Journal of Information Systems and Management Research
Bilişim Sistemleri ve Yönetim Araştırmaları Dergisi

http://dergipark.gov.tr/jismar

Araştırma Makalesi / Research Article

Yazılım Hata Kestirimine Nesne Yönelimli Ölçütlerin Etkisi

Tülin ERÇELEBİ AYYILDIZ[*a], Begüm ERKAL [b]

[a,*] *Başkent Üniversitesi, Bilgisayar Mühendisliği Bölümü, ANKARA, TÜRKİYE*
[b] *Başkent Üniversitesi, Bilgisayar Mühendisliği Bölümü, ANKARA, TÜRKİYE*

MAKALE BİLGİSİ

Alınma: 24.10.2019
Kabul: 25.12.2019

Anahtar Kelimeler:
Nesne Yönelimli Ölçütler, Yazılım Hata Kestirimi, Yazılım Ölçütleri

Sorumlu Yazar
e-posta:
ercelebi@baskent.edu.tr

ÖZET

Yazılım projelerinin sonucunda amaç sadece doğru çalışan bir ürün çıkarmak değildir. Gerçekleştirilen yazılımın kalitesel anlamda değerlendirilmesi ve kalitenin ölçülmesi de gerekmektedir. Yazılım ne kadar kaliteli olursa bakım onarım aşamasındaki maliyetler de o kadar azalacaktır. Yazılım kalitesini etkileyen önemli noktalardan biri yazılımdaki hataların sayısıdır. Bu sebeple geliştirilen yazılımlardaki hataların erken safhada tespit edilebilmesi oldukça önem taşımaktadır. Bu çalışmanın amacı; geliştirilmiş açık kaynak kodlu yazılım projelerindeki hata sayısının belirlenmesi ve belirlenen hata sayıları ile yazılım kalite ölçütleri arasındaki ilişkinin analiz edilmesidir. Bu amaçla, 20 adet açık kaynak kodlu java programlama diliyle geliştirilmiş oyun projeleri veri seti olarak kullanılmıştır. Yazılım kalite ölçütlerinin analizinde Understand adı verilen statik kod analiz aracı kullanılmıştır. Projelerdeki hata sayılarını tespit etmek için Spotbugs hata tespiti aracından faydalanılmıştır. Yazılım hataları ve yazılım kalite ölçütleri arasındaki ilişkiyi çıkarabilmek için doğrusal regresyon yöntemi uygulanmıştır. Analiz sonucunda çıkan modelin kestirim doğruluğu yapılmıştır. Sonuçlar, yazılım hata sayısını tahmin etmek için yazılım kalite ölçütlerinden yararlanmanın mümkün olduğunu göstermektedir.

The Effect of Object Oriented Metrics on Software Bug Prediction

ARTICLE INFO

Received: 24.10.2019
Accepted: 25.12.2019

Keywords:
Object Oriented Metrics, Software Bug Prediction, Software Metrics

Corresponding Authors
e-mail:
ercelebi@baskent.edu.tr

ABSTRACT

At the end of the software projects, the aim is not only to produce a product that works correctly. It is also necessary to evaluate the quality of the software performed and to measure the quality. The higher the quality of the software, the lower the costs of maintenance and repair. One of the important points that affect software quality is the number of bugs in software. For this reason, it is very important to detect bugs in the software developed at an early stage. The aim of this study; to determine the number of bugs in developed open source software projects and to analyze the relationship between identified bug numbers and software quality metrics. For this purpose, 20 open source game projects developed with java programming language were used data sets. Static code analysis tool called Understand was used to measure software quality metrics. Spotbugs bug detection tool was used to determine the bugs in the projects. Linear regression was used to determine the relationship between software bugs and software quality metrics. The prediction accuracy of the model was obtained. The results show that it is possible to use software quality metrics to estimate the number of software bugs.

