Kamu Yazılımlarında Ürün Kalitesinin Değerlendirilmesi İçin Pratik Bir Kod Kalitesi Modeli

Savaş Öztürk[1] , Nurhan Yağcı[2]

[1,2]Yazılım Test ve Kalite Değerlendirme Laboratuvarı, TÜBİTAK BİLGEM, Gebze, Kocaeli

[1] e-posta:{savas.ozturk[1],nurhan.yagci[2]}@tubitak.gov.tr

Özet. Yazılım kalitesini etkileyen faktörler temel olarak doğrudan ölçülebilenler ve doğrudan ölçülemeyenler olarak ikiye ayrılır. Kamu yazılımlarında süreç ve ürün kalitesi genellikle ihmal edilir ve bu durum maliyeti olumsuz etkiler. Yazılımın test edilmesi ve incelenmesi istendiğinde çoğu zaman projenin sonuna yaklaşılmıştır ve elde sadece yazılım kodu vardır. Bu tür yazılımların kalitesi değerlendirilmek istendiğinde, doğrudan ölçülemeyen kullanılabilirlik, test edilebilirlik, taşınabilirlik gibi kalite faktörlerinin incelenmesi zordur. Diğer taraftan yazılım kodu doğrudan ölçülebilir ve yazılımın kalitesi hakkında ipuçları verebilir. Bu çalışmada, sadece yazılımın koduna bakarak yazılımın kalitesi hakkında elde edilebilecekler araştırılmış ve yazılım kalitesi, yazılım güvenilirliği ve bakım yapılabilirliğini ölçmek amacıyla geliştirilen pratik yazılım kalite modelleri karşılaştırmalı olarak incelenmiştir. Sonuç olarak, yazılımın kalitesini hızlı ve basit bir şekilde ölçmek amacıyla kullanılacak pratik bir model önerilmiştir.

Anahtar Kelimeler: Yazılım Kalite Metrikleri, Ölçüm Otomasyonu, Araç Seçimi

1 Giriş

TÜBİTAK BİLGEM TDBY Yazılım Test ve Kalite Değerlendirme Laboratuvarı (YTKDL), başta TÜBİTAK olmak üzere kamunun yazılımlarında kaliteyi artırmak amacıyla yazılım testleri ve yazılım kalitesi alanlarında hizmet veren bir birimdir. Laboratuvarımıza yazılım kalitesi değerlendirmesi için gelen taleplerden ve sorulardan bazıları aşağıdaki gibidir:

1. Destek vereceğimiz yazılım projelerinin değerlendirilmesinde ve ürün kalitesinin artırılmasında yazılım kalitesi değerlendirmesinden nasıl faydalanabiliriz? (Kalkınma Bakanlığı, Sanayi Bakanlığı)
2. Yazılım geliştiren KOBİ'leri süreç ve ürün kalitesine gore derecelendirmek mümkün müdür? (Sanayi Bakanlığı, TSE)

3. BT Denetimlerinde yazılım kalitesini değerlendirmeye nasıl katabiliriz? (Sayıştay)
4. Tedarik ettiğimiz/geliştirdiğimiz/alt yükleniciye yaptırdığımız projelerde yazılımların kalitesini en az maliyetle ve en kısa sürede nasıl değerlendirebiliriz? (Çok sayıda kamu kurumu)

Taleplerin ve talep edenlerin çeşitliliği; sadece uzmanlar tarafından değil herkes tarafından anlaşılabilen, ekonomik, aynı ürün için her zaman aynı sonucu veren, uygulanabilir iyileştirme önerileri içeren bir kalite değerlendirme raporlamasına ihtiyaç olduğunu göstermektedir.

Ülkemizde başta e-devlet yazılımları olmak üzere kamunun ihtiyacına yönelik geliştirilen yazılımlar, genelde bakım yapılabilirlik ve güvenilirlik sorunları içerir, yeterince test edilmeden onaylanır ve kullanılırken çok sayıda sorun yaşanır. Müşteri, yazılımın şartnameye uygun olarak geliştirildiğinin incelenmesi ve test edilmesi gerektiğini genellikle teslim alırken fark eder. Bu aşamada kendisine sadece yazılımın kodu ile varsa şartnamede belirtilen kısıtlı sayıda dokümantasyon teslim edilir. Kod incelenerek yazılımın kalitesi ve içerdiği hata sayısında öngörüde bulunma başarısı da doğal olarak düşmektedir.

Diğer yandan sadece statik kod analizi ile yazılım hakkında çok önemli bilgiler edinilebileceği, yazılımın barındırdığı çok sayıda hatanın testlerden önce elimine edilebileceği araştırma konusu olmuştur. Johns [1] çok sayıda yazılım projesinden yaptığı çıkarımla, statik kod analizi ile yazılımdaki toplam hata (defect) sayısının %60'ının tespit edilebileceğini ortaya koymuştur. Rome Laboratuvarı Modeli gibi güvenilirlik modelleri de bir projede konsept aşamasından testlere kadar bütün süreçlere ait denetim listeleri ve ölçümleri değerlendirse de, en çok hatanın kodlama sürecinde yapılan yanlışlardan kaynaklandığını iddia etmektedir [2].

Bu çalışmada, yazılımın kalitesinin değerlendirilmesi için sadece kodun sunulduğu durumlarda, öncelikle incelemeye tabi yazılımın kalitesinin pratik bir şekilde değerlendirilmesi, ardından da etkin ve uygulanabilir iyileştirme önerilerinin sunulması için altyapı kurulması amaçlanmıştır. Sadece kod üzerinde inceleme yapılarak, yazılım kalitesinin ne seviyede olduğunu değerlendirmek için kullanılan modeller incelenmiş ve bu modellerden günümüz ihtiyaçlarına göre uygulanabilir bir kod kalitesi değerlendirme modeli oluşturulmuştur. Modeli oluştururken yazılım kalitesine yönelik ISO/IEC 25010:2011 standardı [3] ve McCall yazılım kalitesi modeline [4] ilave olarak bakım yapılabilirliğe ve güvenilirliğe yönelik güncel modellerden Rome Lab. Güvenilirlik Modeli, Sonar Kalite İndeksi Modeli, Sonar Toplam Kalite Modeli, Oman Bakım Yapılabilirlik İndeksi, SIG (Software Improvement Group) Bakim Yapılabilirlik Modeli, Neufelder Full-Scale Modeli incelenmiştir.

