DO-178C Uyumlu Yazılım Geliştirme Projelerinde Assembly Kodlama İçin Yapısal Kapsama Analizi

Dilara Gizem Pektaş, Mehmet Umut Pişken

Savunma Teknolojileri ve Mühendislik A.Ş., Mühendislik ve Sertifikasyon Müdürlüğü, Ankara, Türkiye

{gpektas, mpisken}@stm.com.tr

Özet. Kod kapsama analizi, temel amacı test faaliyetlerinin yeterliliğini ölçmek ve test aşamasından çıkmak için gerekli kriterlerin sağlanıp sağlanmadığını kontrol etmek olan bir faaliyettir. Emniyet kritik yazılım geliştirme standartlarının birçoğu kod kapsama analizini karşılanması gereken bir amaç olarak tanımlamaktadır. Aviyonik yazılım geliştirmede kullanılan DO-178C standardı da kod kapsama analizini bir amaç olarak belirlemiştir. Ancak DO-178C standardında yer alan kod kapsama analizine ilişkin tanımlara bakıldığında, yüksek seviyeli dillerde bulunan "Boolean" ifadelere atıflar olduğu ve Assembly programlama diline yönelik beklentilerin tanımlanmamış olduğu görülmektedir. Bu sebeple Assembly programlama dilinin kullanılacağı kod kısımları için yapısal kapsama analiz faaliyetinde nelere dikkat edilmesi gerektiği bir problem olarak karşımıza çıkmaktadır. Bu çalışmada, Assembly programlama dili ile geliştirilen kodlarda, DO-178C standardının beklediği kapsama türlerinin karşılanması için yapılması gereken çalışmalar ile karşılaşılabilecek problemlerden bahsedilecek ve bu problemlerin çözümüne ilişkin öneriler verilecektir.

Anahtar Kelimeler: Kod Kapsama Analizi, DO-178C, Assembly Programlama Dili

Structural Coverage Analysis for Assembly Coding in DO-178C Compliant Software Development Projects

Abstract. The key objective of code coverage analysis activity is to measure the adequacy of testing activities and to decide on whether the exit criteria of the testing phase have been met or not. Many of the safety-critical software development standards define code coverage analysis as an objective to be met. The DO-178C standard used for avionics software development projects also defines code coverage analysis as an objective. However, when the description of code coverage analysis defined in DO-178C is investigated, it can be seen that there are references to "Boolean" expressions which are specific to high-level languages, but no expectations are defined for Assembly Programming Language. For this reason, structural coverage analysis considerations for source code which is implemented by using Assembly Programming Language becomes a challenge in DO-178C compliant

safety-critical software development projects. This study aims to determine how to handle structural coverage related objectives defined in DO-178C, for source code implemented by using Assembly Programming Language. Also this paper will refer to the problems that may arise during the structural coverage process and present a proposal for solution of these problems.

Keywords:Code Coverage Analysis, DO-178C, Assembly Programming Language

1 Giriş

Kod kapsama analizi, koşturulan test senaryolarının ilgili yazılım kaynak kodunun ne kadarlık bölümünü çalıştırdığını anlamak üzerine yapılan analizler olarak tanımlanmaktadır. Yazılım dünyasında kimi zaman test faaliyetlerinin yeterliliğini ölçmek veya test aşamasından çıkmak için gerekli kriterlerin sağlanıp sağlanmadığını kontrol etmek için kullanılabilen kod kapsama analizi, emniyet kritik yazılım geliştirme standartlarının birçoğu tarafından(EN50128:2011, IEC62304 gibi) karşılanması gereken bir amaç olarak tanımlanmaktadır. Kod kapsama analizi yapılmasını isteyen emniyet kritik yazılım geliştirme standartlarından birisi de aviyonik yazılım geliştirmede kullanılan DO-178C standardıdır.

DO-178C standardı, kod kapsama analizini "Yapısal Kapsama Analizi (Structural Coverage Analysis)" olarak isimlendirmekte ve aşağıdaki gibi tanımlamaktadır:

- Gereksinim tabanlı testlerin koşturulması sonucunda kodun ne kadarlık bölümünün çalıştırıldığını anlamaya yönelik yapılan analiz [1]

DO-178C standardının beklediği kod kapsama analizi, yazılım dünyasında kullanılan yapısal testler ile karıştırılmamalıdır. Yapısal testlerde amaç, kodun tamamını kapsayacak şekilde testler geliştirilmesi iken, DO-178C standardındaki amaç, gereksinimler temel alınarak hazırlanmış olan test senaryoları aracılığıyla kodun ne kadarlık bölümünün kapsandığını analiz etmektir [2]. Analizin temel amacı, herhangi bir gereksinime karşılık gelmeyen gereksiz kod parçalarını tespit etmek, testlerdeki ve/veya gereksinimlerdeki olası eksiklikleri bulmaktır. Analiz sonuçlarına göre kod, test ve gereksinimlerde gerekli düzeltmeler yapılarak tüm uygunsuzluklar giderilinceye kadar test koşuları ve analizleri devam ettirilmektedir.