1. GİRİŞ *(INTRODUCTION)*

Günümüzde teknolojinin de ilerlemesi ile birlikte yazılım dünyası büyük gelişmeler göstermektedir. Bu gelişimlere rağmen geliştirilen yazılımlarda bazı hatalara rastlanmaktadır. Hatalı modüller geliştiricilerin proje üzerinde daha fazla zaman harcamasına sebep olmaktadır. Piyasaya çıkan yazılımların belli aralıklarla güncellenmesi bu hataların giderilmesi için kullanılan yöntemlerden biridir. Yazılım hataları; yazılım güvenilirliğini, yazılım kalitesini ve bakım maliyetini önemli ölçüde etkilemektedir [1]. Bu hataların erken aşamada tespit edilebilmesi oldukça önemlidir. Yazılımdaki hataların erken tespit edilmesi hem daha kaliteli bir yazılım hem de müşteriye teslim edilecek yazılımın daha az hata içermesini sağlar. Boehm [2]'e göre, geliştirilen bir yazılımın müşteriye teslim edilmesinden sonra ortaya çıkan maliyet, geliştirme sürecinde ortaya çıkacak olan maliyetten neredeyse yüz kat daha büyük olabilmektedir. Hatalar tespit edilirken genellikle test araçları kullanılır ve yazılım ürününün kullanıma uygunluğunun doğrulanması sağlanır. Bazen test araçları da hataların yakalanmasında yeterli olmamaktadırlar. Bu nedenle yazılım geliştirme sürecinde hataları belirleyebilecek alternatif yöntemlere ihtiyaç duyulabilmektedir [3]. Bu ihtiyaç ışığında bu çalışmaya konu olan araştırmada, yazılım geliştirme sürecinde geliştiricilerin hata sayısını önceden yazılım kalite ölçütlerine dayalı öngörebilmelerini sağlayan bir kestirim yöntemi geliştirilmesi amaçlanmıştır.

2. ÖLÇÜT TABANLI YAZILIM HATA KESTİRİM YAKLAŞIMLARI *(METRIC BASED SOFTWARE BUG PREDICTION APPROACHES)*

Yazılım ölçütleri, yazılımların ölçülebilen ya da yapılan ölçümler sonucuna göre hesaplanan değerlerine denilmektedir [4].

Yazılım ölçütleri ile geliştirilen yazılımlardaki hatalı kod parçaları tespit edilebilir. Böylelikle yazılımın içerdiği modüllerdeki kod parçalarının hangisinin daha önce test edilmesi gerektiği, hangi bölümlere daha önem vererek yazılması gerektiği gibi önemli gereksinimler karşılanır. Literatürde hata kestirimini belirlemede çeşitli sayıda yazılım ölçütleri yer almaktadır. Bu ölçütler belli hata kestirim yaklaşımlarına göre gruplandırılmaktadır. Tablo 1'de hata kestirim yaklaşımları özetlenmiştir [4-8]. Hata kestiriminde en yaygın olarak kullanılan yaklaşım kaynak kodu ölçütleridir. Bilinen Kaynak Kod ölçütleri; CK Ölçütleri, Halstead Ölçütleri, Kod Satırı (LOC) Ölçütü, Mccabe Karmaşıklık Ölçütleridir [9].

Tablo 1. Yazılım Hata Kestirim Yaklaşımları
(Software Bug Prediction Approaches)

Tür	Hata Gerekçesi	Kimin Tarafından Kullanıldığı
Süreç Ölçütleri	Hatalar değişikliklerden kaynaklanır.	[5]
Önceki Kusur Ölçütleri	Geçmiş kusurlar gelecekteki kusurları öngörür.	[6]
Kaynak Kodu Ölçütleri	Karmaşık değişiklikler daha fazla hataya açıktır	[7]
Değişikliklerin Entropisi Ölçütleri	Kaynak kod ölçütleri, kod karmaşasının daha iyi kestirilmesini sağlar.	[8]
Kod Çalkantısı Ölçütleri	Kaynak kod ölçütleri, kod karmaşasının daha iyi kestirilmesini sağlar.	Novel
Entropi Ölçütleri	Kaynak kod ölçütleri, değişikliklerin entropisini daha iyi tanımlar.	Novel

Kod satırı; 1960'ların sonlarında kullanılmıştır [10]. Kod Satırı (LOC); uygulamanın boyutunu hesaplamak için kullanılan en eski ve basit yazılım ölçütlerinden biridir. 1970'lerin sonunda ise Mccabe ve Halstead ölçütleri kullanılmaya başlanmıştır. 1974 yılında Thomas Mccabe tarafından ise kendi adını verdiği Mccabe Karmaşıklığı Ölçütü öne sürülmüştür [11]. Bu ölçütler 1970'lerden 1990'ların başına kadar uzanan süre boyunca üzerinde en çok çalışılan ölçütler olmuştur. Li ve Henry [12], ile Chidamber ve Kemerer [13], tarafından yapılan araştırmalar sonucunda önerilen nesne yönelimli ölçütler popüler hale gelmiştir. Chidamber ve Kemerer (CK) Ölçütleri; 1994 yılında CK Metrik Suite başlığı altında 6 tane sınıf tabanlı ölçüt önermiştir [13]. Bu ölçütler; WMC (Sınıfın ağırlıklı metot sayısı), DIT (Kalıtım ağacının derinliği), NOC (Alt sınıf sayısı), RFC (Sınıfın tetiklediği metot sayısı), CBO (Sınıflar arası bağımlılık) ve LCOM (Uyum eksikliği)'dur. Halstead Ölçütleri [14]; bir programı oluşturmak için gereken efor miktarı, farklı operand ve operatörlerin basit sayımlarından ve bunların toplam sıklıklarından elde edilebilmektedir. Karmaşıklık ölçütlerinden bir diğeri olan Mccabe Karmaşıklık Ölçütü ise; test edilmesi zor olan ve güvenilir olmayan yazılım modüllerinin bulunmasını amaçlar.

