Hata o aşamalarda da tespit edilemedi ve yazılım geliştirici tarafından tespit edildi ise (karmaşık sistemlerde bu durum mümkün olabilmektedir) yazılım geliştiricinin sistem davranışını tanımlayan kodlamayı yapmadan önce, modelde bu davranışın tanımlı olmadığını ifade eden hata raporu açması gerekirdi.

```
{
   double calcValue = 0.0f;
   if (validity1 || validity2 || validity3)
   {
      calcValue = (value1 * validity1 + value2 * validity2 + value3 * validity3) /
(validity1 + validity2 + validity3);
      *value = calcValue;
      return true;
   }
      else
   {
      *value = calcValue;
      return false;
   }
}
```

Şekil 7.9 Ortalama alan kod örneği

8 SONUÇ VE ÖNERİLER

Bu çalışmada son yıllarda oldukça popüler olan MTG teknolojisinin hava araçlarında kullanımı ve sertifikasyon sürecini etkileyebilecek konular örnekler ve çözüm önerileri örnekler üzerinden açıklanmıştır.

MTG teknolojisi kullanımı ile sistem seviyesinde modeller geliştirilmekte ve henüz kod dahi yazılmadan simülasyonlar ile model doğrulaması yapılabilmektedir. Bu sayede, modeldeki hata veya olası performans iyileştirmeleri projenin erken fazında tespit edilebilmektedir.

Bu teknoloji ile geliştirilen modellerden kalifiye kod üretme aracı ile otomatik kod üretilmekte ve hava araçlarında kullanılarak otorite tarafından sertifiye edilebilmektedir. Bu sayede, geleneksel yazılım geliştirme yaşam döngüsünde üretilen üst seviye yazılım gereksinimi, alt seviye yazılım gereksinimi, yazılım tasarımı oluşturulmadan ve el ile kodlama yapılmadan kod üretilmiş olmaktadır. Böylelikle bu yazılım yaşam döngüsü verilerinin hazırlanması ve gözden geçirilmesi için bir iş gücü planlaması olmayacağı için bu teknoloji üreticilere son derece önemli bir takvim ve maliyet avantajı sağlamaktadır.

Bununla beraber, MTG teknolojisinin kullanımı ile ilgili dikkat edilecek pek çok husus vardır. Bu hususlardan birisi de, otomatik kod üreten araçların kalifiye olması gerekir. Kalifiye araçların kısıtlı bir kütüphanesi olduğu için sistem geliştirme için daha esnek olunan başka araçlar tercih edilmektedir. Bunun sonucu olarak model geliştirilen araç ile otomatik kod üreten araçlar genellikle birbirinden farklı olmaktadır. Bu durumda beraberinde, simülasyon ve gözden geçirmelerle doğrulanan modelin otomatik kod geliştirecek araca aktarımında modelin davranışının değişme olasılığını getirmektedir. Birbirinden farklı iki araç arasındaki veri akışı esnasında, başta araçların kütüphanelerinin farklı olması gibi nedenlerden veri (burada model) bozulabilir. Bu tip durumlarda modelin davranışının aktarım esnasında bozulmadığını ispat eden bir yöntem kullanılmalıdır. Bu çalışmanın uygulama bölümünde bu kontrol kapsamında kullanılan bir yöntem verilmiştir. Bu yönteme göre, Simulink aracında model çizilmiş ve doğrulandıktan sonra otomatik kod üretmek üzere kalifiye bir araç olan SCADE aracına aktarılmıştır. SCADE aracında bu modelden otomatik kod

üretilmiş ve üretilen kod s-function dosyası olarak tekrar bir Simulink bloğuna dönüştürülmüştür. Orijinal model ile geri dönüşüm ile oluşan model aynı girdiler ile beslenerek elde edilen sonuçlar karşılaştırılmıştır. Sonuçlar belli bir toleransın içinde ise, modelin aktarım esnasında davranışının değişmediği sonucuna varılmaktadır. Burada toleransın nasıl tanımlanacağı konusu kritiktir. Örneğin, orijinal model ile koddan elde edilen model karşılaştırıldığında hücum açısı (Angle of Attack) değerinde bir sapma olduğu varsayılsın. Hücum açısı, hava aracı için son derece kritik bir değerdir. Hücum açısı küçük düz uçuş yapan ve küçük bir aralıktan hareket eden uçak için 1-2 derecelik fark uçağı düşürebilecek etki yaratabilirken, çok geniş bir aralıkta hareket eden çok daha hızlı olan bir savaş uçağında çok önemli olmayabilir. Bu nedenle, iki model karşılaştırıldığında ortaya çıkan sapmanın uçuş emniyetine etkisinin çok iyi analiz edilmesi gerekir. Her ne kadar bu araçlar kullanılarak yapılan dönüşümlerde sapmalar ihmal edilebilir derecede düşük çıksa bile, kabul edilebilir sapma (tolerans) sistem ve emniyet ekibi ile birlikte detaylı bir şekilde değerlendirilmelidir.