Günümüzde, kodlama faaliyetleri genellikle C, C++, Ada gibi yüksek seviyeli programlama dilleri ile yapılmaktadır ve yapısı itibarıyla daha fazla hataya yatkın olan Assembly programlama dili ile programlama oldukça azalmıştır. Ancak aviyonik yazılım geliştirme projelerinde özellikle sürücü(driver) yazılımları gibi donanımla doğrudan irtibatı olan bazı yazılımların geliştirilmesinde az miktarda da olsa Assembly programlama dili ile geliştirme söz konusu olabilmektedir. DO-178C standardının yapısal kapsama türlerine ilişkin tanımlamaları incelendiğinde, yüksek seviyeli programlama dillerinde bulunan "Boolean" gibi ifadelere atıflar olduğu görülmektedir. Ayrıca DO-178C standardında, Assembly programlama dili ile geliştirilmiş yazılımlara yönelik yapısal kapsama analizi amacının nasıl karşılanması gerektiğine ilişkin herhangi bir bilgi mevcut değildir. Bu sebeple Assembly programlama dilinin kullanılacağı kod kısımları için yapısal kapsama analiz faaliyetinde nelere dikkat edilmesi gerektiği bir problem olarak karşımıza çıkmaktadır.

Bu çalışmada, Assembly programlama dili ile geliştirilen kodlarda, DO-178C standardının beklediği kapsama türlerinin karşılanması için yapılması gereken çalışmalar ve karşılaşılabilecek problemlerden bahsedilecektir. Çalışmanın ikinci bölümünde DO-178C standardında bahsi geçen yapısal kapsama analiz türlerinden kısaca bahsedilecektir. Üçüncü bölümde, Assembly programlama dilinde DO-178C standardında geçen kapsama türlerine ilişkin bir örnek üzerinden gidilerek karşılaşılabilecek olası problemler ortaya konulacaktır. Çalışmanın son bölümünde ise sonuç ve önerilere yer verilecektir.

2 Yapısal Kapsama Analizine Genel Bakış

Emniyet kritik yazılım sınıfına giren aviyonik yazılımlarda, sistem seviyesinde gerçekleştirilen emniyet analizleri sonucunda, yazılıma atanan tasarım teminatı seviyesinin gereklerini karşılamak adına DO-178C dokümanı referans olarak kullanılmaktadır. DO-178C, uçuşa elverişlilik sertifikasyonunda, uçuşa elverişlilik gereksinimlerinin kabul edilebilir seviyede karşılanması amacıyla yazılım yaşam döngüsü boyunca karşılanması gereken hedefleri, bu hedefleri karşılamak için gerekli aktiviteleri ve aktivitelerin bağımsızlık seviyesini, sertifikasyona dair kredi alınması için kullanılabilecek kanıtları, bazı durumlarda uygulanabilirliği olan ek hususları tanımlar [1]. Bu sayede yazılımın kabul edilebilir seviyede uçuş emniyeti sağladığı güvence altına alınır.

DO-178C, yazılım yaşam döngüsü sürecini üç ana başlık altında inceler: *yazılım planlama süreci*, *yazılım geliştirme süreci* ve *bütünleşik süreçler*. Bütünleşik süreçler projenin başlaması ile hayata geçen, planlama ve yazılım geliştirme süreçleri ile paralel yürütülen ve proje devam ettiği sürece devam eden doğrulama, kalite ve konfigürasyon aktivitelerinin tamamıdır. Yapısal kapsama analizi de, doğrulama sürecinin bir parçasıdır.

Yapısal kapsama, kod yapısının mevcut gereksinim tabanlı test senaryolarıyla ne kadarının çalıştırılabildiğinin ölçümüdür. Bu ölçümü yapabilmek için, genellikle enstrümantasyon(instrumentation) olarak adlandırılan özel bir yöntem ile kaynak koda, özel kod parçacıkları eklenir. Kaynak kodda yapılan bu değişiklik sayesinde, gereksinim tabanlı testlerin koşumu sonrasında kaynak kodun hangi kısımlarının çalıştırılabildiği kolaylıkla ayrıştırılabilir. Gereksinim tabanlı testlerin tamamı koşulduktan sonra yaklaşık %80 oranında kapsama oranına ulaşılması hedeflenmelidir [3]. Test koşuları sonucunda elde edilen bu ham kapsama oranının, yapılan yapısal kapsama analizleri ile %100'e tamamlanması beklenmektedir.