3. YAZILIM HATA KESTİRİMİ KONUSUNDA YAPILAN ÇALIŞMALAR *(STUDIES ON SOFTWARE BUG PREDICTION)*

Yazılım hata kestirimi konusunda yapılan bir çok çalışma bulunmaktadır. Çalışmalar aşağıda belirtildiği şekildedir:

D'Ambros vd. [15]; sınıf düzeyinde hata tahmini yapabilmek için veri seti tasarlamışlardır. Bu set beş sistemin; hata, değişim ve versiyon bilgilerinden oluşmaktadır. Aynı zamanda kendi oluşturdukları web sitesinde kullandıkları veri setini sunarak denemelerinin tekrar edilebilirliğini sağlamışlardır. Kullandıkları sürümlerin iki haftada bir tüm ölçüt değerleri, her sınıf için sonraki hata sayılarını elde etmişlerdir. Önceki hata raporlarının en iyi kestiriciler olduğunu bildirmişlerdir.

Zimmermann vd. [16] tarafından yapılan çalışmada; Eclipse hazır hata veri seti kullanılmış ve seçilen karmaşıklık ölçütleri ile hata tahmini yapılmaya çalışılmıştır. Kullandıkları veri setleri toplamda 6 dosyadan oluşmakta ve her seviye için dosya ve sürüm bulunmaktadır. İlk olarak sürüm öncesi ve sonrası hata sayılarını tespit etmişlerdir. Bu tespiti yaparken "BUGZILLA" hata takip sisteminden yararlanmışlardır. Hata sayıları ve veri setindeki karmaşıklık ölçütleri arasındaki "ilişki", "Spearman Korelasyonu" kullanılarak hesaplanmıştır. Dosya düzeyinde; McCabe karmaşıklığı (VG), toplam kod satırı (LOC) ölçütlerinde 0,400'ün üzerinde bir ilişki olduğu görülmüştür. Bu dosyaların hata eğilimli dosyalar olduğunu gösteren bir işaret olduğu belirtilmiştir. Paket düzeyinde ise; ölçütlerin çoğu yüksek ilişki göstermiştir. En yüksek olanların ise statik alan sayısı (NSF) ve statik metod sayısı (NSM) olduğu belirtilmiştir. Yaptıkları çalışma ile karmaşıklık ölçütlerinin hataları öngörebileceği, daha karmaşık yapıya sahip olan kodun daha fazla hata içerebileceğini öne sürmüşlerdir.

Gyimothy vd. [17] yaptığı çalışmada; "Chidamber ve Kemerer" tarafından öne sürülen nesne yönelimli ölçütlerin C++ dilinde yazılmış açık kaynak kodunda hata eğilimleriyle nasıl ilişkisi olduğunu hesaplayarak açıklamışlardır. Hata veritabanı kullanarak bulunan hata sayılarını "BUGZILLA" hata takip sisteminden elde etmişlerdir. Hangi ölçütlerin hata tahmininde kullanımının daha iyi olabileceği araştırılmıştır. CBO ölçütünün sınıfların hata eğilimlerini kestirmede en iyisi olduğunu gözlemlemişlerdir. LCOM ölçütünün de hata kestiriminde doğruluğunun iyi olduğu fakat değerinin düşük olduğunu belirtmişlerdir. DIT ve NOC ölçütlerinin hata kestiriminde güvenilmez olduğunu gözlemlemişlerdir.

4. YÖNTEM *(METHOD)*

Bu çalışmada yazılım hataları konusunda yapılan daha önceki çalışmalar incelenmiş, erişim kolaylığı açısından 20 adet açık kaynak kodlu "Java" programlama diliyle yazılmış oyun projesi veri seti olarak seçilmiştir. "Java" programlama dilinin seçilmesinin sebebi; güvenilir olması, platform bağımsızlığına sahip olması ve proje geliştirmesinin hızlı olmasıdır. Oyun projelerinin seçilme sebebi ise; günümüzde oyun projelerinin çok popüler olmasıdır. Yapılan çalışmada, yazılım geliştiricilerinin en çok kullandığı "GitHUB" ve "SourceForge" sitelerinden açık kaynak kodlu projeler indirilmiştir. Bu sitelerin tercih edilme sebepleri; hızlı ve güvenli olması, kapsamlı raporlara sahip olması ve birden fazla kişinin eş zamanlı olarak aynı proje üzerinde çalışabilmesidir [18]. Öncelikli olarak "Understand" statik kod analiz aracıyla analiz edilerek seçilen ölçütlerin değerleri elde edilmiştir.

