Fırat Üniversitesi Müh. Bil. Dergisi
34(2), 635-645, 2022

Araştırma Makalesi
https://doi.org/10.35234/fumbd.1114056

Yazılım Ürün Ölçütlerinin Uygulamalı İncelenmesi

Ayşe KAHVECİ YETİŞ[1*], Resul DAŞ[2]
[1,2] Yazılım Mühendisliği, Teknoloji Fakültesi, Fırat Üniversitesi, Elazığ, Türkiye
[*1]aysekahveci23@hotmail.com, [2]rdas@firat.edu.tr

(Geliş/Received: 10/05/2022; Kabul/Accepted: 26/07/2022)

Öz: Yazılım sistemleri, yaşantımıza her geçen gün artarak dahil olmaktadır. Buna bağlı olarak yazılım geliştiricileri tarafından bir ihtiyaç için alternatif pek çok yazılım sistemi geliştirilmektedir. Kullanıcı istekleri doğrultusunda şekillenen sistemlerde çalışma performansı ve güncellenebilir yazılım seçenekler arasında eleyici özelliklerden bazılarıdır. Kod kalitesini belirlemek için metrik ya da ölçüt olarak tanımlanan değerlerden faydalanılır. Bu çalışmada, ürün ölçütlerinde yer alan kaynak kod ölçütleri ve sınıf tabanlı ölçütler açıklanmıştır. Kaynak kod ölçütlerinden olan McCabe ile sınıf tabanlı ölçütlerden olan Chidamber ve Kemerer, Brito e Abreu MOOD, Bansiya ve Davis QMOOD ölçüt kümeleri irdelenmiş ve sınıf tabanlı ölçütler tablo halinde sunulmuştur. Ölçütlerin uygulama üzerinde gösterilmesi için Java tabanlı bir kütüphane uygulaması geliştirilmiştir. Geliştirilen masaüstü uygulamasının ölçütleri, Eclipse CodeMR eklentisi yardımıyla analiz edilmiştir. Yapılan analizler sonucunda riskli ölçütler tespit edilerek sonuçlar tablolar halinde sunulmuştur.

Anahtar kelimeler: Yazılım ölçütleri, McCabe, CK, MOOD, QMOOD.

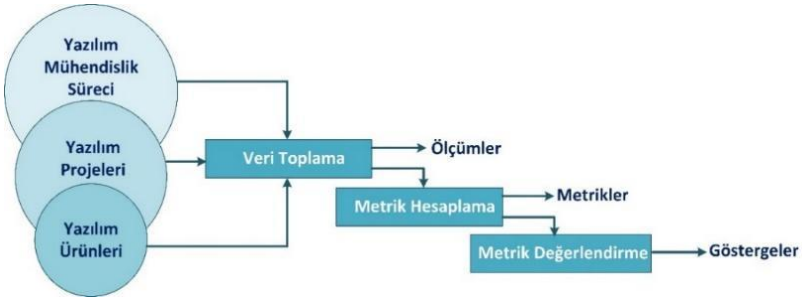
Applied Analysis of Software Product Metrics

Abstract: Software systems are increasingly included in our lives day by day. Accordingly, many alternative software systems are developed by software developers for a need. Working performance and updatable software in systems shaped according to user requests are some of the eliminating features among the options. Values defined as metrics or criteria are used to determine code quality. In this study, source code criteria and class-based criteria in product criteria are explained. McCabe in the source code criteria and Chidamber and Kemerer, Brito e Abreu MOOD, Bansiya and Davis QMOOD metric clusters in the class-based criteria were examined and class-based metrics were presented in tables. A Java-based library application has been developed to display metrics on the application. The criteria of the developed desktop application were analyzed with the help of the Eclipse CodeMR plugin. As a result of the analysis, risky criteria were determined and presented with tables.

Key words: Software metrics, McCabe, CK, MOOD, QMOOD.

1. Giriş

Yazılım mühendisliği, bir yazılım veya sistemin yazılım yaşam döngüsü boyunca farklı süreçlerden geçtiğini ifade eder [1]. Bu süreçlerin her birinde farklı parametreler ve çeşitli ölçümler kullanılarak doğrulama işlemleri yapılır. Bu nedenle yazılım mühendisliğinde ölçüm, önemli bir rol oynar ve metrik olarak da ifade edilir [2]. Bir yazılım ürününün niteliklerini ölçmek için titiz bir yaklaşım gereklidir. Her ölçüm açıkça tanımlanmış ve kolayca anlaşılabilir çeşitli parametrelere ve özelliklere bağlı, belirli bir amaç veya ihtiyaçları ifade etmelidir. Yazılım mühendislik süreçleri, yazılım projeleri ve yazılım ürünleri Şekil 1'de gösterildiği üzere iç içe girmiş önemli olgulardır. Yazılım kalite ölçütlerinin bu üç olgudan toplanan veriler sonucunda elde edildiği görülmektedir [3].