2.1. DO-178C Açısından Yapısal Kapsama

DO-178C, yazılımın kritiklik seviyesine göre 3 farklı kapsama tipini amaç olarak belirlemektedir[1]:

- **İfade Kapsaması (Statement Coverage):** Tasarım teminatı seviyesi C olan yazılımlar için beklenen kapsama tipidir. Program içerisindeki her ifadenin en az bir kere çalıştırılması beklenmektedir.

- **Karar Kapsaması (Decision Coverage):** Tasarım teminatı seviyesi B olan yazılımlar için beklenen kapsama tipidir. Program içerisindeki her giriş (entry) ve çıkışın (exit) en az bir kere çalıştırılmış ve her karar ifadesinin en az bir kere doğru(true) ve yanlış(false) değerlerini almış olması beklenmektedir.
- **Uyarlanmış Koşul/Karar Kapsaması (Modified Condition/Decision Coverage– MC/DC):** Tasarım teminatı seviyesi A olan yazılımlar için beklenen kapsama tipidir. Diğer iki yönteme göre daha karmaşık bir yapısı vardır:
    - Program içerisindeki her giriş(entry) ve çıkış(exit) en az bir kere çalıştırılmış,
    - Bir karar içerisindeki her koşul en az bir kere doğru(true) ve yanlış (false) değerlerini almış,
    - Her karar en az bir kere doğru(true) ve yanlış(false) değerlerini almış
    - Karar içerisindeki her koşulun birbirinden bağımsız olarak kararı etkilediğinin gösterilmiş olması beklenmektedir.

Tasarım teminatı seviyesi D olan yazılımlar için yapısal kapsama analizi, karşılanması gereken bir amaç olarak beklenmemektedir. Yapısal kapsama analizi, gereksinim tabanlı test senaryolarının tamamı koşturulduktan sonra, kodun çalışmayan kısımlarının neden çalışmadığını çözümleme yöntemidir. Yapısal kapsama analizi ile hedef [1];

- Kod içindeki her ifadenin en az bir kere çalıştırıldığının garanti altına alınması,
- İstenmeyen ve/veya test edilmeyen fonksiyonalitenin belirlenmesi,
- Ölü (dead code) ve/veya yabancı (extraneous code) kodun belirlenmesi,
- Etkin olmayan kod mekanizmasının (deactivation mechanism) doğru bir şekilde çalıştığının gösterilmesi,
- Hatalı mantık uygulamalarının belirlenmesidir.

DO-178C, enstrümante edilmiş kod ile gerçekleştirilen test koşumu sonrasında kodun çalıştırılamayan kısımlarının analizi için aşağıda belirtilen yapısal kapsama analizi çözümlemelerini tanımlar[1]:

- Gereksinim tabanlı testlerde eksiklik
- Yazılım gereksinimleri ile uyumsuzluk
- Yabancı kod – Ölü kod (extraneous code – dead code)
- Etkin olmayan kod(deactivated code)

DO-178C tarafından doğrudan çözümleme yöntemi olarak tanımlanmayan; ancak yapısal kapsama analizinde sıkça ele alınan bir başka tanım ise savunma amaçlı (defensive) yazılmış kod parçasıdır. Savunma amaçlı yazılan kod parçası, gerekçeli bir tasarım kararı sonucunda, koruma ve güvenlik amacıyla yazılan kod parçasıdır. Sistem ya da yazılım gereksinimlerine doğrudan bir izlenebilirliği bulunmamakla birlikte bilinçli olarak tasarıma hizmet etmesi amacı ile yerleştirilmiştir. Burada dikkat edilmesi gereken en önemli nokta, savunma amaçlı kodun ölü kod ile karıştırılmaması gerektiğidir.

3 Assembly Programlama Dili Açısından Yapısal Kapsama Analizi

3.1. Assembly Programlama Dili Kullanımı

Assembly programlama dili, makineye özgü olan "düşük seviyeli" bir programlama dilidir. "Yüksek seviyeli diller" ile anahtar kelimeler(keywords), kütüphaneler(libraries) ve donanım ile dil arasında yüksek seviyede soyutlama sağlayan bir sözdizimi(syntax) sunulurken, düşük seviyeli programlama dilinden kasıt, kod yapısı ve sözdiziminin (syntax) makine diline yakın olmasıdır [4]. Makine dili, kullanılan mikroişlemciye göre değişiklik gösteren, bizler için anlaşılması, çözümlemesi zor olan bir dildir. Assembly programlama dili bu noktada, bilgisayarın işlemcisi (processor), hafızası (memory) ve Girdi/Çıktı (Input/Output – IO) sistemleriyle iletişim kurmak için daha okunabilir bir sözdizimi kullanımı olanağı sunmaktadır.