MTG teknolojisinde sistem ve yazılım ekiplerinin birbiri ile koordineli çalışması ve verilerdeki (örneğin modeldeki) olası değişiklikten yazılım ekibini nasıl haberdar edecekleri, yazılım ekibinin model ile ve modeli doğrulamak için kullanılan test prosedürlerine yorumlarının sistem ekibine nasıl iletileceği gibi geri dönüş gereken tüm aşamaların tanımlanması gerekir. İlave olarak, MTG teknolojisinde sistem ekibinin de DO-178C amaçlarına uyması gereken MTG yaşam döngüleri mevcuttur. Bu teknoloji yeni olduğu için, sistem ekibinin havacılık kuralı olan DO-178C rehber dokümanı ile ilgili tecrübesinin yazılım ekibi kadar fazla olmadığı sektörü deneyimleri ile sabittir. O nedenle sistem ekibinin sistem seviyesinde geliştirdiği verilerin DO-178C uyum gösterim sürecinde sorunlar yaşanabilir. Bu sorunları önlemek için proje başında ve belli aralıklarla yazılım ekibinin, sistem ekibine mümkünse bitmiş proje örnekleri üzerinden uygulamalı eğitim vermeleri önerilmektedir.

MTG teknolojisi ile yazılım geliştirilecek ise, yukarıda anlatılan bölümlerdeki tüm konuların dikkate alınarak yazılım ve sistem planları yazılmalıdır. Bu planlar üzerinde sertifikasyon otorite ile projenin başında hemfikir olmak çok elzemdir.

Aksi halde, yaşam döngüsü ilerlemişken sertifikasyon otoritesinin yapılmasını istediği ilave çalışmalar projenin takvim ve maliyetini arttırabilir.

MTG teknolojisinde modelden kalifiye araçlar ile simülasyon durum ve prosedürlerinin de oluşturulabileceği belirtilmişti. Modelden hem kod hem de simülasyon test durum ve prosedürlerinin otomatik olarak üretilmesi, modeldeki bir hatanın tespit edilmemesi ve hava aracına hatalı kodun yüklenmesine neden olabilir. Böyle bir yaklaşım izlenecek ise, otomatik olarak üretilen simülasyon durum ve prosedürleri ile üst seviye gereksinimler arasında izlenebilirlik kurulmalıdır. Böylelikle model gözden geçirmede yakalanamayan bir hata var ise, üst seviye gereksinimlere göre geliştirildiği doğrulanmış simülasyon prosedürü ile tespit edilebilir.

Bu teknolojide, geleneksel yazılım yaşam döngüsünde yapılan gözden geçirmeler (yazılım gereksinim gözden geçirmesi, kod gözden geçirmesi, yazılım tasarım gözden geçirmesi) azalacak veya hiç yapılmayacaktır. Her ne kadar bu durum üreticiler için takvim avantajı sağlıyor gibi görünse de, bu aşamalarda yapılan kontroller yapılmamış olacaktır. MTG teknolojisinde temel gözden geçirme model ve üst seviye gereksinimin uyumu kapsamında yapılan gözden geçirmedir. Bu gözden geçirmeden ve model simülasyonunda sonra kod modelden otomatik üretilmektedir. Bu nedenle, üst seviye gereksinimlerin model ve simülasyon test durum ve prosedürleri ile uyumluluğu kapsamında yapılan gözden geçirme ve kurulan izlenebilirlikler son derece kritiktir. MTG teknolojisi kullanıcılarına, bu gözden geçirme için gerekli zaman ve iş gücünün gerçeği yansıtacak şekilde hesaplanarak proje takvimine eklenmesi tavsiye edilmektedir. İlave olarak, MTG teknolojisinde gözden geçirmeleri azalması ile azalan kontrol noktaları nedeni ile model gözden geçirmelerine yazılım ve sistem mühendisler ve tecrübeli mühendislerin katılması son derece önemlidir.

MTG teknolojisinde modellerin konfigürasyon ve değişiklik yönetimi süreçlerinin tanımlanması, ilgili sorumluların projenin başından itibaren atanması son derece önemlidir. Üst seviye sistem gereksinimleri ile modelin, model ile modelden üretilen kodun ve üst seviye gereksinimler ile model simülasyon durum ve prosedürlerinin uyumlu sürümlerinin sistematik bir şekilde takip edilmesi gerekir.

