Bileşen kalitesi ölçümünde statik kod analizi yaklaşımı

Berkhan Deniz[1], Soner Çınar[1]

[1]Yazılım Mühendisliği Müdürlüğü, Mühendislik Dir., SST Grup Bşk.
ASELSAN A.Ş.

{berkhand, scinar}@aselsan.com.tr

Özet. Statik kod analizi yazılım kaynak kodunun kalite, güvenlik ve güvenilirlik açısından analiz edildiği bir yazılım doğrulama metodudur. Diğer doğrulama faaliyetlerinden farklı olarak, statik analiz otomatize edilebilmekte; bu sayede yazılımın koşmasına gerek kalmadan, ya da özel test durumları oluşturulmadan uygulanabilmektedir. Çeşitli metrikler kullanılarak yazılımların değerlendirilmesi yöntemi, birçok yazılım firması ve araştırmacı tarafından yaygın olarak uygulanmaktadır. Bu çalışmada, yazılım bileşenlerinden otomatik olarak toplanan metrikler kullanılarak, önceden belirlenmiş birtakım kurallara göre bileşen kalitesini ölçmek üzere geliştirilen metot anlatılacaktır. Öncelikle, bu metot kapsamında ölçülen ve hesaplanan metrikler tanımlanacak ve bu metriklerin seçilme nedenleri aktarılacaktır. Sonrasında, bu metrikler kullanılarak yapılan bileşen kalite puanı hesaplama yöntemi açıklanacaktır. Son olarak da, bu metot ile elde edilen kazanımlar ve gelecek için planlanan çalışmalar aktarılacaktır.

Anahtar Kelimeler: Statik kod analizi, Kalite ölçümü, Kod tabanlı metrikler, Bileşen kalitesi, Kodlama standartları

1 Giriş

Statik analiz (statik kod analizi, kaynak kod analizi, statik program analizi olarak da adlandırılabilir), yazılım kaynak kodunun kalite, güvenlik ve güvenilirlik açısından analiz edildiği bir yazılım doğrulama metodudur. Diğer doğrulama faaliyetlerinden farklı olarak, statik analiz otomatize edilebilmekte; bu sayede yazılımın koşmasına gerek kalmadan, ya da özel test durumları oluşturulmadan uygulanabilmektedir.

Statik analiz ile yazılım geliştiricileri ve testçileri; taşma, sıfırla bölme, hafıza ve işaretleyici hataları gibi çeşitli çalışma zamanı hatalarını tespit ve teşhis edebilirler. Bu analizler olası hataların tespit edilmesinin dışında, incelenen yazılımın format açısından değerlendirilmesi (girinti ve boşluk uzunlukları gibi), kaynak kod metriklerinin hesaplanması, mimari yapısının ortaya konulması, kodlama standartlarına uygunluğu gibi önemli özelliklerini de ortaya çıkarmaktadır [1]. Temel statik kod analiz teknikleri, kod satır sayısı, kod içerisindeki yorum yüzdeleri, kod karmaşıklığı gibi yazılım kalite metriklerini üretebildiği gibi; çeşitli kodlama standartlarına uygunluğu da ölçmektedir.

Bu çalışmada statik kod analizi ile otomatik olarak elde edilen birtakım metrikler kullanılarak, geliştirilen bileşenlere puan tayin edilmesi yaklaşımı anlatılacaktır. Bu sayede, belirli bir puanın altında kalan bileşenlerin; gerekli iyileştirme çalışmaları yapılmadan kullanıma alınmaması hedeflenmiştir.

Çalışmanın geri kalanı şu şekilde düzenlenmiştir; bölüm 2'de statik kod analizi ve yazılım kalitesi ilişkisi ile ilgili literatür araştırmaları özetlenmiş, bölüm 3'te önerilen kalite ölçüm metodu sunulmuş, kullanılan metrikler ve seçilme kriterleri ve elde edilen kazanımlar anlatılmıştır, bölüm 4'te önerilen yöntemin geçerliliğini etkileyen faktörlerin üzerinden geçilmiştir. Son olarak bölüm 5'te, sonuçlar ve gelecek için çalışma önerileri verilmiştir.

2 Literatür

2.1 Kalite ölçümü

Yazılım kalitesi ile ilgili olarak literatür incelendiğinde, en çok raporlanan kalite metriğinin yazılım güvenilirliğiyle ilgili olan "hata tespit metriği" olduğu görülmektedir [2]. Bu metrik, yazılım ürününde tespit edilen toplam hata sayısıyla ilişkilidir. Bu metriğin ölçümü için yorum dışı kaynak kod satır sayısı (ing. non-comment source line of codes - SLOC) başına düşen hata sayısı önerilmektedir. Bu ölçümlerde tespit edilen tüm hatalar (derleme ve mantık hataları) sayılmaktadır [3].