Günümüzde, özellikle aygıt sürücüleri (device drivers), alt seviye gömülü sistemler (low-level embedded systems) ve gerçek zamanlı sistemler (real-time systems) geliştirme projelerinde Assembly programlama dili kullanımına rastlanmaktadır. Bu dilin tercih edilme gerekçeleri arasında öncelikli olarak donanıma doğrudan müdahale edebilme, özel işlemci komutlarına erişebilme, kritik performans konularını adresleme ihtiyaçları sayılabilir. Assembly programlama dili, sistem kaynaklarını kontrol etme olanağı sunarak, geliştirilen sistemin performans ve etkinliğinin optimizasyonunun sağlanmasına aracı olur [5].

3.2. Assembly Programlama Dili ile Yazılım Kapsaması

Emniyet kritik aviyonik yazılımlarda gerektiği durumlarda Assembly kullanımına rastlanmaktadır. Assembly ile kodlanan kısımların tasarım teminat seviyesinin C ve üzeri olarak sınıflandırılması durumunda DO-178C gereğince yapısal kapsama hedefine tabi olduğundan, atanan seviyeye göre yapısal kapsama analizinin gerçekleştirilmesi gerekmektedir. Bu dilin kullanımının getirdiği en büyük endişe, dilin yapısı sebebi ile bu analizin nasıl gerçekleştirileceğidir. Piyasada mevcut araçların bir kısmı Assembly kodun enstrümantasyonunu sağlıyor iken, özellikle aviyonik yazılımlarda tercih edilen birçok araç bu desteği sağlamamaktadır. Bu durumda, sadece bu kısımların kapsamasının sağlanması amacı ile DO-178C araç kalifikasyonu gereksinimleri ile uyumlu [6] ek bir araç ile çalışma ya da ilgili kısmın analizi için alternatif bir yöntem belirleme seçenekleri arasında, projenin çıkarları doğrultusunda karar vermek uygun olacaktır.

Tablo 1. Assembly Kod Kapsama Seviyesi [7]

DO-178C Kapsama Türleri	Assembly/Makine Dili Kapsama Seviyesi
İfade Kapsaması	Komut (instruction) kapsaması
Karar Kapsaması	Komut (instruction) kapsaması + giriş noktası (entrypoint) kapsaması + tüm dallanma komutlarının (branching instructions) dal (branch) kapsaması

DO-178C Kapsama Türleri	Assembly/Makine Dili Kapsama Seviyesi
Uyarlanmış Koşul/Karar Kapsaması (UK/KK)	Dal (branch) kapsaması + Assembly kodun soyutlanabildiği düzeyde (örn. Sözde kod - Pseudo kod seviyesinde) MC/DC uygulaması

DO-178C standardında, Assembly programlama dili kullanılması durumunda yapısal kapsama analizine yönelik amaçların nasıl karşılanması gerektiğine ilişkin bir tanımlama mevcut değildir. Benzer şekilde, literatür taraması yapıldığında da, Assembly programlama dili ile geliştirilen yazılımlar için yapısal kapsama analizinin nasıl uygulanması gerektiğine ilişkin herhangi bir çalışmaya rastlanılmamaktadır. Ancak konuya ilişkin Federal Aviation Administration (FAA) tarafından "Software Verification Tools Assessment Study"[7] adlı raporda bazı öneriler sunulmuştur. Tablo 1' de bu raporda verilen ve Assembly programlama dili ile geliştirilen kod kısımlarının hangi seviyede kapsanması gerektiğine ilişkin öneriler özet olarak sunulmuştur. Buna göre, kapsama türü bazında aşağıdaki işlemlerin yapılması önerilmektedir:

- **İfade Kapsaması**: İfade kapsaması için izlenecek yöntem, yüksek seviyeli dillerde izlenmesi gereken yöntemden çok farklı olmayıp amaç, gereksinim tabanlı testlerin koşumu sonrasında, Assembly kod dosyası içerisinde tanımlı tüm kodların çalıştırılmasıdır.
- **Karar Kapsaması:** Karar kapsaması için izlenecek yöntemde sadece ifade kapsaması karşılanmamalı, aynı zamanda dallanma komutlarının, kodun kapsamasındaki etkisi de dikkate alınmalıdır. Dallanma komutları, kodun gidişatında değer atamaları ya da karşılaştırmaları yapıldıktan sonra, kod yapısı içerisinde dallanmayı sağlayan komutlardır (Örn. beq(branch equal to), bne(branch not equal to) gibi). Buradaki endişe koddaki sıçramalardan dolayı, kodun belirli bir kısmının çalıştırılmama ihtimalinin oluşmasından kaynaklıdır ve yapısal kapsama yapılırken bu durum göz ardı edilmemelidir. Ayrıca, kod dosyası içerisinde tanımlı fonksiyonlara en az bir kere girildiği de gösterilmelidir.
- **Uyarlanmış Koşul/Karar Kapsaması**: Bu yöntem, önceki bölümlerde de belirtildiği üzere en karmaşık kapsama yöntemi olup Assembly kullanımı ile çözümlemesi daha zor bir problem haline gelmektedir. "Software Verification Tools Assessment Study" [7] raporu, Assembly kod dosyası üzerinde dal kapsamasının sağlanmasını tavsiye etmektedir. Bunun yanı sıra, her bir koşulun en az bir kere denendiğini ve değişen koşulların birbirinden bağımsız olarak sonucu etkilediğini, doğrudan Assembly dosyası üzerinden analiz etmek yerine, Assembly ile yazılan kodun Sözde kod (Pseudo kod) karşılığının oluşturulup, bunun üzerinden analizin yapılmasını önermektedir.