"Understand" kod analiz aracı; kaynak kod analizi için geliştirilmiş olup yazılımı analiz ederek kaynak kodu hakkında detaylı ölçüt raporu, bağımlılık analizi, diyagramlar ve grafik bilgisi gibi bir çok bilgiyi vermektedir. Aynı zamanda birden fazla yazılım dilinde destek sağlamaktadır [19]. Ölçütler seçilirken yazılım hatası ile olan ilişkileri dikkate alınmıştır. Tablo 2' de hata ile olan ilişkileri gösterilmiştir. Tablo 2 incelendiğinde; 6 CK ölçütünün hata yoğunluğunu arttırdığına dair referanslar görülmektedir ve bu sebeple bu 6 ölçütün analizlerde yer almasına karar verilmiştir.

Tablo 2. Ölçütler ve Yazılım Hatasıyla Olan İlişkileri
(Metrics and Their Relationship with Software Bug)

Ölçüt Adı	Hata ile İlişkisi	Referans
WMC	Yüksek WMC hata yoğunluğunu arttırır ve kaliteyi azaltır	[20]
DIT	Yüksek bir DIT metriğinin hataları arttığı belirtilmiştir.	[21]
RFC	Çok sayıda metodun çağrılmasını tetiklemesi, sınıfın testinin ve hata ayıklamasının zorlaşmasıdır.	[13]
CBO	Sınıflar arası bağımlılık yükseldikçe hataların arttığı belirtilmiştir.	[13]
LCOM	Düşük uyumluluk karmaşıklığı artırır, bu nedenle geliştirme aşamasında hata yapılma ihtimali yükselir.	[22]

Seçilen projelerdeki hata sayılarını bulmak için "SpotBugs" programı seçilmiştir. "SpotBugs" hem kendi arayüzüne sahiptir hem de "Eclipse" geliştirme ortamında "plugin" modunda kurulabilmektedir. "SpotBugs" programı ile sadece "Java" dilinde yazılmış olan mantıksal hataların tespiti yapılabilmektedir [23].

"Eclipse" geliştirme ortamına, seçilen projeler yüklendikten sonra "SpotBugs" analiz aracıyla 20 projede bulunan mantıksal hatalar tespit edilip, söz dizimi hataları (syntax) dikkate alınmamıştır.
"Spotbugs" programı incelendiğinde bulduğu hataların mantıksal hatalar olduğu gözlemlenmiştir ve bu hataların gerçekten hata olup olmadığı test edilmiştir. Mantıksal hata içeren küçük bir modül geliştirilerek program test edilmiştir. Test ederken bir sonsuz döngü örneği yazıldığı zaman programın bu hatayı tespit ettiği ve ne tür bir hata olduğunu belirttiği gözlemlenmiştir.

Seçilen projeler bütünlüğü sağlamak için nesne yönelimli ("Java" programlama dilinde) yazılmış olup orta ölçekli projelerden oluşmaktadır.

Çalışmada, projelerden elde hata sayıları ise Tablo 3' te verilmiştir. Projelerdeki hata sayıları belirtilen "Spotbugs" analiz aracıyla elde edilmiştir.

Tablo 3. Kullanılan Projelerdeki Hata Sayıları
(Bug Numbers in Used Projects)

P.A	H.S	P.A	H.S	P.A	H.S	P.A	H.S
P1	61	*P6*	38	*P11*	28	*P16*	24
P2	41	*P7*	51	*P12*	49	*P17*	16
P3	58	*P8*	40	*P13*	12	*P18*	37
P4	56	*P9*	96	*P14*	18	*P19*	46
P5	59	*P10*	42	*P15*	18	*P20*	20

P.A : Proje Adını,
H.S : Hata Sayısını temsil etmektedir.

4. SONUÇLAR *(RESULTS)*

Projelerin analizi daha önceden de belirtildiği gibi "Understand" aracıyla yapılmıştır. Her proje için ölçüt değerlerinin ortalaması, ölçüt değerleri toplamının sınıf sayısına bölünmesi ile hesaplanmış ve Tablo 4' te verilmiştir.