Bütün bu modeller incelendiğinde kod üzerinde karmaşıklık, büyüklük, tasarımsal yaklaşım, birim test miktarı, uygulama açıklık bulguları, tekrarlanmış kod oranı, kodlama stili gibi çok çeşitli faktörün dikkate alındığı ve bunların çeşitli kombinasyonlarla bileşkesinin alınarak modeller oluşturulduğu görülmektedir. Öznel değerlendirmelere yol açabilecek denetim listelerinden kaçınılarak ve herkes tarafından erişilebilir özellikle açık kaynak kodlu araçlar tercih edilerek inceleme yapılabilecek bir model önerilmiştir. İncelenen modellerde uygulanabilirliği tarafımızca mümkün olmayan ya

da programlama paradigmalarındaki gelişime ayak uyduramayan bazı kriterler uyarlanarak kullanılmıştır. Önerilen modelin etkinliği, seçilen açık kaynak kodlu yazılımlar üzerinde yapılan analizlerle gösterilmiştir.
Bildirinin ikinci bölümünde adı geçen modeler anlatılacak, üçüncü bölümde modellerin karşılaştırmalı analizi yapılacak, dördüncü bölümde önerilen model açıklanacak ve beşinci bölümde çalışmanın özeti yer alacaktır.

2 Pratik Bakış Açısıyla Yazılım Kalite Modelleri

Yazılım kalite modelleri yetmişli yıllardan itibaren kullanılmakta ve yazılımın kalitesini bütünlük, esneklik, tekrar kullanılabilirlik gibi çeşitli açılardan incelemektedir. Bu bölümde yazılım kalitesini değerlendirmeyi hedefleyen belli başlı modeller incelenecek ve farklılıkları karşılaştırılacaktır.

2.1 McCall Modeli

Yazılım kalitesini belirleyen faktörler iki geniş kategoriye ayırılabilir: 1) Doğrudan ölçülebilenler (örneğin satır sayısı başına düşen hata sayısı) 2) Doğrudan ölçülemeyenler (kullanılabilirlik ya da bakım yapılabilirlik) Her iki kategori için de ölçmek istenen yazılım (doküman, program, veri) bazı kıyas noktaları ile karşılaştırılarak kalitenin seviyesi belirlenmeye çalışılır. Yazılım kalitesini doğrudan ölçmek amacıyla kullanılan ölçütleri geliştirmek çok zordur ve bazı durumlarda imkansızdır.Yazılım kalitesini etkileyen faktörlerin belirlenmesi ve kalitenin sayısal bir şekilde ifade edilmesine yönelik ilk çalışmalardan birisi McCall Kalite Modeli'dir [4].
McCall modeli modelinde bakım yapılabilik, güvenilirlik, esneklik gibi 11 adet kalite faktörü ve bu faktörleri etkileyen 23 kriteri tanımlamıştır. Bu model kalitenin karakteristikleri ile metrikler arasında bağ kurmuştur. Her faktörün kaliteye etkisini ölçmek için F_q, yazılım kalite faktörü, c_n regresyon katsayısı, m_n de metrik değeri olmak üzere (1) tanımlanmıştır.

$$F_q = c_1 m_1 + c_2 m_2 + \ldots + c_n m_n \quad (1)$$

Bu modelde metriklerin çoğu öznel bir şekilde ölçülmektedir. Değerlendiriciye her faktörle ilgili sunulan sorulara 0 (düşük) ile 10 (yüksek) skalasında bir not vermesi istenmektedir. Değerlendiricinin yazılımla ilgili rolüne ve kişisel özelliklerine göre farklı sonuçlar çıkması muhtemeldir. Metriklerin nesnelliği tartışmalıdır ve yazılım ürününün işlevselliği doğrudan ele alınmamıştır.

2.2 ISO/IEC 9126 ve ISO/IEC 25010 Standartları

ISO/IEC 9126 standardı yazılım ürün kalitesini 6 temel karakteristik bileşeni ile ifade eder [5]: İşlevsellik, güvenilirlik, bakım yapılabilirlik, taşınabilirlik, kullanılabilirlik ve verimlilik. Bu 6 karakter 27 alt karaktere bölünür. Standard, bu faktörlerin göstergesi olabilecek metrikler hakkında da standard bir bilgi sunar. ISO kalite modeli standard bir terminoloji sunarak metrikler ve kalite faktörleri hakkında ortak çerçeve oluşturur.

Ancak standartta listelenen, harcanan efor, hata düzeltme veya test süresi gibi metrikler doğrudan ölçülemeyen metriklerdir ve birtakım kabullere dayanır. ISO/IEC 9126'nın güncellenmiş hali olan ISO/IEC 25010 standardı da kalite faktörlerine uyumluluk ve güvenliği ekler, ancak doğrudan ölçülebilir metriklere değinmez [3][6].