3.3. Assembly Kodu Yapısal Kapsama Analizi Çalışması

Bu çalışmada öncelikle C++ dilinde yazılmış bir kod parçasının İfade, Karar ve Uyarlanmış Koşul/Karar kapsamasını sağlamak için gerekli olan minimum gereksinim tabanlı test seti belirlenmiştir. Çalışmanın ikinci aşamasında ise, Visual Studio Community 2017 sürümünün derleyici özelliği ile C++ kodunun x86 Assembly programlama dili karşılığı oluşturulup C++ kodunun yapısal kapsaması için belirlenen minimum set,

aynı işlevi gerçekleştiren Assembly kodu üzerinde çalıştırılarak kapsama oranları belirlenmiştir. Kapsama oranları belirlenirken, herhangi bir yardımcı araç kullanılmamıştır. Bunun yerine her kod satırına bir kesme noktası(break point) konularak, yazılım hata ayıklama modunda(debug mode) derlenmiş kod üzerinde testler çalıştırılıp kapsama oranları manuel olarak hesaplanmıştır. Burada hedeflenen çalışma, emniyet kritik bir yazılım gereksiniminin tam ve doğru test edildiğinin gösterimi değil, kodun yapısal kapsamasını sağlamak için oluşturulacak minimum setin üst seviye bir dil ile alt seviye bir dil üzerindeki etkilerini gözlemlemektir.

Çalışma için seçilen örnekte, hava hızı değerinin, bir döngü önceki hava hızı değerine göre belirlenen tolerans değeri içerisinde değişmesi durumunda, ekranda değerin hangi renkte gösterileceği gereksinimi kodlanmıştır.

```
int UpdateAirSpeed()
{
    int ret_val = 4;
     if( ((airspeed >= prev_airspeed) && (airspeed - prev_airspeed) < 10) ||
         ((prev_airspeed > airspeed) && (prev_airspeed - airspeed) < 10) )
     {
        if(airspeed >= 1 && airspeed <=100)
        {
            ret_val = 1; //AirSpeedData.fontColor = Color.Yellow;
        }
        else if (airspeed > 100 && airspeed < 500 )
        {
            ret_val = 2; //AirSpeedData.fontColor = Color.White;
        }
        else if (airspeed >= 500 && airspeed <= 600 )
        {
            ret_val = 3; //AirSpeedData.fontColor = Color.Red;
        }
     }

    return ret_val;
}
```

Şekil 1. Hava Hızı Değeri Değişimi C++ Kodu

Şekil 1'de tanımlanan C++ kodunu kapsamak için oluşturulan test girdi/çıktı seti ve test koşumu sonrası C++ kodu üzerinden elde edilen yapısal kapsama oranları Tablo 2'de tanımlanmıştır.

Tablo 2. C++ Test Senaryoları & Kod Kapsama Oranları

Kapsama Türü	Girdiler	Sonuç	Kapsama Oranı
İfade	İ1: airspeed= 50, prev_airspeed = 50	ret_val= 1	%100
	İ2: airspeed= 150, prev_airspeed = 150	ret_val= 2	
	İ3: airspeed= 550, prev_airspeed = 550	ret_val= 3	
Karar	K1: airspeed= 50, prev_airspeed = 0	ret_val= 4	%100
	K2: airspeed= 50, prev_airspeed = 50	ret_val= 1	
	K3: airspeed= 150, prev_airspeed = 150	ret_val= 2	
	K4: airspeed= 550, prev_airspeed = 550	ret_val= 3	