Tablo 4. Ölçütler ve Ortalama Değerleri
(Metrics and Average Values)

PROJE	**WMC**	**DIT**	**RFC**	**NOC**	**CBO**	**LCOM**	**FANIN**	**VG**	**NPRM**	**NPM**
P1	7,3	1,9	4,3	0,5	7,7	31,9	1,3	0,8	3,5	0,2
P2	7,6	2,4	3,5	0,7	8,8	15,2	1,0	0,6	1,7	0,3
P3	9,3	2,3	6,9	0,5	2,6	35,8	1,3	0,2	5,5	0,7
P4	4,2	1,5	7,9	0,4	6,5	33,9	1,2	1,4	6,9	0,8
P5	10,8	1,7	7,4	0,6	5,0	35,2	1,2	0,3	7,2	0,1
P6	9,6	1,1	4,7	0,1	7,3	21,6	1,6	0,6	3,8	0,7
P7	10,0	1,8	5,5	0,4	6,3	34,8	1,3	0,5	4,8	0,5
P8	8,3	1,5	4,3	0,2	8,0	27,5	1,4	0,7	3,5	0,4
P9	25,7	1,6	10,3	0,4	14,1	51,3	1,4	0,3	8,2	0,8
P10	13,3	1,8	7,4	0,7	4,8	24,7	1,0	0,7	4,6	1,9
P11	9,6	1,8	5,1	0,3	6,1	23,8	1,3	0,5	3,8	1,2
P12	15,2	1,2	7,5	0,1	9,5	33,4	1,4	0,8	6,4	0,6
P13	7,7	1,1	4,4	0,0	6,0	16,5	1,5	0,3	3,2	1,1
P14	4,6	1,2	3,2	0,0	8,0	26,7	1,2	0,6	2,7	0,4
P15	6,7	1,5	4,3	0,1	5,1	26,5	1,2	0,6	3,8	0,2
P16	12,5	1,1	3,3	0,1	8,1	27,4	1,1	1,1	2,6	0,6
P17	12,5	1,3	5,4	0,2	7,6	17,0	1,3	0,2	3,8	1,5
P18	13,7	1,4	6,9	0,1	9,7	31,6	1,5	0,5	5,6	0,8
P19	19,8	1,0	4,6	0,0	5,2	47,4	1,0	0,9	1,5	0,0
P20	9,4	1,8	5,4	0,4	3,9	27,9	1,3	0,1	4,9	0,3

"Korelasyon" en genel kullanımda; veriler arasındaki karşılıklı bağımlılığın bir ölçüsüdür. İki değişken arasındaki ilişki derecesine ise korelasyon katsayısı denilir. Korelasyon katsayısı -1 ile 1 aralığında değer alır. Korelasyon katsayısının +1,00'a yaklaşması iki değişken arasında aynı yönde ilişkinin olduğunu gösterir, -1,00'a yaklaşması ise iki değişken arasında ters yönde ilişki olduğunu gösterir [24].

Projelerin hata sayıları ve ölçütlerin ortalama değerleri arasındaki ilişki "Minitab" istatistiksel yazılımı ile hesaplanmıştır. Ölçütler ve toplam hata arasındaki ilişki "Pearson Korelasyonu" kullanılarak hesaplanmıştır. "Pearson Korelasyonu" iki değişken arasındaki ilişkiyi, önemini ve yönünü inceleyen ilişki yöntemidir [24].

Yüksek ilişki gösteren ölçütler ile projelerdeki hata sayıları arasında bir model oluşturmak için ise

"Stepwise Doğrusal Regresyon" analizi uygulanmıştır [25]. "Stepwise Doğrusal Regresyon"; en yaygın kullanılan değişken seçim tekniğidir. Her adımda değişkenleri ekleyerek veya çıkararak yinelemeli bir regresyon model dizisi oluşturur. Herhangi bir adımda bir değişkenin eklenmesi veya çıkarılması kriteri genellikle kısmi F testi ile ifade edilir [24]. Doğrusal regresyon denklemlerinde R^2 değeri yordayıcı değişkenlerin yordanan değişkenin varyansını ne oranda açıkladığına dair araştırmacılara bilgi verir [26]. R^2 değerinin yüksek olması yordayıcı değişkenlerden oluşan doğrusal denklemnin yordanan değişkeni iyi oranda kestirebildiğini göstermektedir.

Uygulanan "Stepwise Doğrusal Regresyon" sonucu Tablo 5' de verilmiştir. R^2 değerinin yüksek olduğu üçüncü adım seçildiğinde çıkan sonuca göre hata sayısı için denklem eşitliği Eş. 1'de verilmiştir:

$$\boldsymbol{Hata\ Sayısı} = (LCOM * 1{,}54) + (NOC * 43{,}3) + (CBO * 2{,}5) - 36{,}8 \quad (1)$$

Tablo 5. Stepwise Doğrusal Regresyon Sonucu
(Stepwise Linear Regression Result)

Adım	1	2	3
Sabit	-8,396	-22,188	-36,852
LCOM	1,650	1,680	1,540
T-Değeri	4,800	6,790	7,720
P-Değeri	0,000	0,000	0,000
NOC		39,900	43,300
T- Değeri		4,270	5,810
P- Değeri		0,001	0,000
CBO			2,510
T- Değeri			3,370
P- Değeri			0,004
S	13,900	9,920	7,830
R-Sq	56,110	78,800	87,590
R-Sq(adj)	53,670	76,310	85,270

P değeri 0,05'ten küçük ise değişken istatistiksel olarak anlamlı bir değişkendir. T değeri ise T testine dayalı olarak elde edilen bir değerdir.