Başka bir çalışmada [4], bakım süresi boyunca yazılım ürünündeki toplam değişiklik sayısı (iyileştirme ya da hata çözme kaynaklı) kalite ölçümü için önerilmiştir.

Bu metriklere ek olarak, bakım için harcanan zaman (ing. rework effort) metriği de kalite ölçümünde kullanılmıştır [5].

Diğer bir çalışmada ise, hataların kritikliği ve hata çözme kolaylığı kalite göstergesi olarak önerilmiştir [5]. Kod satır sayısı başına düşen değişiklik sayısı ve sürümler arasındaki kod değişim yüzdesi de önerilen diğer yazılım kalitesi göstergesi metrikleridir.

Hata yatkınlığı metriği de kalite ölçümü için kullanılan başka bir metriktir. Bu metrik, yazılımın yapısal özellikleri üzerinden hesaplanan, yazılım ürününde hata çıkması olasılığı olarak tanımlanmıştır [6].

2.2 Kod tabanlı metriklerin kalite ölçümünde kullanımı

Önceki bölümde kalite ölçümüne yönelik genel yazılım ürün özelliklerinden bahsedilmiştir. Bu bölümde ise, kod tabanlı metrikler üzerinde; özel olarak da nesneye yönelik metrikler üzerinde durulacaktır.

Literatürde yaygın olarak kabul görmüş çok çeşitli metrik kümeleri bulunmaktadır. Nesneye yönelik metrikler arasında en çok kullanılan metrikler Chidamber ve Kemerer (CK) metrikleridir [7]. Bu metrikler aşağıda sıralanmıştır:

- Sınıfın Ağırlıklı Metot Sayısı – Weighted Methods per Class (WMC)
- Kalıtım Ağacının Derinliği – Depth of Inheritance Tree (DIT)
- Alt Sınıf Sayısı – Number of Children (NOC)

- Nesne Sınıfları Arasındaki Bağımlılık – Coupling Between Object Classes (CBO)
- Sınıfın Tetiklediği Metot Sayısı – Response for a Class (RFC)
- Metotların Uyumluluğu – Lack of Cohesion in Methods (LCOM)

CK metrikleri gibi nesneye yönelik yazılım metrikleri dışında, geleneksel işleve dayalı yazılım metrikleri de halen yaygın olarak kullanılmaktadır [8]. Bu metriklere kaynak kodu satir sayısı, açıklama yüzdesi ve McCabe çevrimsel karmaşıklığı [9] (ing. Cyclomatic Complexity) örnek olarak gösterilebilir.

CK metriklerinin, kod satır sayısı ve karmaşıklık gibi kod metriklerinin, yazılım kalitesinin ölçümünde ve hataya yatkın yazılım modüllerinin tespitinde kullanımıyla ilgili kapsamlı bir analiz [2]'de verilmiştir.

CK ve kod satır sayısı metriklerinin yazılım kalitesinde kullanımıyla ilgili çeşitli deneysel araştırmalar da yapılmıştır [9, 10]. Yazılım kalitesinin hata yatkınlığı, güncellenen kod oranı, üretkenlik ve bakım için harcanan zamanla ölçüldüğü bu çalışmalarda, değerlendirilen tüm metriklerin hata yatkınlığı ile ilişkili olduğu gösterilmiştir.

2.3 Kod tabanlı metrik kullanımı ile elde edilen kazanımlar

Statik kod analizi ile üretilen metrikler sayesinde, yazılım kalitesi ölçülebilir ve iyileştirilebilir hale gelmektedir. Yazılım kalitesi ölçümünü kaynak kod kalitesi ve yazılım ürün kalitesi olarak ikiye ayırabiliriz. Kaynak kod kalitesi standartlara uygunluk, idame ettirilebilirlik, derleyici uyarılarının azlığı gibi niteliklerle; yazılım ürün kalitesi fonksiyonellik, güvenilirlik, kullanılabilirlik, verimlilik, güvenlik gibi metriklerle ölçülmektedir.