Kapsama Türü	Girdiler	Sonuç	Kapsama Oranı
	K5: airspeed= 650, prev_airspeed = 650	ret_val= 4	
UK/KK	U1: airspeed= 0, prev_airspeed = 0	ret_val= 4	%100
	U2: airspeed= 50, prev_airspeed = 60	ret_val= 4	
	U3: airspeed= 59, prev_airspeed = 60	ret_val= 1	
	U4: airspeed= 150, prev_airspeed = 0	ret_val= 4	
	U5: airspeed= 150, prev_airspeed = 150	ret_val= 2	
	U6: airspeed= 550, prev_airspeed = 550	ret_val= 3	
	U7: airspeed= 650, prev_airspeed = 650	ret_val= 4	

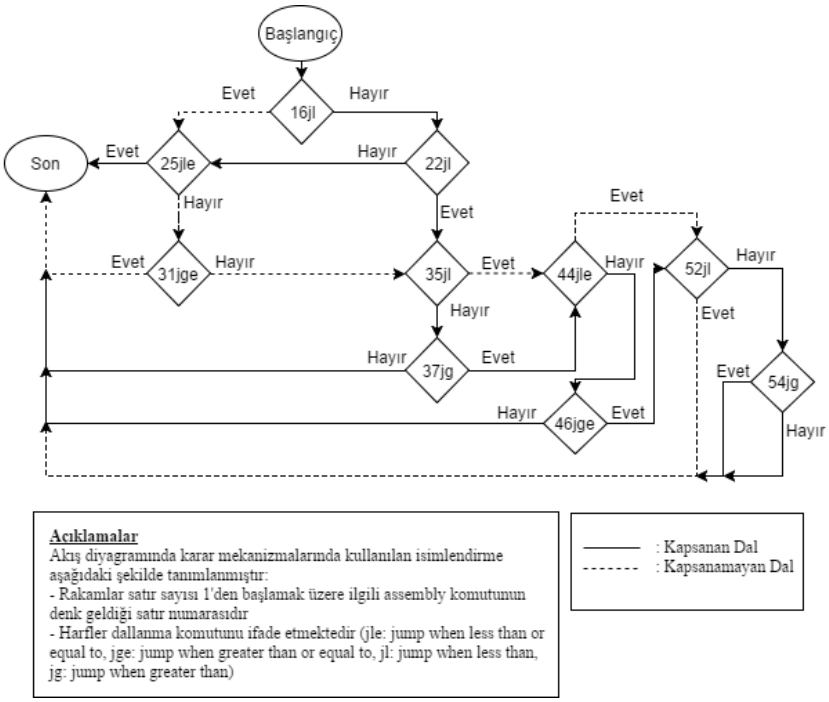
Çalışmanın ikinci safhası için Tablo 2'de verilen girdi seti Assembly kod için denenmiş ve aşağıdaki komut kapsama oranları elde edilmiştir:

Tablo 3. Assembly Kodu Kapsama Oranları

Girdi Seti	Kapsanan Komut / Toplam Komut	Kapsama Oranı
İfade (İ1-İ2-İ3)	36/45	%80
Karar (K1-K2-K3-K4-K5)	39/45	~%86,7
UK/KK (U1-U2-U3-U4-U5-U6-U7)	45/45	%100

Burada ilk olarak dikkat çeken nokta; test girdi setlerinin aynı işleve sahip kodun, üst seviye bir programlama dili ile Assembly programlama dilinde yazılmış halleri arasında farklı kapsama oranları elde edilmiş olmasıdır. İfade kapsaması için hazırlanan gereksinim tabanlı test seti, üst seviye programlama dili ile geliştirilmiş olan kod (Şekil 1) üzerinde %100 ifade kapsaması sağlamış olmasına rağmen, aynı test seti Assembly kodu üzerinde koşturulduğunda %80 oranında komut kapsaması sağlanabilmiştir.

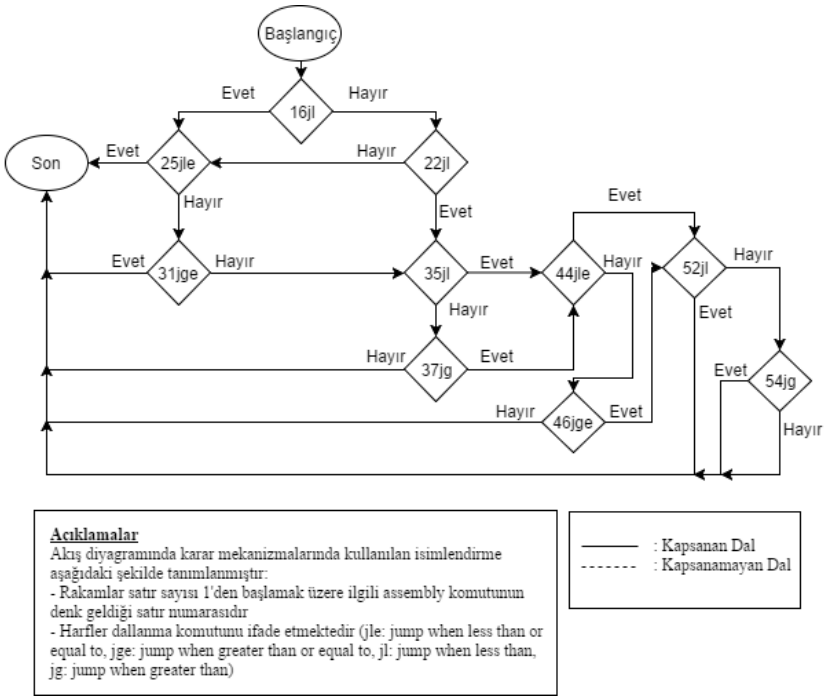
Karar ve UK/KK seviyesinde değerlendirme yapabilmek için de FAA tarafından yayınlanan "Software Verification Tools Assessment Study" [7]'de tanımlandığı üzere dallanma kapsamasını da incelemek gerekmektedir. Kapsanan dallanmaların daha anlaşılır olması için Şekil 2 ve Şekil 3 sunulmuştur.