T değerinin mutlak değerinin 1,98'den büyük olması beklenir [26]. Tablo 5' te en güçlü değişkenden başlanarak, diğer değişkenlerin de modele dahil edilmesi ile birlikte sonuçlar verilmiştir. Sonuçlarda LCOM, NOC ve CBO değişkenlerinin en güçlü değişkenler olduğu görülmektedir. Bu 3 değişkene bakılarak hata sayılarının anlamlı bir şekilde kestirilebildiği söylenebilir.

4.1. Kestirim Doğruluğu *(Prediction Accuracy)*

20 projeye, regresyon sonucunda çıkan model denklemi uygulanarak her proje için formül yardımıyla kestirilen hata sayıları hesaplanmıştır. Çıkarılan formül ile LCOM, CBO ve NOC değerleri yerlerine konularak kestirilen hataların sayısı hesaplanmıştır. Gerçek hata sayıları ve çıkarılan formül ile hesaplanan hata sayıları Tablo 6' da verilmiştir:

Tablo 6. Gerçek Hata ve Kestirilen Hata Sayıları
(Actual Bug and Estimated Bug Numbers)

P.A	H.S	K.H.S	P.A	H.S	K.H.S
P1	61	57,56	*P11*	28	30,44
P2	41	43,07	*P12*	49	44,29
P3	58	47,62	*P13*	12	7,70
P4	56	50,42	*P14*	18	25,44
P5	59	56,61	*P15*	18	21,29
P6	38	19,76	*P16*	24	30,28
P7	51	51,95	*P17*	16	18,99
P8	40	36,29	*P18*	37	43,80
P9	96	96,37	*P19*	46	50,32
P10	42	47,17	*P20*	20	34,83

P.A : Proje Adını,
H.S : Hata Sayısını,
K.H.S : Kestirilen Hata Sayısını temsil etmektedir.

Bağıl Hata (Magnitude Relative Error-MRE), kestirim modellerinin değerlendirilmesinde en yaygın kullanılan ölçüdür. Pred(l) kestirim seviyesi ise literatürde sıklıkla kullanılır ve belirli bir doğruluk düzeyi için yapılan gözlemlerin bir oranıdır [27].

MRE, Pred(l) eşitlikleri Eş. 2 ve Eş. 3'de verilmiştir.

$$\mathrm{MRE} = |\mathrm{Egercek} - \mathrm{Etahmin}| \ / \ \mathrm{Egercek} \quad (2)$$

$$\mathrm{Pred(l)} = \mathrm{k/N} \quad (3)$$

Eş. 3 'de "k", MRE değeri "l" değerinden daha düşük olan proje sayılarını belirtmektedir. Eşitlikteki "N" sayısı, veri setindeki toplam proje sayısını ifade eder.

Pred değerleri hesaplanırken "l" değeri 0,25 ve 0,30 olarak alınmıştır çünkü bu değerler kestirim modelleri yaratırken en sık kullanılan sayılardır [27-28].

Conte ve diğerleri [27]; ortalama MRE'nin 0,25'ten düşük ve Pred(0,25) için ise 0,75'e eşit veya büyük olmasının kabul edilebilir olduğunu önermiştir. Genel olarak bir kestirim modelinin doğruluğunun Pred(0,25) ile orantılı olduğundan bahsetmişlerdir.

Tate ve diğerleri [28] ise; Pred(0,30) değerinin kullanılması gerektiğini savunmuştur. Pred(0,30) değerinin ise 0,70'i geçmesi gerektiğini belirtmişlerdir. MRE değerleri; tahmin edilen hata sayısından gerçek hata sayısının çıkarılarak, tekrar gerçek hata sayısına bölümünden elde edilmiştir. Hesaplanan MRE değerleri Tablo 7' de verilmiştir.

Tablo 7. Hesaplanan MRE Değerleri
(Calculated MRE Values)

Proje Adı	MRE	Proje Adı	MRE
P1	0,06	*P11*	0,09
P2	0,05	*P12*	0,10
P3	0,18	*P13*	0,36
P4	0,10	*P14*	0,41
P5	0,04	*P15*	0,18
P6	0,48	*P16*	0,26
P7	0,02	*P17*	0,19
P8	0,09	*P18*	0,18
P9	0,00	*P19*	0,09
P10	0,12	*P20*	0,74
Kestirim Doğruluğu pred(0,25) = 0,75 pred(0,30) = 0,80			

Tahmin başarısının iyi olduğu; Conte ve diğerlerinin [27] belirttiği gibi; Pred(0,25) için 0,75'e eşit veya büyük olması, Tate ve diğerlerinin de belirttiği gibi Pred(0,30) için ise 0,70'den büyük olması tezinden yola çıkılarak gösterilmiştir.