Yazılım geliştiriciler ve yöneticiler kod tabanlı metriklerden değişik amaçlar için yararlanmaktadır. Sistem seviyesinde tahminlerde bulunulması, erken ölçümlerle güvenlik açısından sıkıntılı bileşenlerin önceden belirlenmesi ve güvenli tasarım ve programlama yönergelerinin geliştirilmesi bu amaçlara örnek olarak verilebilir [7, 11]. Bunlara ek olarak, tüm alternatif metrikler arasından uygun metriklerin secimi, yazılım geliştiricilerin ve yöneticilerin yazılım tasarımlarının yapısal ve kalite özelliklerini tanımlamalarına yardımcı olmaktadır [12]. Önceki bir araştırmada, metrikler ve yazılımın önceki sürümlerine ait veriler ilişkilendirilerek, yazılım modüllerinin hataya bağışık olup olmayacakları başarılı bir şekilde tahmin edilebilmiştir [12]. Benzer şekilde; tasarım, kod ve gereksinimlerden elde edilen nesneye yönelik metriklerle bileşen hata oranları tahmin edilebilmiştir [13]. Başka yazarlar da nesneye yönelik metriklerin kullanılabilirliğini deneysel olarak incelemiş ve hata bağışıklığı ve hata kritikliği konularının ilgili metriklerle ilişkisini göstermişlerdir [10].

Bunlara ek olarak, nesneye yönelik metrikler yazılım geliştiricilerin ve yöneticilerin gerekli geliştirme ve test faaliyetlerinin planlamalarını yapmalarını kolaylaştırmaktadır [14]. Yazılım geliştirme sırasında, geliştirilen bileşenlerin hata yoğunlukları konusunda kestirimde bulunabilmek, yazılımın hangi parçalarının daha fazla test edilmesi gerektiğinin anlaşılması açısından önemlidir [15]. Ayrıca, geliştiriciler yazılım tasarımının kalitesini doğrulamak ve dolayısıyla yazılım kalitesini ve üretkenliği artırmak için de metrikleri toplayıp, analiz etmektedir [9].

Metrikler yeni teknolojilere geçme konusunda da geliştiricilere ve tasarımcılara yardımcı olmaktadır; çünkü metriklerle elde edilen hızlı veriler sayesinde, yeni özelliklere karar verme sureci kolaylaşmaktadır [16].

Sonuç olarak, metrikler uygun olarak kullanıldıkları takdirde, geliştirme ve bakım maliyetlerini azaltır ve yazılım ürününün kalitesini önemli ölçüde artırır. Aşağıdaki tabloda metrik kullanımının avantajları özetlenmiştir:

Tablo 1. Yazılım metriklerinin avantajlarının özeti

Sistem seviyesi tahminlerde kullanılması [7, 11]
Güvenilmez bileşenlerin önceden belirlenmesi [12]
Güvenli tasarım ve programlama yönergelerinin geliştirilmesi [7, 11]
Yazılımın ve yazılım tasarımının kalitesinin ve yapısının belirlenmesi [12]
Hataya yatkınlığın kestirimi [10, 12, 13]
Geliştirme ve test maliyetlerinin tahmini [14]
Yazılım tasarım kalitesinin doğrulanması [9]
Yazılım kalitesinin ve üretkenliğin iyileştirilmesi [9]
Yeni teknolojilere geçme surecinde hızlı tepki alınması [16]
Geliştirme ve bakım maliyetlerinin azaltılması [16]

3 Önerilen Bileşen Kalitesi Ölçüm Metodu

Önceki bölümde anlatıldığı üzere, çeşitli metrikler kullanılarak yazılımların değerlendirilmesi yöntemi, birçok yazılım firması ve araştırmacı tarafından yaygın olarak uygulanmaktadır. Bu bölümde, yazılım bileşenlerinden otomatik olarak toplanan metrikler kullanılarak, önceden belirlenmiş birtakım kurallara göre bileşen kalitesini ölçmek için önerilen metot anlatılacaktır.

3.1 Metodun Geliştirildiği Ortam ve Kullanılan Altyapılar

Bu metot ASELSAN SST (Savunma Sistem Teknolojileri) Grup Başkanlığı bünyesinde bulunan Hava Savunma Silah Sistemleri ve Görüntü İşleme (HSSS-Gİ) yazılım ekibi tarafından geliştirilmiştir. Ekip C++ dilini kullanarak silah sistemleri için gömülü yazılım ürünleri üretmektedir.