KAYNAKLAR LİSTESİ

[1] Saraç T., Certification Aspects of Model Based Development for Airborne Software, 2019 IEEE 2nd International Conference on Information and Computer Technologies, Kahului, pp.42-49, 14-17 March 2019.

[2] Erkkinen, T., Conrad, M., Verification, Validation, and Test with Model-Based Design, Commercial Vehicle Engineering Congress & Exhibition 2008, SAE Technical Paper, Chicago, pp.56-62, October 2008.

[3] Internet: Wilfredo Torres-Pomales, December 2014, NASA/TM-2014-218670, Is Model-Based Development a Favorable Approach for Complex and Safety-Critical Computer Systems on Commercial Aircraft? https://ntrs.nasa.gov/search.jsp?R=20150000600, [Accessed: 20- Jun-2019].

[4] Jaw L., and Van, H., Model-based Approach to Validation and Verification of Flight Critical Software, 2008 IEEE Aerospace Conference, Montana, USA, 1-8 March 2008.

[5] Bhatt, D., and Hall, B., Model-Based Development and The Implications To Design Assurance And Certification, 24th Digital Avionics Systems Conferrence, IEEE, Washington DC, USA, 30 Oct.- 3 Nov 2005.

[6] MQO Working Group, Model Quality Objectives, Version 1.0, Matwork, MQO Working Group Munich, pp. 3-12, September 2008.

[7] Eisemann, U., Applying Model-Based Techniques for Aerospace Projects in Accordance with DO-178C, DO-331, and DO-333, HAL Archives, Tolouse, France, pp. 2-6, June 2016.

[8] Jackson M., and Henry, R., Orion Gn&C Model Based Development: Experience and Lessons Learned, American Institute of Aeronautics and Astronautics, Houston, USA, pp. 1-8, August 2012.

[9] Robert G., Pettit IV, Highlighting the Challenges of Model-Based Engineering for Spaceflight Software Systems, 2013 5th Interntional

Workshop on Modelling in Software Engineering, IEEE, San Francisco CA, USA, pp.14-22, May 2013.

[10] Internet: Kramer et al, 2016/2017/Model-based Testing User Survey: Results, http://www.cftl.fr/wp-content/uploads/2017/02/2016-MBT-User-Survey-Results.pdf , 2012, [Accessed: 20- Jun- 2019].

[11] Internet: Department Of Defense Standard Practice, MIL-STD-882E System Safety, https://www.system-safety.org/Documents/MIL-STD-882E.pdf, s.25-35 ,11 May 2012, [Accessed: 20- Jun- 2019].

[12] RTCA, DO-331 Model-Based Development and Verification Supplement to DO-178C and DO-278A, RTCA Washington DC, pp. 36-47, December 2011.

[13] Internet: Global air travel, passenger-kilometers per year, 1936-2016 , Too much tourism_ Global air travel and climate change » Darrin Qualman.html https://www.darrinqualman.com/global-air-travel-climate-change/, [Accessed: 20- Jun- 2019].

[14] Internet: Wright brothers, http://en.wikipedia.org/wiki/Wright_brothers, [Accessed: 20- Jun- 2019].

[15] Internet: Ethiopian Airlines Group, Aircraft Accident Investigation Bureau Preliminary Report, Federal Democratic Republic of Ethiopia, South East of Addis Ababa, Ethiopa, pp. 14-23, March 2019.

[16] Internet: European Commission, Annual Analyses of the EU Air Transport Market, https://ec.europa.eu/transport/sites/transport/files/2016___eu_air_transport_industry_analyses_report.pdf, [Accessed: 20- Jun- 2019].

[17] RTCA, DO-178C Software Considerations in Airborne Systems and Equipment Certification, RTCA Washington DC, s. 56-67, December 2011.

[18] FAA, Airborne Software Development Assurance Using EUROCAE ED-12 and RTCA DO-178, AC No: 20-115D, FAA Washington DC, pp. 12-25, July 2017.

[19] Society of Automotive Engineers, ARP 4761 Guidelines And Methods For Conducting The Safety Assessment Process On Civil Airborne Systems And Equipment, Society of Automotive Engineers, SAE International Press, Washington DC, USA, pp. 45-59, December 1996.

[20] Marques J., Yelisetty S., Dias L., Using Model-Based Development as Software Low-Level Requirements to Achieve Airborne Software, 2012 Ninth International Conference on Information Technology – New Generation, Las Vegas, NV, USA, 16-18 April 2012.