5. TARTIŞMA VE ÖNERİLER *(DISCUSSION AND RECOMMENDATIONS)*

Bu çalışmada yazılım projelerinin kalitesi ile hata sayıları arasındaki ilişkinin çıkarılabilmesi ve hata kestirimi için bir model önerisi sunulması amacıyla 20 adet açık kaynak kodlu, nesne yönelimli oyun projeleri incelenmiştir. "Understand" aracı ile elde edilen yazılım ölçütleri arasında LCOM, NOC ve CBO ölçütlerinin "SpotBugs" kod analiz aracıyla ortaya çıkarılmış toplam hata sayılarını belirleme katsayısının (R^2) %85,2'sini açıkladığı görülmüştür. Bu üç yordayıcıdan elde edilen kestirim denkleminin pred(0,25)=0,75 ve pred(0,30)=0,80 oranında doğru kestirim yaptığı bulunmuştur. Elde edilen pred değerleri literatürde geçen değerleri desteklemektedir. Yapılan çalışmanın, literatürdeki pred(0,25) ve pred(0,30) eşik değerlerini karşıladığı ve kestirim doğruluğunun kabul edilebilir düzeyde olduğu görülmüştür. İlgili çalışmalarda kullanılan hazır veri setleri incelendiğinde; hata kestirimi ile ilgili olabilecek ölçütlerin bazılarının bu veri setlerinin içinde olmamasından dolayı açık kaynak kodlu projeler üzerinden gidilerek proje analizleri gerçekleştirilmiştir.

İlgili çalışmalar incelendiğinde Gyimothy ve diğerleri [17]; CBO ve LCOM ölçütlerinin hata kestiriminde iyi olduklarını söylerken NOC ölçütünün hata kestiriminde güvenilmez olduğunu belirtmişlerdir. Yapılan bu çalışmada ise LCOM, NOC ve CBO ölçütlerinin hata kestiriminde iyi sonuçlar verdiği görülmüştür. Gyimothy ve diğerleri NOC ölçütünün hata ile olan ilişkisinin güvenilir olmamasını belirtmesine rağmen bu çalışma NOC ölçütünün de hata kestiriminde önemi olduğunu göstermiştir. Yapılan çalışmada Gyimothy ve diğerlerinin çalışmasından farklı bulgular gözlemlenmesinin sebeplerinin seçilen programlama dillerinin farklı olması ve kullandıkları hata takip sisteminin farklı olmasından kaynaklı olduğu düşünülmektedir.

Bu araştırma kapsamında elde edilen kestirim denklemi 20 proje ile sınırlıdır. Bu nedenle gelecekteki farklı projelerde ya da çalışmalarda daha fazla proje eklenerek modelin kestirim doğruluğunun arttırılması hedeflenebilir.

KAYNAKLAR *(REFERENCES)*

[1] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou and C. Zhai, "Have Things Changed Now?", in *Proc. of the 1st Workshop on Architectural and System Support for Improving Software Dependability*, ASID 2006, San Jose, California, USA, October 21, 2006, ACM Newyork, NY, USA, 2006 pp. 25-33.

[2] B. Boehm, *Software Engineering Economics*, NJ: Prentice-Hall, 1981.

[3] Y. Singh, *Software Testing*, UK: Cambridge University Press, 2012.

[4] M. D'Ambros, M. Lanza and R. Robbes, " Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison", *Empirical Software Engineering*, vol. 17, pp. 531-577, August 2011. Doi: 10.1007/s10664-011-9173-9.

[5] R. Moser, W. Pedrycz and G. Succi, "A Comparative Analysis of The Efficiency of Change Metrics and Static Code Attributes for Defect Prediction", in *Proc. of International Conference on*

Software Engineering, 10-18 May 2008, Leipzig, Germany, Available: IEEE Xplore, http://www.ieee.org. [Accessed: 18 June 2019].

[6] S. Kim, T. Zimmermann, E. J. Whitehead, Jr. and A. Zeller, "Predicting Faults from Cached History", in *Proc. of the 2007 29th International Conference on Software Engineering (ICSE07), 20-26 May 2007, Minneapolis, USA*. Available: IEEE Xplore, http://www.ieee.org [Accessed: 18 June 2019].

[7] V. Basili, L. Briand and W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Transactions on Software Engineering*, vol. 22, no. 10 pp. 751-761, Oct. 1996. Doi: 10.1109/32.544352.