Ekip içerisinde, aktif olarak kullanımda olan bir kodlama stilleri ve yazılım isimlendirme kuralları rehberleri bulunmaktadır. Ayrıca ekip tarafından, MISRA C++ [17] kodlama standardının kendi ihtiyaçlarına göre seçilmiş bir altkümesi kullanılmaktadır. Ekip tarafından geliştirilen programlar sayesinde, bileşenlerin kodlama stilleri rehberine, isimlendirme kuralları rehberine ve kodlama standardı altkümesine uygunlukları otomatik olarak ölçülebilmektedir. Bu kapsamda ölçülen metriklerin ayrıntıları bir sonraki bölümde verilecektir.

Ekip yazılım ürünlerini UML tabanlı bir geliştirme aracı kullanarak geliştirmektedir. Bileşenlerin UML modelleri üzerinden otomatik olarak yapılan ölçümleri de öne-

rilen metot kapsamında kullanılmaktadır. Bu metriklerin ayrıntıları da bir sonraki bölümde verilecektir.

Son olarak bileşenlerin kod satır sayısı, yorum satır sayısı, boş satır sayısı gibi yapısal kod metrikleri de önerilen metot kapsamında kullanılmakta ve otomatik olarak ölçülebilmektedir.

3.2 Ölçülen Metrikler

Bileşen kalite ölçüm metodu kapsamında otomatik olarak ölçülen yazılım metrikleri 5 küme altında toplanmıştır:

Yapısal metrikler. Satır sayısı (ing. Line of Code - LOC), kaynak kod satır sayısı (ing. Source Line of Code - SLOC), yorum satır sayısı (ing. Comment Line of Code - CLOC), boş satır sayısı (ing. Blank Line of Code - BLOC).

Eleman sayıları. Paket sayısı, sınıf sayısı, metot sayısı, arayüz fonksiyonu sayısı, sanal (ing. virtual) arayüz fonksiyonu sayısı, statik arayüz fonksiyon sayısı, değişken sayısı (ing. attribute), argüman sayısı, tip sayısı.

Kodlama standartları metrikleri. Kodlama stilleri ihlal sayısı, isimlendirme kuralları kapsamında değerlendirmeye alınan durum sayısı, isimlendirme kuralları ihlal sayısı, MISRA kodlama standardı ihlal sayısı, derleme uyarıları sayısı.

UML model metrikleri. Açıklaması yazılmamış paket sayısı, açıklaması yazılmamış sınıf sayısı, açıklaması yazılmamış arayüz fonksiyonu sayısı, model üzerinden tanımlanmamış yapı sayısı, model üzerinden tanımlanmamış sıralama (ing. enumeration) sayısı, model üzerinden seçilmemiş tip sayısı, ilklenmemiş değişken sayısı.

Birim test metrikleri. Birim test sayısı, birim test uyarıları sayısı, birim test hataları sayısı.

Yukarıda listelenen metrikler kullanılarak, 17 adet karma metrik tanımlanmış ve bileşen kalitesi ölçüm yaklaşımı bu metrikler üzerine kurulmuştur (Bkz. Tablo 2).

3.3 Metriklerin seçilme nedenleri

Önerilen metot kapsamında kullanılan metrikler Tablo 2'de verilmiştir. Bu bölümde, kullanılan metriklerin bileşen kalitesi ölçümü açısından neden uygun metrikler olduğu anlatılacaktır.

M1 ve M2 kod satır sayısı metrikleridir. Bu metrikler değerlendirilen bileşen kodunun anlaşılırlığının ölçülebilmesi amacıyla kullanılmıştır.