[21] Rierson, L., Developing Safety-Critical Software, s.343-358, CRC Press, Newyork USA, pp. 425-453, 2013.

[22] Internet: Developing The Apollo Lunar Module Digital Autopilot, https://www.mathworks.com/help/simulink/slref/developing-the-apollo-lunar-module-digital-autopilot.html, [Accessed: 20- Jun- 2019].

[23] Society of Automotive Engineers, ARP 4754A Guidelines for Development of Civil Aircraft and Systems, SAE International Press, Washington DC, USA, pp. 31-42, USA, 2010.

[24] Model Quality Objectives for Embedded Software Development with MATLAB/Simulink,https://www.mathworks.com/content/dam/mathworks/white-paper/mqo-paper-v1.0.pdf , [Accessed: April 17, 2018].

[25] RTCA, DO-248C, Supporting Information for DO-178C and DO-278A, RTCA Washington DC, USA, pp.29-32, 13.12.2011.

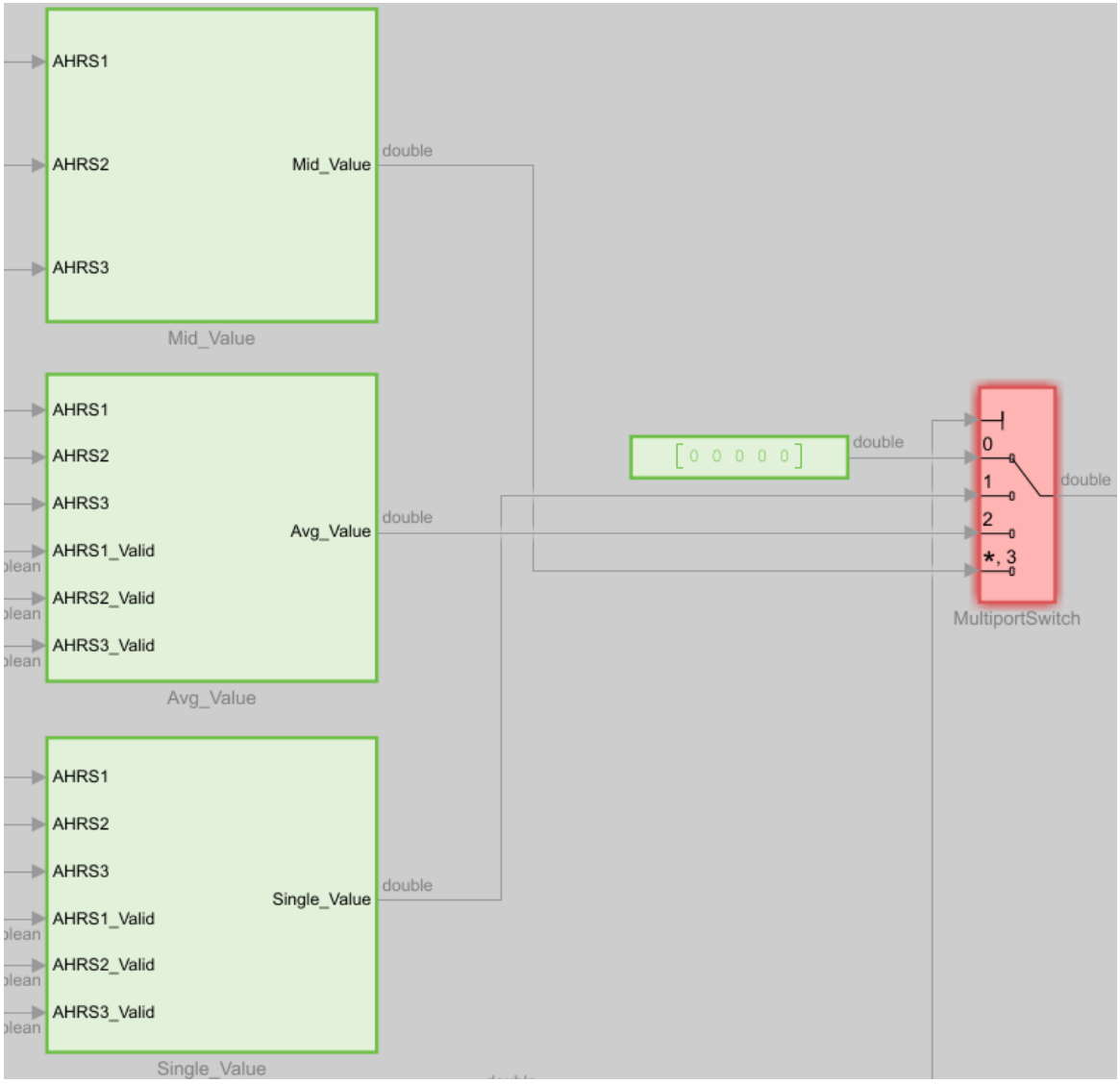
[26] Federal Aviation Administration, Software Approval Guidelines 8110.49, Federal Aviation Administration Oklahoma, USA, s. 12-17, March 2018.