[8] A. E. Hassan, "Predicting Faults Using the Complexity of Code Changes", *in Proc. of 2009 31st International Conference on Software Engineering, 16-24 May 2009, Vancouver, BC, Canada*. Available: IEEE Xplore, http://www.ieee.org. [Accessed: 22 July 2019].

[9] A. S. Nuez-Varela, H. G. Perez-Gonzalez, F.E. Martinez-Perez, and C. Soubervielle-Montalvo, "Source Code Metrics: A Systematic Mapping Study", *Journal of Systems and Software*, vol. 128, pp. 164-197, April 2017. Doi: 10.1016/j.jss.2017.03.044.

[10] N. E. Fenton and M. Neil, "Software Metrics: Successes, Failures and New Directions", *Journal of Systems and Software*, vol. 47, pp. 149-157, July 1999. Doi: 10.1016/S0164-1212(99)00035-7.

[11] T. Mccabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, December 1976. Doi: 10.1109/TSE.1976.233837.

[12] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, vol. 23, no. 2, pp. 11-112, November 1993. Doi: 10.1016/0164-1212(93)90077-B.

[13] S. Chidamber and C. Kemerer, "A Metrics Suite for Object-Oriented Design", *IEEE Transactions on Software Engineering*, vol. 20, no.6, pp. 476-493, Jun 1994. Doi: 10.1109/32.295895.

[14] M. H. Halstead, "Natural laws controlling algorithm structure?", *ACM SIGNPLAN Notices*, vol. 7, no. 2, pp. 19-26, February 1972. Doi: 10.1145/953363.953366.

[15] M. D'Ambros, M. Lanza and R. Robbes, "On the Relationship Between Change Coupling and Software Defects", in *Proc. of the 2009 16th Working Conference on Reverse Engineering, October 13-16, 2009, Lille, France.* Available: IEEE Xplore, http://www.ieee.org. [Accessed: 20 June 2019].

[16] T. Zimmermann, R. Premraj and A. Zeller, "Predicting Defects for Eclipse", *in Proc. of the 2007 Third International Workshop on Predictor Models in Software Engineering*, PROMISE' 07, 20-*26 May 2007, Washington, DC, USA*. Available: IEEE Xplore, http://www.ieee.org. [Accessed: 22 July 2019].

[17] T. Gyimothy, R. Ferenc and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction" *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897-910, Nov. 2005. Doi: 10.1109/TSE.2005.112.

[18] Y. Perez-Riverol, L. Gatto, R.Wang, T. Sachsenberg, J. Uszkoreit, F.V. Leprevost,C. Fufezan, T. Tement, S.J.Eglen, D.S.Katz, T.J.Pollard, A.Konovalov, R.M. Flight, K. Blin and J. A. Vizcaino, "Ten Simple Rules for Taking Advantage of Git and Github", *PLOS Computational Biology*, vol. 12, no.7, July 2016. Doi: 10.1371/journal.pcbi.1004947.

[19] Understand scitools Features page, *scitools.com*, Copyright 1996, Available: http://www.scitools.com/features/. [Accessed: January. 15, 2019].

[20] M. Thapaliyal and G. Verma, "Software Defects and Object-Oriented Metrics - An Empirical Analysis", *Journal of Computer Applications*, vol. 9, no. 5, pp. 41-44, November 2010. Doi: 10.5120/1379-1859.

[21] K. M. Breesam, "Metrics for Object-Oriented Design Focusing on Class Inheritance Metrics" in *Proc. of the 2007 2nd International Conference on Dependability of Computer Systems*, DepCoS-RELCOMEX '07, *July 2007, Szklarska, Poland.* Available: IEEE Xplore. [Accessed: 15 June 2019].

[22] U. Erdemir, U. Tekin and F. Buzluca, " Object Oriented Software Metrics and Software Quality", *Software Quality and Software Development Tools Symposium*, 9-10 October, Istanbul. Available: http://softwaresuccess.org. [Accessed: 10 September 2019].

[23] "Spotbugs", *spotbugs.github*, Available: http://spotbugs.github.io. [Accessed: April. 11, 2019].

[24] D. C. Montgomery and G. C. Runger, Applied *Statistics and Probability for Engineers*, New York, NT: John Wiley & Sons, 2010.

[25] A. Field, *Discovering Statistics Using SPSS* (5th ed.), United States: Sage Publications, 2009.

[26] B. G. Tabachnick and L. S. Fidell, *Using Multivariate Statistics* (3rd ed.), Boston: Allyn & Bacon, 1996.

[27] S. D. Conte, H. E. Dunsmore and V.Y. Shen, *Software Engineering Metrics and Models*, Redwood City, CA, USA: The Benjamin/Cummings, 1986.

[28] G. Tate, J. Verner, *Software Costing in Practice, The Economics of Information Systems and Software*, Oxford, Butterworth-Heinemann, 1991.