Tablo 2. Kullanılan metrikler

M1: Satır sayısı başına düşen yorum satır sayısı (Yorum satır sayısı / satır sayısı)
M2: Satır sayısı başına düşen boş satır sayısı (Boş satır sayısı / satır sayısı)
M3: Kaynak kod satır sayısı başına düşen derleme uyarıları sayısı (Derleme uyarıları sayısı / kaynak kod satır sayısı)
M4: Metot başına düşen kodlama stilleri ihlal sayısı (Kodlama stilleri ihlal sayısı / metot sayısı)
M5: İsimlendirme kuralları ihlal oranı (İsimlendirme kuralları ihlal sayısı / isimlendirme kuralları kapsamında değerlendirmeye alınan durum sayısı)
M6: Kaynak kod satır sayısı başına düşen MISRA kodlama standardı ihlal sayısı (MISRA kodlama standardı ihlal sayısı / kaynak kod satır sayısı)
M7: Açıklaması yazılmamış paket oranı (Açıklaması yazılmamış paket sayısı / paket sayısı)
M8: Açıklaması yazılmamış sınıf oranı (Açıklaması yazılmamış sınıf sayısı / sınıf sayısı)
M9: Açıklaması yazılmamış arayüz fonksiyonu oranı (Açıklaması yazılmamış arayüz fonksiyonu sayısı / arayüz fonksiyonu sayısı)
M10: Soyut olmayan arayüz fonksiyonu oranı ((Sanal arayüz fonksiyonu sayısı + statik arayüz fonksiyon sayısı) / arayüz fonksiyonu sayısı)
M11: Model üzerinden tanımlanmamış yapı oranı (Model üzerinden tanımlanmamış yapı sayısı / tip sayısı)
M12: Model üzerinden tanımlanmamış sıralama oranı (Model üzerinden tanımlanmamış sıralama sayısı / tip sayısı)
M13: Model üzerinden seçilmemiş değişken ve argüman sayıları oranı (model üzerinden seçilmemiş tip sayısı / (değişken ve argüman sayıları toplamı))
M14: İlklenmemiş değişken oranı (ilklenmemiş değişken sayısı / değişken sayısı)
M15: Birim test sayısının arayüz fonksiyonları dışındaki fonksiyonlara oranı (Birim test sayısı / (metot sayısı - arayüz fonksiyonu sayısı))
M16: Uyarı veren birim testleri oranı (Birim test uyarıları sayısı / birim test sayısı)
M17: Hata veren birim testleri oranı (Birim test hataları sayısı / birim test sayısı)

M3, M4, M5 ve M6 bileşen kodunun çeşitli standartlara uygunluğunun değerlendirildiği metriklerdir. Standartlara uygunluk sayesinde, bu standartların getirdiği kalite özelliklerinin bileşene kazandırılması hedeflenmiştir [1]. Önceden belirlenmiş kurallara uygun kod geliştirmenin kodun anlaşılabilirliğini ve güvenilebilirliğini artıracağı da düşünülerek bu metrikler bileşen kalitesi ölçümü kapsamında kullanılmıştır.

M7, M8 ve M9 UML modeli üzerinden alınan metriklerdir. Bileşen içerisindeki birimlere, bu birimlerin görevlerine ve kullanım amaçlarına yönelik açıklamaların girilip girilmediğini ölçmektedir. Bileşenin anlaşılabilirliğinin ve bakim yapılabilirliğinin ölçümü açısından bu metrikler önerilen metotta kullanılmaktadır.

M10 bileşende bulunan arayüz fonksiyonlarıyla ilgili bir metriktir. Doğru programlama pratiği olarak, geliştirilen yazılım ürünlerindeki arayüz fonksiyonlarının soyut olması ve böylece bu arayüzlerden kalıtım yolu ile gerçeklenen sınıfların tüm arayüz fonksiyonlarını gerçeklemeleri hedeflenmiştir. Bu şekilde, arayüz fonksiyonlarının

güncellenmeleri sonucu oluşabilecek imza değişikliği gibi durumlardan, bileşen geliştiricisinin derleme zamanında haberdar olması hedeflenmiştir. Bu metrik bileşen yazılımının uyarlanabilirliği açısından önemlidir.

M11, M12 ve M13 metrikleri UML modeli üzerinden ölçülen metriklerdendir. Kullanılan kod geliştirme aracı, yapı ve sıralama tiplerinin tanımlanmasının ve değişken ve argümanların tiplerinin seçilmesinin hem elle (manüel olarak) hem de model üzerinden (çoktan seçme yoluyla) yapılmasına izin vermektedir. Yapı ve sıralama tipleri model üzerinden tanımlandıkları takdirde, geliştirme aracının dışarıya açtığı programlama arayüzleri üzerinden erişilebilir olmaktadır ve kodun tamamındaki kullanım şeklinin tek bir yerde tanımlanmasına olanak sağlamaktadır. Değişken ve argüman tipleri model üzerinden seçildiği takdirde ise, bu tiplerin isim değişiklikleri sonucu oluşabilecek derleme hataları engellenmekte ve ilgili bağımlılıklar otomatik olarak bileşene eklenmektedir. Tanımlamaların elle yapılması yerine model üzerinden yapılması sayesinde, bileşenlerin geliştirme zamanı azalmakta ve bakım yapılabilirliği ve uyarlanabilirliği olumlu yönde etkilenmektedir. Bu sebeplerle, bu metrikler önerilen metoda dâhil edilmiştir.