Şekil 2. Assembly Kod Karar Dallanma Kapsaması

Çalışmada geliştirilen C++ kodunun(Şekil 1) karar kapsaması için oluşturulan K1-K2-K3-K4-K5 test adımları, aynı kodun Assembly programlama dilindeki karşılığı için koşturulduğu zaman Şekil 2'de verilen dal kapsama verisi elde edilmiştir. Bu veri kullanılarak hesaplanan Assembly kodu dal kapsama oranının, Kapsanan Dal Sayısı(13)/Toplam Dal Sayısı (20) hesabına göre %65 olduğu gözlemlenmiştir. Bu durum göstermektedir ki gereksinim tabanlı test seti üst seviye programlama dilinde %100 karar kapsaması sağlarken aynı set, Tablo 1. Assembly Kod Kapsama Seviyesi'nde Assembly programlama dili için tanımlanmış olan karar kapsama seviyesini sağlamak açısından yetersiz kalmıştır.

C++ kodunun(Şekil 1) UK/K kapsaması için oluşturulan U1-U2-U3-U4-U5-U6-U7 test adımları, aynı kodun Assembly programlama dilindeki karşılığı için koşturulduğu zaman Şekil 3'te verilen dal kapsama verisi elde edilmiştir. Bu veri kullanılarak hesaplanan Assembly kodu dal kapsama oranının, Kapsanan Dal Sayısı(20)/Toplam Dal Sayısı (20) hesabına göre %100 olduğu gözlemlenmiştir.



Şekil 3. Assembly Kod - Uyarlanmış Koşul/Karar Dallanma Kapsaması

Üst seviye programlama dilinde, UK/KK için oluşturulan test seti çalıştırılarak %100 UK/K kapsaması elde edilmiş ve FAA tarafından yayınlanan "Software Verification Tools Assessment Study" [7]'de hedeflenen sözde kod (pseudo code) kriteri de bu test seti ile sağlanmıştır (bkz. Tablo 2). Aynı test seti, Assembly programlama dili ile oluşturulan kod üzerinde çalıştırıldığında da, hem komut hem de dal kapsaması %100 oranında elde edilmiştir.

Bu çalışmadan elde edilen sonuçlar neticesinde iki önemli gözlemden bahsedilebilir. Bunlardan ilki, gereksinim tabanlı test setinin, tasarım teminatı seviyesine (seviye A, B ve C için) karşılık gelen yapısal kapsama tipini sağlamaya yönelik oluşturulan test adımlarına ilişkindir. Üst seviye programlama dillerinde ifade ve karar kapsaması için yeterli olan test adım sayısı, Assembly programlama dili için yetersiz kalmış ve ancak UK/KK için yazılan test seti ile tam kapsama sağlanmıştır. Bu durumda, tasarım teminatı seviyesi B veya C olarak atanan fonksiyonların Assembly gibi alt seviye diller ile geliştirilmesi durumunda hedeflenen yapısal kapsamanın sağlanabilmesinin, üst seviye dillere göre daha maliyetli olabileceği söylenebilir. Bir diğer önemli gözlem ise tasarım teminatı seviyesi A olarak atanan ve üst seviye dil kullanılarak geliştirilen fonksiyonlar için yazılan test seti her ne kadar karmaşık ve maliyetli gözükse de, derleyici tarafından çalıştırılabilir kaynak koda dönüştürülmeden önce oluşturulan alt seviye Assembly programlama dilinde de yüksek oranda kapsama elde edilebilmektedir. Bu sonuç uçuş kritik yazılımlar için yapılan doğrulama faaliyetlerinin yeterliliğini ortaya koymaktadır. DO-178C doğrulama aktiviteleri hedefleri gereğince tasarım teminatı seviyesi A olarak atanan fonksiyonlar için derleyici tarafından üretilen nesne kodunun(object code) da

analize tabi olduğuna da dikkat çekmek gerekmektedir. Nesne kodu analizleri ile derleyici tarafından üretilen nesne kodu içerisinde kaynak koda izlenebilirliği kurulamayan herhangi bir kod parçasının olmadığının da güvence altına alınması gerekmektedir.