[27] Internet: 1994-2019 The MathWorks, Inc., Generate Code Using Embedded Coder®, https://www.mathworks.com/help/ecoder/examples/generating-code-using-embedded-coder.html. [Accessed: 20- Jun- 2019]

EKLER

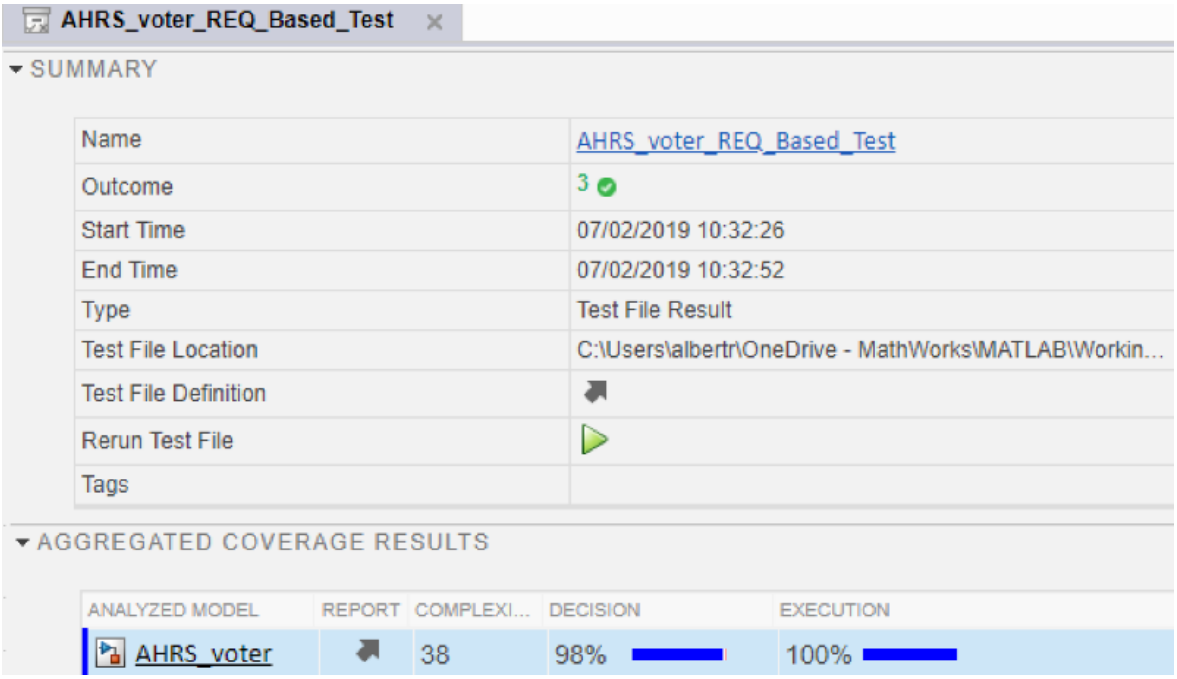
EK 1. Simulink Görüntüleri

Şekil 8.1'de örnekte üç cihazdan gelen verinin ortalamasını alan bir model örneği görülmektedir. Bu modelde türetilmiş model elemanı "Derived" şeklinde etiketlenmiştir. İlave olarak, yapılan model simülasyonlarında eksik model kapsama oluşan model elemanı da kırmızı renkte gösterilmektedir.



Şekil Ek 1.1 Simulink aracında geliştirilen modelden türetilmiş model örneği

Şekil 8.2'de Simulink aracında yapılan simülasyonda elde edilen kapsama oranı %98 olarak görünmektedir. Şekil 8.2'de kırmızı ile görünen model elemanında eksik kapsama olduğu için kapsama oranı %98 olarak elde edilmiştir. Eksik kapsama, her üç cihazın sıfır verisini göndermesi durumunu sistem seviyesinde gereksinim olarak tanımlanmamışken, modele eklenmesi durumunda oluşmuştur.



Şekil Ek 1.2 Simulink aracında oluşan eksik kapama örneği