M14 metriği bileşende bulunan ilklenmemiş değişkenleri ölçmektedir. İlklenmemiş değişkenler, bileşendeki en önemli hata ve güvenlik açığı kaynakları arasında bulunmaktadır [1]. Bu metrik, bileşen güvenliğinin ve güvenilirliğinin ölçümü açısından çok kritik olarak değerlendirilmiş ve bu sebeple önerilen metoda eklenmiştir.

Son olarak, M15, M16 ve M17 metrikleri bileşenlerin birim testlerine yönelik olarak seçilmiş metriklerdir. Bileşenler geliştirilirken, bileşenin sağlamlığı açısından bileşende bulunan (arayüz fonksiyonları dışındaki) tüm fonksiyonların test edilmesi planlanmıştır. Bu sebeple, bu metrikler de önerilen metoda dâhil edilmiştir.

3.4 Bileşen kalitesi ölçüm metodu

Bu bölümde, Tablo 2'deki metrikler kullanılarak oluşturulan bileşen kalitesi ölçüm metodu anlatılacaktır.

İlk olarak Tablo 2'deki metriklerin nasıl değerlendirildiği anlatılacaktır. Bu metriklerin her biri için, üç seviyeli olarak yüzde cinsinden bir başarı kriteri oluşturulmuştur. Bu seviyeler "Başarılı", "Uyarı var" ve "Başarısız" olarak isimlendirilmiştir.

Metriklerin başarı kriterleri tablosu Tablo 3'te gösterilmiştir. Tablo ayrıntılı olarak incelendiğinde, önerilen bileşen kalitesi ölçüm metodunda hangi metriklere daha çok önem verildiği, hangi metriklerde toleransın daha düşük olduğu anlaşılmaktadır. Örnek olarak; bazı metriklerde başarı kriteri %0-1 iken (M3, M9, M11, M12 ve M17), M14 metriğinde kesinlikle %0'dır.

Önerilen metot dört aşamadan oluşmaktadır:

1. Kalite ölçümü yapılacak bileşen için Bölüm 3.2'de anlatılan metriklerin otomatik olarak ölçülmesi
2. Alınan metrikler kullanılarak, Tablo 2'deki metriklerin, ölçümü yapılacak bileşen için hesaplanması
3. Bu metriklerin Tablo 3'teki başarı kriterlerine göre değerlendirilmesi

4. Değerlendirme sonucunda, metriklerin puanlandırılması (başarılı metriklere 100, uyarı veren metriklere 50, başarısız metriklere ise 0 puan verilmektedir) ve bu puanların ortalamasının "bileşen kalite puanı" olarak belirlenmesi

Tablo 3. Metrik başarı kriterleri

	Başarılı		Uyarı var		Başarısız	
	Min	Maks	Min	Maks	Min	Maks
M1	20%	100%	10%	20%	0%	10%
M2	0%	30%	30%	50%	50%	100%
M3	0%	1%	1%	2%	2%	100%
M4	0%	10%	10%	20%	20%	100%
M5	0%	10%	10%	20%	20%	100%
M6	0%	10%	10%	20%	20%	100%
M7	0%	5%	5%	20%	20%	100%
M8	0%	10%	10%	50%	50%	100%
M9	0%	1%	1%	2%	2%	100%
M10	0%	20%	20%	40%	40%	100%
M11	0%	1%	1%	2%	2%	100%
M12	0%	1%	1%	2%	2%	100%
M13	0%	5%	5%	10%	10%	100%
M14	0%	0%	0%	0%	0%	100%
M15	50%	100%	10%	50%	0%	10%
M16	0%	10%	10%	20%	20%	100%
M17	0%	1%	1%	2%	2%	100%

Örnek bir bileşen için ölçülen metrikler Tablo 4'te gösterilmiştir. Bu metrikler kullanılarak, hesaplanan metrikler ise Tablo 5'te verilmiştir. Bu tabloda ayrıca, hesaplanan metriklerin değerlendirilmesi ve puanlandırılması ve bu puanların ortalamasının alınarak bileşen kalite puanının hesaplanması da gösterilmiştir.