2.3 Rome Lab. Güvenilirlik Modeli

Yazılım kalitesini ya da kaliteyi oluşturan alt faktörleri değerlendiren çok sayıda model de kalite faktörlerinin bir denetim listesinde yer alan sorulara cevap verilmesi dayalı olduğu benzer bir yöntemi izlemiştir. 1992 yılında ortaya çıkan Rome Laboratuvarı Modeli yazılım güvenilirliği öngörümü (prediction) ve tahminine (estimation) yönelik kapsamlı bir yaklaşım sunar [2]. Bu model yazılım projesinin konsept aşamasından sistemin devreye alınması ve bakım sürecine kadar tüm safhalarda yazılım hakkında topladığı çeşitli veriler ile yazılımın hata yoğunluğunu (yazılımın barındırdığı satır sayısı başına hata sayısı) öngörür, sistem arızası oranını (belirli bir zaman aralığında sistem arızası oluşmasına yol açan yazılımsal hata) tahmin eder. Bu modeli geliştiren ABD Hava Kuvvetleri'nin geliştirdiği projelerden elde edilen birikim kullanılır, bir takım denetim listeleri yardımı ile öznel bilgiler derlenir, dokümantasyon, süreç, ekibin yapısı ve yazılım kodu da dahil olmak üzere kapsamlı bir veri ile güvenilirlik değerleri sayısallaştırılır. Tablo 1'de Rome Lab. Modeli Hata Yoğunluğu öngörüm metrikleri yer alır.

Tablo 1. Rome Lab. Modeli Ölçütleri

Metrik Grubu	Metrik	İsim	Ölçüt	En düşük ve en yüksek çarpan değerleri
Uygulama Tipi Metriği	A	Uygulama Tipi (Application)	Değişik tipte uygulamalar geliştirmenin getirdiği zorluk	2-14 (hata/1000 satır)
İster Metriği	D	Geliştirme Organizasyonu (Development Organization)	Geliştirme organizasyonu, proje yönetimi, metodlar, araçlar, teknikler, dokümantasyon	0.5 - 2.0
Tasarım Metrikleri	SA	Yazılım Anomali Yönetimi (Software Anomaly Management)	Hataya duyarlı tasarımın göstergesi	0.9 - 1.1
	ST	Yazılım İzlenebilirliği (Software Traceability)	Tasarım ve kodun isterlere olan izlenebilirliği	1.0 - 1.1
	SQ	Yazılım Kalitesi (Software Quality)	Kodlama standartlarına uyum	1.0 - 1.1
Gerçekleştirme Metrikleri	SL	Yazılım Dili (Software Language)	Hata yoğunluğunun yazılım diline göre normalizasyonu	1.0 - 1.4
	SX	Yazılım Karmaşıklığı (Software Complexity)	Birim karmaşıklığı	0.8 - 1.5
	SM	Yazılım Modülaritesi (Software Modularity)	Birim büyüklüğü	0.9 - 2.0
	SR	Yazılım Standartları Değerlendirmesi (Software Standards Review)	Tasarım kurallarına uyumluluk	0.75 - 1.5

Modelde yazılımın her sürecinde mevcut dokümantasyon ve veri kullanılarak yapılan değerlendirmede, tarihsel ve deneysel verilerden elde edilen çıkarımlarla hesaplanmış

katsayılar kullanılır. Denetim listelerine verilebilecek öznel cevaplar sonucu etkileyebilir. Hesaplamalarda günümüzde donanımların güçlenmesi ile artık geçerliliği kalmamış ya da elde edilmesi zor bazı parametreler (örneğin işlemci için milyon saniyede gerçekleştirilen işlem sayısı (MIPS) değerleri, artık kullanılmayan diller) bulunması da modelin revize edilmesi ihtiyacını doğurmaktadır. Kullanılan kodlama standardının kalitesi de modele önemli bir katsayı ile girdi sağlayan kriterdir. Ancak bu modelin getirdiği en büyük katkı, yazılım kodu üzerinde yapılan satır sayısı ve çevrimsel karmaşıklık ölçümlerinin, modele nesnel, ölçülebilir bir girdi sağlamasıdır.

Tablo 1'de yer alan SX metriğinin hesaplanmasında McCabe çevrimsel karmaşıklık metriğinden faydalanılır. SM metriğinin hesaplanmasında ise birim kod büyüklüğüne bakılır.

$$SX = (1.5a + b + 0.8c) / NM \quad (2)$$

$$SM = (0.9u + w + 2x) / NM \quad (3)$$

a = çevrimsel karmaşıklığı 20'ye eşit veya daha yüksek olan metodların sayısı
b = çevrimsel karmaşıklığı 20'den az ve 6'dan fazla olan metodların sayısı
c = çevrimsel karmaşıklığı 6'dan az olan metodların sayısı
u = satır sayısı 100'e eşit veya daha düşük olan metodların sayısı
w = satır sayısı 100'den fazla veya 500'den az olan metodların sayısı
x = satır sayısı 500'den fazla olan metodların sayısı
NM = toplam metod sayısı

En eski fakat en çok kullanılan yazılım kalite metriklerinden birisini ortaya koyan McCabe, çevrimsel karmaşıklık eşiğini 10 olarak belirlemiş olup, Rome Lab. Modelinde eşik değeri olarak atanan 6 değeri katı bir kriterdir [6]. Bu model projenin ihtiyaçlarına göre katsayıları değiştirilerek kullanılabilir.

2.4 Full-Scale Yazılım Güvenilirlik Modeli

Roma Lab. Modeli'nde kriterler doksanlı yıllarda 50'ye yakın projeden elde edilen verilerle belirlenmiştir. Neufelder Rome Lab. Modeli'ne benzer ve daha kapsamlı bir model ileri sürmüştür [7]. Full Scale adlı model 3 aşamada, kapsam genişletilerek uygulanabilir. Modelde çoğunlukla günümüz yazılımlarından elde edilen çıkarımlar kullanıldığı için günceldir. Modelin yer aldığı Frestimate adlı yazılım Roma Lab. modelinin uygulanmasını da içermektedir. 2 yılda bir ücrete tabi katsayı güncellemesi yapılabilmektedir. Daha çok emniyet kritik yazılımlar için ihtiyaç olan güvenilirlik analizlerinin yapılabilmesi için tercih edilen bu yaklaşımlar, kamu yazılımlarının değerlendirilmesi için maliyetli olabilir. Bu nedenle, düşük maliyetli ya da ücretsiz açık kaynak kodlu yazılımların incelenmesi gerekmiştir.