4 Sonuç ve Öneriler

Bu çalışmada, DO-178C uyumlu yazılım geliştirme projelerinde kodlama esnasında Assembly programlama dili kullanılması durumunda, DO-178C standardı tarafından karşılanması gereken bir amaç olarak tanımlanmış olan kod kapsama analizinde ne gibi zorluklar ve problemler ile karşılaşılabileceği üzerinde durulmaya çalışılmıştır. Çalışmada her ne kadar bir örnek üzerinden gidildiği için genelleme yapacak sonuçlar elde edilmemiş olsa da, Assembly programlama dili kullanılan projelerde yapısal kapsama analizine ilişkin karşılaşılabilecek olası sorunlara ilişkin gözlemler ortaya konulmuştur. Sertifikasyon gerektiren projelerde, standartların beklediği amaçları kısmen karşılamak gibi bir durum söz konusu olmadığı için, bu tarz teknik problemleri projenin başlangıcında öngörebilmek ve planlamaları buna göre yapabilmek ciddi önem arz etmektedir.

Aviyonik yazılım geliştirmede dünya genelinde kabul görmüş olan DO-178C standardı da, diğer emniyet kritik yazılım geliştirme standartlarına benzer şekilde kod kapsama analizine ilişkin amaçlar tanımlamıştır. Geliştirme esnasında C, C++, Ada gibi yüksek seviyeli diller kullanılması durumunda, piyasada mevcut olan birçok farklı araç kullanılarak test koşuları sonucunda yazılım kaynak kodunun ne kadar kapsandığı verisi toplanabilmekte ve analizler yapılabilmektedir. Ancak bazı durumlarda alt seviye bir dil olan Assembly programlama dili kullanımı da gerekli olabilmektedir. Assembly dil kullanımının kaçınılmaz olduğu veya proje başlangıcında öngörüldüğü durumlarda, planlama fazından itibaren DO-178C standardının atadığı yapısal kapsama analizi hedefinin karşılanması için izlenecek yöntem tanımlanmalı ve hem tanımlama hem de uygulamada sertifikasyon uyumunu denetleyecek otoritenin onayı alınmalıdır.

Özellikle Assembly programlama dilinin sıklıkla kullanıldığı aygıt sürücü yazılımları konusunda uzmanlaşmış firmaların, konuya ilişkin kendi araçlarını geliştirip kalifiye etmeleri de projelerin verimliliğini arttırmak adına doğru bir yaklaşım olabilir. Büyük miktarda Assembly kodu kullanılması durumunda, kapsama analizlerini araç kullanmadan yapmaya çalışmak hem zahmetli hem de hata yapmaya açık bir yaklaşım olacaktır.

Gelecek dönemde, daha değişik kod yapıları için de benzer çalışmaların yapılması, konuya ilişkin genel sonuçlara varmak adına faydalı olacaktır. Ayrıca derleyici optimizasyon seçeneklerinin oluşturulan nesne koduna etkileri ve bu tarz durumlarda üst seviye dilden yazılan kod temel alınarak yapılan yapısal kapsama analiz sonuçlarının yeterliliği üzerine incelemeler yapılmasının da literatüre ve konuyla uğraşan otorite ve firmalara katkı sağlayacağı değerlendirilmektedir.

5 Kaynakça

1 RTCA: DO-178C Software Considerations in Airborne Systems and Equipment Certification. RTCA Inc., Washington D.C. (2011).

2 Rierson, L.: Developing Safety-Critical Software. CRC Press, Boca Raton (2013).

3 Gifford, W.: Structural Coverage Analysis Method. 15th AIAA/IEEE Digital Avionics Systems Conference Proceedings, (1996).

4 Beal, J., Phillips, A., Densmore, D. ve Cai, Y.: High-Level Programming Languages for Biomolecular Systems. Springer, New York (2011).

5 Puhan, J., Bürmen, A., Tuma, T. ve Fajfar, I.: Teaching assembly and C language concurrently. International Journal of Electrical Engineering Education, 47, 2, pp. 120–131 (2010).

6 RTCA: DO-330 Software Tool Qualification Considerations, RTCA Inc., Washington D.C. (2011).

7 FAA: Software Verification Tools Assessment Study. FAA, Washington D.C. (2007).