Tablo 4. Örnek bileşen için ölçülen metrikler

Yapısal Metrikler	Satır sayısı	28175
	Bos satır sayısı:	4968
	Kaynak kod satır sayısı:	13795
	Yorum satır sayısı:	9303
Eleman Sayıları	Paket:	14
	Sınıf:	67
	Metot:	356
	Arayüz fonksiyonu:	71
	Sanal arayüz fonksiyonu:	34
	Statik arayüz fonksiyonu:	35
	Değişken:	860
	Argüman:	456

	Tip:	75
Kodlama Standartları Metrikleri	Kodlama stilleri ihlalleri:	0
	İsimlendirme kuralları kapsamında değerlendirmeye alınan durum sayısı:	2282
	İsimlendirme kuralları ihlal sayısı:	130
	MISRA kodlama standardı ihlal sayısı:	227
	Derleme uyarıları sayısı:	4
UML Model Metrikleri	Açıklaması yazılmamış paket sayısı:	1
	Açıklaması yazılmamış sınıf sayısı:	5
	Açıklaması yazılmamış arayüz fonksiyonu sayısı:	0
	Model üzerinden tanımlanmamış yapı sayısı:	0
	Model üzerinden tanımlanmamış sıralama sayısı:	0
	Model üzerinden seçilmemiş tip sayısı:	0
	İlklenmemiş değişken sayısı:	0
Birim Test Metrikleri	Birim test sayısı:	0
	Birim test uyarıları sayısı:	0
	Birim test hataları sayısı:	0

Tablo 5. Örnek bileşen için hesaplanan metrikler (% cinsinden) ve bileşen kalite puanı

Metrikler	Ölçüm (%)	Değerlendirme	Metrik Puanı
M1: Satır sayısı başına düşen yorum satır sayısı:	33,02%	Başarılı	100
M2: Satır sayısı başına düşen bos satır sayısı:	17,63%	Başarılı	100
M3: Kaynak kod satır sayısı başına düşen derleme uyarıları sayısı:	0,03%	Başarılı	100
M4: Metot başına düşen kodlama stilleri ihlal sayısı:	0,00%	Başarılı	100
M5: İsimlendirme kuralları ihlal oranı:	5,70%	Başarılı	100
M6: Kaynak kod satır sayısı başına düşen MISRA kodlama standardı ihlal sayısı:	1,65%	Başarılı	100
M7: Açıklaması yazılmamış paket oranı:	7,14%	Uyarı var	50
M8: Açıklaması yazılmamış sınıf oranı:	7,46%	Başarılı	100
M9: Açıklaması yazılmamış arayüz fonksiyonu oranı:	0,00%	Başarılı	100
M10: Soyut olmayan arayüz fonksiyonu oranı:	50,70%	Başarısız	0
M11: Model üzerinden tanımlanmamış yapı oranı:	0,00%	Başarılı	100
M12: Model üzerinden tanımlanmamış sıralama oranı:	0,00%	Başarılı	100
M13: Model üzerinden seçilmemiş değişken ve argüman sayıları oranı:	4,94%	Başarılı	100
M14: İlklenmemiş değişken oranı:	1,05%	Başarısız	0
M15: Birim test sayısının arayüz fonksiyonları dışındaki fonksiyonlara oranı:	0,00%	Başarısız	0
M16: Uyarı veren birim testleri oranı:	100,00%	Başarısız	0
M17: Hata veren birim testleri oranı:	100,00%	Başarısız	0
		Bileşen Kalite Puanı:	67,65%

3.5 Bileşen kalite puanı ölçümü ile sağlanan kazanımlar

Önerilen ve yazılım geliştirme süreci boyunca aktif olarak kullanılan bu metot ile öncelikle bileşenlerin değerlendirilmesinin standartlaşması amaçlanmıştır. Bileşenlerin çıkarılan tüm sürümlerinin bileşen kalite puanı hesaplanması şart koşulmuştur.

Buna ek olarak da, kullanıma alınacak sürümlerde, "başarısız" olarak değerlendirilen hiçbir metrik bulunmaması kuralı benimsenmiştir. Bu sayede, bileşenin kalitesi istenen seviyeye gelene kadar, yazılım geliştirme çalışmalarının sürdürülmesi sağlanmıştır.

Ayrıca, ekip içerisinde kullanılan standartlara (isimlendirme, MISRA, kodlama stilleri) uygun olarak kod geliştirmesi hedeflenen, şirket dışı geliştiricilerin (altyüklenici gibi) ve ekibe yeni başlayanların kurallara uygun olarak, istenen kalitede yazılım geliştirmeleri temin edilmiştir. Bu şekilde kurallara ve standartlara uygun yazılım geliştirilmesi, metrik ölçümünün otomatik olarak yapılması ve geliştiricilerin nerede hata yaptıklarına dair otomatik geri besleme almaları sayesinde sağlanmıştır.