2.5 Software Improvement Group (SIG) Bakım Yapılabilirlik Modeli

Yalnızca kodun incelenmesi ile yazılımın bakım yapılabilirliğini derecelendiren SIG Bakım Yapılabilirlik Modeli, oldukça pratik ve herkes tarafından anlaşılabilir bir çözüm sunar [8]. Avrupa'da sertifkasyon kuruluşu TüvIT bu modeli kullanarak yazılımlara Bakım Yapılabilirlik sertifikası vermektedir [9]. Yazılımın projesinin

büyüklüğü, yazılım birimlerinin büyüklüğü ve karmaşıklığı, kod tekrarı oranı ve birim testlerinin kodun ne kadarını kapladığı (coverage) ölçülerek, yazılım analiz edilebilirlik, değiştirilebilirlik, test edilebilirlik ve kararlılık kriterleri yönünden derecelendirilir. Nihai sonuç ise yazılım bakım yapılabilirliğinin 1 (en düşük) ila 5 (en yüksek) arasında bir bakım yapılabilirlik notu ile ifade edilebilmesidir. SIG Bakım Yapılabilirlik Modeli açık kaynak kodlu bir yazılım kalitesi aracı olan Sonar'da eklenti olarak yer almıştır. SIG modelinin yazılım büyüklüğüne ve karmaşıklığa yaklaşımı Rome Lab. Modeli'nden biraz farklıdır. Rome Lab. Modeli'nde eşik değerleri aşan metodların sayısı tüm metod sayısına oranlanırken, SIG'de birimlerin satır sayıları, genele oranlanmaktadır. SIG modelinin yazılım büyüklüğü, kod tekrarları ve test kapsaması kriterleri Tablo 2, birim kod karmaşıklığı Tablo 3, birim kod büyüklüğü ise Tablo 4'deki derecelendirmeye göre yapılır.

Tablo 2. SIG Bakım Yapılabilirlik Modeli seviyelendirme

Seviye	Yazılım Büyüklüğü	Kod Tekrarları	Test Kapsama
1	> 1310000	> 20%	> 0%
2	> 655000	> 10%	> 20%
3	> 246000	> 5%	> 60%
4	> 66000	> 3%	> 80%
5	> 0	> 0%	> 95%

Tablo 3. SIG Bakım Yapılabilirlik Modeli karmaşıklığın seviyelendirmeye etkisi

Kritiklik seviyesi	Karmaşıklık
Very high	> 50
High	> 20
Medium	> 10
Low	> 0

Kritiklik seviyesi	Birim Satır Sayısı
Very high	> 100
High	> 50
Medium	> 10
Low	> 0

Seviye	Medium	High	Very High
1	Seviye 2,3,4 ya da 5 değilse		
2	< 50%	< 15%	< 5%
3	< 40%	< 10%	< 0%
4	< 30%	< 5%	< 0%
5	*< 25%*	*< 0%*	*< 0%*

2.6 Oman Bakım Yapılabilirlik İndeksi

SIG modelinin geliştirilmesinde en önemli motivasyon ISO/IEC 9126, ISO/IEC 25010 gibi standartların yazılım kalitesinde yönelik nesnel, ölçülebilir girdiler sağlayamayışı ve Oman Bakım Yapılabilirlik İndeksi'nin ayrıştırıcı bir sonuç vermemesi olmuştur [8][10]. Oman bakım yapılabilirlik metriği satır sayısı, yorum satır sayısı, karmaşıklık ve halstead metriklerini birlikte değerlendiren matematiksel bir modeldir [11]. Günümüz yazılım paradigmaları için geçerli sonuçlar üretememektedir. C++, Java gibi yüksek seviyeli dillerde Halstead metriğinin kullanımı anlam ifade etmemektedir

[12]. Düşük kaliteli yazılımlarda bile yüksek bakım yapılabilirlik notları ortaya çıkmaktadır. Eşik değeri olan 85'in altında çıkan proje sayısı yok denecek kadar az olmuştur. Model en çok yorum satır sayısının bakım yapılabilirliğine etkisini kıyaslamada etkilidir (Şekil 1).

$$MI_3 = 171 - 5.2\ln(V) - 0.23V(g) - 16.2\ln(LOC)$$

Halstead McCabe LOC

$$MI_4 = MI_3 + 50\sin\sqrt{2.46\,perCM}$$

% comments

Şekil 1. Oman Bakım Yapılabilirlik İndeksi'nin hesaplanması

2.7 Sonar Toplam Kalite Modeli (Sonar Total Quality Plugin - TQ)

Sonar'da kalite değerlendirmeye yönelik entegre edilmiş pratik modeller SIG ile sınırlı değildir. Çok sayıda eklenti arasından Quality Index, Total Quality Plugin ve SQALE eklentileri de kod üzerinden değerlendirme yapmaktadır. Sonar bu eklentilerin genişleştilmiş versiyonlarını ticari sürümüne dahil etmektedir.
Total Quality modeli de SIG modeli gibi farklı açılardan kodu incelemeyi amaçlar. İncelenen diğer modellere göre ilave olarak mimari karmaşıklığın, nesne yönelimli tasarım metriklerinin ve birim test başarı oranının kullanıldığı görülmektedir. Ancak kullanımdan kaldırıldığı için bazı parametreler hakkında belirsizlikler vardır. Total Quality (4)'deki gibi hesaplanır.

TQ = 0.25*ARCH + 0.25*DES + 0.25*CODE + 0.25*TS (4)

Mimari: İlk defa mimariye yönelik bir metrik tanımlanmış olsa da Tangle Index'in ne olduğu ve nasıl hesaplandığı literatürde bulunmamaktadır.

ARCH = 100 – TI (Karmaşıklık indeksi) (5)

Tasarım: Nesne Yönelimli programlama paradigması ve Chidamber&Kemerer metriklerinin kullanılması açısından da olumlu bir örnektir [13]. Ancak burada da acel adındaki ivmeleme faktörünün ne amaçla kullanıldığı ve nasıl ayarlanabileceği net değildir.