4 Değerlendirme

Bu çalışmada anlatılan metot, gömülü sistemler için güvenlik kritik yazılım ürünleri üreten bir ekip tarafından geliştirilmiştir. Metot kapsamında ölçülen ve hesaplanan metrikler yazılım geliştirilen alan için uygun ve yeterli görülmüştür. Hesaplanan metriklerin başarı kriterleri de, yine ekip tarafından öznel olarak belirlenmiştir. Önerilen değerlendirme ve puanlama yönteminin, farklı uygulama alanlarında kullanılması durumunda, o alan ihtiyaçlarına yönelik olarak güncellenmesi gerekebilir.

5 Sonuç ve Gelecek için Planlar

Önerilen metot ile geliştirilen yazılım bileşenlerinin ardışık sürümler arasında kalitesinin iyileştirilebilmesi sağlanmıştır. Bileşenlerin kullanıma alınabilmeleri için ekip tarafından önceden belirlenmiş alt eşik puanların üzerinde kalite puanına sahip olması şart koşulduğundan, kullanımdaki bileşenlerin belli bir kalitenin üzerinde olacakları temin edilmiştir.

İleriki hedefler arasında, bileşenlerin CK metriklerinin de ölçülmesi ve burada önerilen kalite puanının ölçülen diğer kod metrikleriyle ilişkisinin belirlenmesi bulunmaktadır. Ayrıca, bileşenlerin kullanıma alınmaları sonrasında tespit edilecek hatalarla, burada önerilen kalite puanlarının arasındaki ilişkinin belirlenebilmesi de gelecek planları arasında bulunmaktadır.

Kaynaklar

1. Louridas, Panagiotis. "Static code analysis." Software, IEEE 23.4 (2006): 58-61.
2. Berkhan Deniz, "Investigation of The Effects of Reuse on Software Quality in an Industrial Setting," M.S. thesis, Electrical and Electronics Engineering Dept., Middle East Technical University, Ankara., Turkey, 2013.
3. William Frakes and Carol Terry, "Software reuse: metrics and models," ACM Computing Surveys, vol. 28, no. 2, pp. 415-435, June 1996.
4. W. B. Frakes and G. Succi, "An industrial study of reuse, quality, and productivity," Journal of Systems and Software, vol. 57, no. 2, pp. 99 - 106, June 2001.

5. Parastoo Mohagheghi and Reidar Conradi, "Quality, productivity and economic benefits of software reuse: a review of industrial studies," Empirical Software Engineering, vol. 12, no. 5, pp. 471-516, 2007.
6. Lionel C. Briand, Jurgen Wust, John W. Daly, and D. Victor Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," The Journal of Systems and Software, vol. 51, no. 3, pp. 245-273, May 2000.
7. Khaled El-Emam, "Object-oriented metrics: A review of theory and practice," in Advances in software engineering. New York, NY, USA: Springer-Verlag New York, Inc, 2002, pp. 23-50.
8. Erdemir Ural, Umut Tekin, and Feza Buzluca. "Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi." Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu (2008).
9. R. Subramanyam and M.S Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects," IEEE Transactions on Software Engineering, vol. 29, no. 4, pp. 297- 310, April 2003.
10. Yuming Zhou and Hareton Leung, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults," IEEE Transactions on Software Engineering, vol. 32, no. 10, pp. 771-789, October 2006.
11. Rüdiger Lincke, Jonas Lundberg, and Welf Löwe, "Comparing software metrics tools," in ISSTA '08 Proceedings of the 2008 international symposium on Software testing and analysis, Seattle, Washington, USA, 2008, pp. 131-142.
12. Nachiappan Nagappan, Thomas Ball, and Andreas Zeller, "Mining metrics to predict component failures," in ICSE '06 Proceedings of the 28th international conference on Software engineering , Shanghai, China, 2006, pp. 452-461.
13. Yue Jiang, Bojan Cukic, Tim Menzies, and Nick Bartlow, "Comparing design and code metrics for software quality prediction," in PROMISE '08 Proceedings of the 4th international workshop on Predictor models in software engineering, Leipzig, Germany, 2008, pp. 11-18.
14. Linda H. Rosenberg, "Applying and Interpreting Object Oriented Metrics," in Software Technology Conference, 1988.
15. Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G. L. (2002). Code quality analysis in open source software development. Information Systems Journal, 12(1), 43-60.
16. Seyyed Mohsen Jamali, "Object Oriented Metrics (A Survey Approach) ," Department of Computer Engineering Sharif University of Technology, Tehran, Iran, 2006.
17. MISRA-C++: 2008: Guidelines for the Use of the C++ Language in Critical Systems. MIRA Limited, 2008.