DES=0.15*NOM+0.15*LCOM+0.25*RFC0.25*CBO+0.20*DIT (6)
NOM = (1 - (class_complexity - 12) / (acel * 12)) * 50 + (1 - (method_complexity - 2.5) / (acel * 2.5)) * 50
LCOM = (1 - (lack_of_cohesion_of_method - 1) / (acel * 1)) * 100
RFC = (1 - (response_for_class - 50) / (acel * 50)) * 100
CBO = (1 - (efferent_coupling - 5) / (acel * 5)) * 100
DIT = (1 - (depth_of_inheritance_tree - 5) / (acel * 5)) * 100

Birim Kod Büyüklüğü: Bu metrik yorum satırlarını ve tekrar kod yoğunluğuna baktığı gibi Sonar aracı içerisinden hesaplanacak kural uyumluluğu indeksini de formülasyona ekler.

Code = 0.15*DOC + 0.45*RULES + 0.40*DRYNESS (7)

DOC = Belgelenmiş API yoğunluğu (yorum satırları)
RULES = Kurallara uygunluk indeksi
DRYNESS = 100 – tekrarlanan kod yoğunluğu

Birim Test Kapsaması: Burada birim test kapsama oranı %80 etki ettiği gibi ilk defa birim test başarım oranı kullanılmıştır.

Test = 0.80*COV + 0.20*SUC (8)
COV = Test kapsama oranı
SUC = Birim test başarım oranı

2.8 Sonar Kalite Endeksi Modeli (Sonar Quality Index Plugin- SQI)

Sonar Kalite Endeksi Modeli, PMD, CheckStyle gibi açık kaynak kodlu güçlü kalite araçlarından faydalanmaktadır. Bu model statik kod analizi ile uygulama açıklarını değerlendirir, kodlama stili yanlışlıklarını puanlamaya katar. Modelin bileşenleri aşağıdaki gibidir:

Uygulama Açıkları Analiz bulguları: PMD aracı ile analiz edilen ve listelenen hataların kritikliğine göre puanlama yapılır ve tekrarlanmış kodları elimine ederek bulduğu geçerli satır sayısı değeri ile puanlama normalize edilir. Blocker türünde bir hata, 10 adet Minor ya da Info türünde hataya eşdeğer Kabul edilmiştir. Statik kod analizinde en büyük sorunlardan birisi "false positive" adı verilen yanlış tespitlerin çokluğudur. Dolayısı ile Info ve Minor türünde yanlış şekilde oluşturulan yüzlerce hata çıkabilir ve sonucu yanlış etkileyebilir. Info ve Minor'un hesaplamaya katılmaması önerilir.

Coding = (Blocker * 10 + Critical * 5 + Major * 3 + Minor + Info) / ValidLines (9)
ValidLines: Toplam satır sayısı – Tekrarlanan kod satır sayısı
Blocker: PMD Blocker türünde hata sayısı
Critical: PMD Critical türünde hata sayısı
Major: PMD Major türünde hata sayısı
Minor: PMD Minor türünde hata sayısı
Info: PMD Info türünde hata sayısı

Birim Kod Karmaşıklığı: Karmaşıklığa yaklaşım daha önce incelenen modellerdekine benzemekle birlikte eşik değerlerince ve normalizasyon yaklaşımında farklılık bulunmaktadır.

Complexity = (Complexity>30 * 10 + Complexity>20 * 5 + Complexity>10 * 3 + Complexity>1) / ValidLines (10)

Birim Test Kapsaması: Birim test kapsama oranı doğrudan kullanılır.

Kodlama Stili: CheckStyle aracı ile listelenen hataların kritikliğine göre puanlama yapar ve tekrarlanmış kodları elimine ederek bulduğu geçerli satır sayısı değeri ile puanlamayı normalize eder.

Style = (Errors * 10 + Warnings) / ValidLines * 10 (11)
Errors: CheckStyle ile tespit edilen Blocker ve Critical türünde hataların toplamıdır.
Warnings: CheckStyle ile tespit edilen Major, Minor ve Info türünde hataların toplamıdır.

Coding, *Complexity*, *Coverage* ve *Style* bileşenlerini kullanarak PMD hatalarının en fazla etki ettiği kalite endeksi (10) ve karmaşıklığı 30'dan fazla olan metodların sayısının oranına göre hesaplanan karmaşıklık faktörü (11) adlı iki adet kalite ölçütü üretilir.

Quality Index = 10 - 4.5 * Coding - 2 * Complexity - 2 * Coverage - 1.5 * Style (12)

Complexity Factor = (5 * Complexity>30) * 100 / (Complexity>1 + Complexity>10 + Complexity>20 + Complexity>30) (13)

Bu model de Sonar'ın sonradan eklediği modeller nedeniyle destek vermeyi bıraktığı eklentiler arasında yer almıştır.

2.9 Yazılım Kalite Risk Oranı (YKRO)

YKRO, yazılım kalite metrik ölçüm sonuçlarını kullanarak bir her metod ve sınıf için yazılımın toplam kalitesini ne derece etkilediğini sayısal olarak belirlemeye dayanan bir modeldir [12]. Metod bazında ve sınıf bazında ayrı analizler gerçekleştirilir. Metodların ve sınıfların YKRO toplamları 100'ü verir. Bu nedenle kalite problemleri projeye mi yayılmış, belirli yerlerde mi yoğunlaşmış soruları YKRO analizi ile cevaplanabilir.

Bir metot ya da sınıfın YKRO değeri Formül (14) ve Formül (15) e göre hesaplanır.

$$Td_j = \sum (MV_{ij} - Thr_j) \quad (14)$$

$$YKRO_i = \sum ((MV_{ij} - Thr_j) / Td_j * 100) * C_j \quad (15)$$

Formül (14) ve Formül (15) de i metot ya da sınıf numaralandırıcısıdır, j metrik tipi, Td_j metot ya da sınıfın risk değeri, MV_{ij} metrik ölçüm sonucu, Thr_j seçilen metriğin eşik değeri ve C_j seçilen metriğe verilen katsayıdır. MV_{ij} - Thr_j değerinin Thr_j´in MV_{ij} den büyük olması durumunda 0 olarak kabul edilmesi gereklidir.

3 Tartışma ve Öneriler

Bir önceki bölümde incelenen yazılım kalite modellerinin bir kısmının çoğunlukla dolaylı ölçülebilen kriterlere dayandığı, bir kısmının da ticarileşerek maliyetinin arttığı gözlenmektedir. Örneğin; sayılan Sonar eklentilerinin tamamı belli bir süre sonra daha kapsamlı başka bir modelle yer değiştirmiş ve son modellerden olan SQALE modelinin ileri özellikleri ise ücretli olarak sunulmuştur. SQALE'in benzeri bir model olan SQuARE ise ISO/IEC 9126 standardının kalite faktörlerini göze hitap eden bir arayüz paneli ile sunan ücretli bir yazılımdır.

Kamunun ihtiyacı ücretsiz ve pratik bir çözümdür, bazılarının kullanımı için neredeyse uzman olmak gereken bu modelleri anlaşılır şekilde sunmak önemli bir fayda sağlayacaktır. Bu amaçla model incelemesinden elde edilen çıkarımlar kullanılarak, yazılım kodundan elde edilebilen tüm ipuçları çekilecek ve kapsamlı ama pratik bir model önerilecektir. İncelenen modellerin baz aldığı kriterler Tablo 5'de özetlenmiştir.

Tablo 4. Yazılım Kalite Modelleri ve İncelenen Girdiler

	DOĞRUDAN ÖLÇÜLEBİLEN KRİTERLER									DOLAYLI
	Ürün Büyüklüğü	**Birim Büyüklüğü**	**Halstead metrikleri**	**Birim Karmaşıklığı**	**Kod Tekrarları**	**Statik Kod Analizi Bulguları**	**Birim test kapsama**	**Tasarım metrkleri**	**Kodlama Stili**	**Denetim Listeleri**
McCall										X
25010										X
ROME		X		X					X	X
Full-scale										X
SIG	X	X		X	X		X			
Oman	X		X	X						
TQI		X					X	X		
SQI		X		X	X	X	X		X	
YKRO		X		X				X		

4 Önerilen Model

Pratik kalite modellerinin incelenmesi ve tartışılması ışığında, YTKDL yazılımları aşağıdaki kriterlere göre değerlendirecek, bir model ve uygulama geliştirmiştir:

- Birim metod satır sayısı ve karmaşıklığı ifade etmede SIG bakım yapılabilirlik metriği oldukça kolay anlaşılmaktadır. Bu nedenle sadece birim satır sayıları geçiş aralıkları Tablo 6a'daki gibi uyarlanarak aynen kullanılmıştır. SIG modeli, YTKDL'deki yazılım kalite değerlendirmesi tecrübelerine dayanarak, birim büyüklüğü karşısında birim karmaşıklığının önemini artıracak şekilde güncellenmiştir.
- İncelenen modellerde tasarıma yönelik bir incelemenin yapıldığı tek model vardır ve orada da modelin etkinliği kanıtlanmamıştır. Kamu yazılımları için tasarımla ilgili yeni bir yaklaşım önerilecektir. YTKDL'de kullanılmakta olan türetilmiş metrik Yazılım Kalite Risk Oranı (YKRO) kullanılacaktır [12]. Buna gore YKRO değeri en yüksek 3 metodun toplamına göre Tablo 6b'deki sınıflandırma kullanılmakta ve bakım yapılabilirlik notuna eklenmektedir. En bozuk ilk 3 sınıfın YKRO toplamı %20'den yüksekse, bu tasarımın iyi yapılmadığı anlamına gelir ve bakım yapılabilirlik notu 1 olur. Belirlenen YKRO seviyelendirmesi notu 1. Adımda anlatılan notlandırmaya eklenir ve 10 üzerinden bir puan elde edilir.
- Kamu yazılımlarında görülen en büyük eksikliklerden birisi müşteri isteklerinin tam anlaşılamayışı, mimari tasarımın yapılmaması ve müşteriye onaylatılmaması, daha sonradan ortaya çıkan isteklerin mimariye etkisi tartılmadan kodlanması ve yeterince test edilmeden kullanıma alınmasıdır. Bunun sonucunda çok yerde kod tekrarı olmakta, bunlar tespit edildiğinde tasarımın baştan aşağı değişmesi gerekmekte ancak riskler içerdiğinden dolayı müdahale edilmemektedir. Diğer yandan birim test alışkanlığı da yok denecek kadar azdır. Çoğu kamu yazılımında kod tekrarı %20'nin üzerinde ve birim test kapsaması da %0 olacağı için, bu kriterlerin ayrıştırıcı sonuçlar vermeyeceğinden dolayı kullanılması tercih edilmemiştir.

Tablo 5. Önerilen Modelde – a) birim satır sayısının seviyelendirmeye etkisi b) kalite risk oranının seviyelendirmeye etkisi

(a)

Kritiklik seviyesi	Birim Satır Sayısı
Very high	> 200
High	> 100
Medium	>50
Low	> 0

(b)

Seviye	Kalite risk oranı
1	> 20%
2	> 10%
3	> 5%
4	> 3%
5	> 0%

- Yazılımın hata yoğunluğunu öngörmek için Rome Lab. Modeli yaklaşımı yardımcı olmaktadır, ancak metod sayılarına gore değil kod satır sayısına gore yapılan hesaplama daha doğru sonuç verecektir, bu nedenle yöntem ve katsayılar aşağıdaki gibi revize edilmiştir. SX ve SM değerleri 100 ile çarpılarak hata yoğunluğu bulunabilir. Başlangıç hata yoğunluğu 10 hata/1000 satır varsayılırsa 1000 * SX * SM değeri yazılımın barındırdığı öngörülen (predicted) toplam hata sayısını verecektir.

$$SX = (1.5a + b + 0.8c) / TLOC \quad (16)$$

$$SM = (0.9u + w + 2x) / TLOC \quad (17)$$

a = çevrimsel karmaşıklığı 20'ye eşit veya daha yüksek olan metodların toplam satır sayısı
b = çevrimsel karmaşıklığı 20'den az ve 10'dan fazla olan metodların toplam satır sayısı
c = çevrimsel karmaşıklığı 10'dan az olan metodların toplam satır sayısı
u = satır sayısı 200'e eşit veya daha düşük olan metodların toplam satır sayısı
w = satır sayısı 200'den fazla veya 500'den az olan metodların toplam satır sayısı
x = satır sayısı 500'den fazla olan metodların toplam satır sayısı
TLOC = toplam satır sayısı

- Statik kod analizi ve kodlama stili analizinin kalite modeline etkisini formülize etme çalışmaları devam etmektedir. Doğru kullanımı yanlışmış gibi gösteren (“false positive”) hataların otomatik olarak ayıklanması çözülmesi gereken önemli bir problem olarak durmaktadır. “Minor” ve “Info” gibi hata türlerini hesaplamaya hiç katmamak bir çözüm olabilir.

Mevcut modellere ve önerilen modellere göre yapılan sınıflandırmaları karşılaştırmak için Şekil 2’de bir örnek sunulmaktadır. 8 adet açık kaynak kodlu Java yazılım kalitesi aracının (JUnit, Findbugs, UCDetector, Xradar, Tattletale, Cobertura, JDepend, QALab) kodları 3 adet mevcut, 2 adet önerilen modele gore değerlendirilmiştir.
Oman ve SIG bakım yapılabilirlik modelleri sonuçları önerilen bakım yapılabilirlik modeli sonuçları ile, Rome Lab. Güvenilirlik modeli sonuçları ise önerilen Güvenilirlik modeli sonuçları ile karşılaştırılmıştır.

	BAKIM YAPILABİLİRLİK MODELLERİ			GÜVENİLİRLİK M.		
	Oman modeline göre	SIG modeline göre	Önerilen modele göre	Rome modeline göre	Önerilen modele göre	
1	UCDetector	Junit	Junit	Junit	Junit	1
2	Junit	QALab	QALab	Cobertura	JDepend	2
3	JDepend	JDepend	JDepend	QALab	QALab	3
4	Xradar	Cobertura	UCDetector	JDepend	UCDetector	4
5	Cobertura	UCDetector	Cobertura	Xradar	Cobertura	5
6	QALab	Xradar	Xradar	UCDetector	Xradar	6
7	Findbugs	Findbugs	Findbugs	Findbugs	Findbugs	7
8	Tattletale	Tattletale	Tattletale	Tattletale	Tattletale	8
	133	5	7	28,0	28,0	
	132	5	7	28,0	28,0	
	128	5	6	28,0	25,4	
	124	4	6	28,0	24,4	
	120	4	5	27,1	18,7	
	118	3	4	27,1	18,3	
	116	1	3	26,2	7,9	
	99	1	2	24,5	-1,0	

Şekil 2. Mevcut ve önerilen modellere gore açık kaynak kodlu yazılımlar üzerinde örnek bir ölçüm karşılaştırması ve yazılımların kalitesine göre sıralanması

Karşılaştırma sonuçları aşağıdaki gibi sıralanabilir:

- Toplamda 5 farklı model aynı yazılımlar üzerinde denenmiştir, UCDetector 4 modele gore en iyi derece olarak 4.lüğe yerleşirken, Oman'da birinci çıkmıştır. Bu durum Oman değerlendirmesinde yorum satır sayısına yüksek puan verilmesinden ve daha sonra kullanılmak üzere yorum satırı haline getirilmiş kodların fazlalığından kaynaklanmaktadır.
- SIG modelinde "bakım yapılamaz" (1) notu alan yazılımlar bile Oman metriğine gore "bakım yapılabilirlik açısından başarılı" eşiği olan 85'in üzerindedir. Bu durum Oman modelinin bakım yapılabilirlik durumu açısından kullanıcıyı yanılttığını ortaya koymaktadır. Oman modeline gore üretilen puan, kullanıcıya hangi iyileştirmeleri yaparsa kaliteyi artıracağına yönelik ipucu sunmamaktadır. Halbuki SIG modeli ve ondan esinlenerek önerilen modelde yazılımın hangi iyileştirmeleri yaparsa kalitesini artıracağı nettir.
- SIG ile "mükemmel bakım yapılabilir" çıkan yazılımlar, önerilen bakım yapılabilirlik modelinde, 10 üzerinden 7 puan almıştır. Önerilen modeldeki YKRO Tasarımsal bileşeni ayırdedicilik sağlamıştır. SIG modeline gore 5 alan yazılımlar YKRO modeline gore 2 almıştır. Bu yazılımlardan 3 almaya en yakın olan yazılım JUnit olup, problemlerin yoğunlaştığı ilk 3 metod üzerindeki %2'lik bir iyileştirme notunu 3'e yükseltecektir.
- En yüksek 3 YKRO değeri toplanarak hesaplanan tasarım kriterinde, yüzdelik aralıkları laboratuvarımızda edinilen tecrübeler ışığında belirlenmiştir. Bu aralık değerleri deneysel çalışmalar ile revize edilecektir. Bu ilk kullanımda 5 üzerinden 2'yi aşan not alabilen yazılım çıkmamıştır.
- QALab ve Findbugs sonuçları Oman modeline gore neredeyse aynı çıkarken SIG modelinde büyük fark (2 basamak) var gözlenmiştir. Önerilen model, QALab ve

Findbugs arasındaki farkı 1 basamağa indirmiştir. SIG modeline bakılarak cope atılması beklenen bir yazılımın iyileştirilebileceği yönünde bir sonuç ortaya çıkmıştır.

- Rome modeline gore en kaliteli ve en kalitesiz yazılım arasında sadece %3.5 güvenilirlik farkı görünmektedir. Halbuki, ilk sıradaki JUnit ile son sıradaki Tattletale arasında çok belirgin farklar tespit edilmiştir. JUnit'te karmaşıklık değeri 20'yi aşan kod oranı %1 bile değil iken, TattleTale'in kodunun neredeyse yarısının karmaşıklığı 20'nin üzerindedir. Önerilen güvenilirlik modeli daha ayrıştırıcı, farkları daha net görmeyi sağlayan sonuçlar vermiştir. JUnit ile TattleTale arasında %29'luk fark (+%28 ile -%1) hesaplanmıştır. Aynı şekilde önerilen bakım yapılabilirlik modeli tasarımla ilgili ölçütleri de hesaba kattığından,
- Çıkarılabilecek bir sonuç da asıl amacı yazılım kalitesini değerlendirmek olan araçlarda da yazılım kalitesi yönünden sorunlar olduğunun görülmesidir.

Önerilen modelde metrik ölçümleri için, laboratuvar imkanları dahilinde McCabe IQ, Understand gibi ticari araçlara ilave olarak Eclipse Metrics eklentisi gibi ücretsiz yazılımlar da kullanılabilmektedir. Sonuç olarak ortaya ücretsiz açık kaynak kodlu yazılımlardan faydalanan ve genel kabul görmüş pratik modellerden uyarlanan, kamu yazılımlarında pratik değerlendirme imkanı sağlayarak ihtiyacı karşılayan bir model ve yazılım geliştirilmiştir. Araçlar tarafından otomatik olarak üretilen ve YTKDL'de üretilen yazılım kalite değerlendirme raporları ilk başlarda çok büyük hacimli ve genelde uygulanması zor öneriler içeren dokümanlar iken, önerilen kalite modeline ait sonuçlar daha anlaşılır ve uygulanabilir maddeler içermektedir.

5 Sonuç

Yazılım kalitesinin, bakım yapılabilirliğinin, güvenilirliğinin kısa sürede değerlendirilmesi, çıkan sonucun da herkes tarafından kolaylıkla ve aynı şekilde anlaşılabilmesi önemli bir ihtiyaç olmuş, bu amaçla birtakım modeller geliştirilmiştir. Bu modeller incelendiğinde, incelenen faktörlerin, ölçütlerin, değerlendirme yöntemlerinin farklılıklar içerdiği görülmektedir. Bu çalışmada, geliştirme süreci bitmiş ya da sonuna yaklaşılmış kamu yazılım projelerinde, hızlı ve anlaşılır bir kalite değerlendirmesinin, mevcut kalite modellerinden faydalanılarak nasıl gerçekleştirilebileceği araştırılmıştır. Sonuç olarak satır sayısı ve karmaşıklık ölçütlerinin uyarlanarak kullanıldığı, tasarım ölçütlerinin yeni bir yaklaşımla ele alındığı, kod tekrarı ve birim test kapsaması ölçütlerinin ise kamu yazılımlarında ayrıştırıcı olmayacağı için kullanılmadığı bir model önerilmektedir. Önerilen model, özellikle, herhangi bir sürece uygun olarak geliştirilmemiş düşük kaliteli yazılımlar için anlık resim çekmekte ve yazılımın geleceği için karar verici olan mercilere faydalı bir girdi sağlamaktadır.

Teşekkür - Yazarlar, bu çalışmanın gerçekleştirilmesi için destek sağlayan TÜBİTAK BİLGEM Yazılım Test ve Kalite Değerlendirme Laboratuvarı'na teşekkür eder.

Kaynaklar

1. Jones, C., “Software Quality Metrics: Three Harmful Metrics and Two Helpful Metrics”, *Technical Report*, 2012
2. Rome Laboratory, Reliability Engineer’s Toolkit, *Technical Report*, 1993.
3. ISO/IEC 25010:2011 - Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models, http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733
4. McCall, J.A, Richards, P.K., Walters, G.F,”Factors in Software Quality”, *Final Tech Report, RADC-TR-77-369*, Rome Air Development Center,Griffith Air Force Base, 1977
5. ISO/IEC 9126-1:2001, Software engineering -- Product quality -- Part 1: Quality model, http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749
6. McCabe, T. J. “A Complexity Measure”, *IEEE Trans. Software Eng*. SE-2, 4 (Dec. 1976), 308-320.
7. Neufelder, A.M, “Predict Software Reliability Before the Code is Written”, *Technical Report*, SoftRel, LLC , 2013.
8. Heitlager, I., Kuipers, T. and Visser, J., “A Practical Model for Measuring Maintainability”. *6th International Conference on the Quality of Information and Communications Technology*, 2007, pp. 30-39.
9. Baggen, R., Schill, K., Visser, J.,”Standardized Code Quality Benchmarking for Improving Software Maintainability”, *14th European Conference on Software Maintenance and Reengineering*, March 15-18, 2010 in Universidad Rey Juan Carlos, Madrid, Spain
10. Oman, P., and Hagemeister, J., Construction and testing of polynomials predicting software maintainability. *In Journal of Systems and Software*, 1994, vol. 24(3), pp. 251-266.
11. Halstead, M. H. "Elements of Software Science", *New York: Elsevier North-Holland*, 1977.
12. Palıgu F., Öztürk, S., Yağcı, N., “Kodu İyileştirmeye Nereden Başlamalı? Bir Yazılım Metrik Yaklaşımı: Yazılım Kalite Risk Oranı”, *7th Turkish National Software Engineering Symposium*, İzmir, 2013
13. Chidamber, S. R. ve Kemerer, C. F., “A Metrics Suite for Object-Oriented Design”, *IEEE Trans. Software Eng.*, vol. 20, no. 6, June 1994, pp. 476-493.