Şekil 1. Yazılım ölçütleri toplama süreci [3]

* Sorumlu yazar: aysekahveci23@hotmail.com. Yazarların ORCID Numarası: [1] 0000-0002-0832-892X, [2] 0000-0002-6113-4649

Bu çalışmanın amacı, geliştirilen bir uygulamaya ait kod ölçütlerinden yaygın olarak bilinenlerin analizini yapmak ve bu analizler doğrultusunda çıkarımlarda bulunmaktır. İkinci bölümde yazılım kalitesinin belirlenmesi için kullanılan ve literatürde en çok yer edinen ölçütlerin sınıflandırılması verilmiştir. Bu kapsamda Kaynak Kod McCabe Ölçütleri ve Sınıf Tabanlı Tasarım Ölçütlerinde yer alan Chidamber ve Kemerer, Brito e Abreu MOOD, Bansiya ve Davis QMOOD ölçütleri açıklanmıştır. Üçüncü bölümde ise geliştirilen örnek uygulamaya ait CodeMR'ın desteklediği ölçütler yer almaktadır. Dördüncü bölümde uygulama sonuçları detaylı bir biçimde açıklanarak görseller ile ifade edilmiştir. Son bölümde ise çalışma hakkında çıkarımlar yapılmış, yorumlarda bulunulmuştur.

2. Yazılımda Ürün Bazlı Ölçütler

Ürün ölçütleri, ürünün kalitesine ilişkin boyut, kod ölçümleri, karmaşıklık, tasarım özellikleri, performans, verimlilik, güvenilirlik, taşınabilirlik ve kalite düzeyi gibi özelliklerini tanımlar. Ürün ölçütleri, gereksinimlerden yerleşik sistemlere kadar, geliştirmelerinin herhangi bir aşamasındaki yazılım ürünü ölçümleridir. Ürün ölçütleri yalnızca yazılım özellikleriyle ilgilidir. Ürün ölçütleri iki sınıfa ayrılır:

• *Dinamik ölçütler*, yürütülmekte olan bir programdan yapılan ölçümlerle toplanır. Dinamik ölçütler bir programın verimliliğini ve güvenilirliğini değerlendirmeye yardımcı olurken, statik ölçütler bir yazılım sisteminin karmaşıklığını anlamaya yardımcı olur [4]. Genellikle yazılım kalite özellikleriyle oldukça yakından ilişkilidir. Belirli görevler için gereken yürütme süresini ölçmek ve sistemi başlatmak için gereken süreyi tahmin etmek nispeten kolaydır. Bunlar doğrudan sistem arızalarının verimliliği ve yazılımın güvenilirliği ile ilgili olup, arızanın türü kaydedilebilir.

• *Statik ölçütler*, tasarım, programlar ve belgeler gibi sistem temsillerinden yapılan ölçümlerle toplanır. Statik ölçütlerin kalite özellikleriyle dolaylı bir ilişkisi vardır. Karmaşıklık, anlaşılabilirlik ve sürdürülebilirlik arasındaki ilişkiyi türetmeye ve doğrulamaya çalışmak için bu ölçütlerin büyük bir kısmı önerilmiştir. Yazılım ürün kalitesinin arttırılması amacıyla ürün ölçütleri üzerine çok sayıda kalite ölçütleri belirlenmiştir. Ancak, bu ölçütler ile ilgili literatürde birbirinden farklı yaklaşımlar mevcuttur [5]. Geliştirilen ürünün amacı, boyutu, kullanmış olduğu teknolojiler, sistematik yapısı gibi birçok önemli kısımların incelenmesi gerekir. Bu incelemelerin hepsinde niceliksel ölçümler de mümkün olamamaktadır. Bu sebeple bu bölümde genel ürün ölçütleri ile birlikte en yaygın kullanılan kod ölçütleri verilmektedir.

2.1. Kod Tabanlı Ölçütler

Yazılımın kalitesinin değerlendirilmesinde sürdürülebilirlik önemli bir etkendir. Sürdürülebilirlik için kodun sürekli gelişime açık olması ve bakımının kolay olması gerekmektedir. Dolayısıyla kodun okunabilirliği ve anlaşılabilirliğinin yüksek olması kalite oranını da arttırır. Yazılım kod kalitesinin belirlenmesi için çeşitli ölçütler önerilmiştir.

McCabe Ölçütleri

Önerilen ölçütlerden en temel olanı geleneksel McCabe kalite ölçütleridir [5]. Geleneksel kalite ölçütleri proje büyüklüğü, kodun okunabilir ve anlaşılabilir olması, algoritmik karmaşıklık gibi içerdiği 3 farklı bileşenle en temel şekilde kodun iç yapısını değerlendirir. Geleneksel ölçütlerde temel konular yardımıyla yazılım sisteminin kalitesi hakkında bilgi sahibi olunur.

Yazılım kalitesi hakkında gerçekleştirilen literatür araştırmalarında en çok raporlanan kalite metriğinin, yazılım güvenilirliğini doğrulamak amacıyla kullanılan 'hata tespit metriği' olduğu görülmüştür. Bu ölçütler, yazılımda var olan toplam hata sayısı hakkında bilgi vermektedir. Bu ve benzeri temel ölçüt türlerinin tespiti için yazılımı oluşturan kodların dikkatlice incelenmesi McCabe kalite ölçütleri yardımıyla gerçekleştirilir [6].

Kod bakımının kolay yapılabilirliğini ve anlaşılabilirliğini arttırmak için yorum satırları kullanılmaktadır. Kod karmaşıklık analizi, program kod satırlarının sayısını dikkate alan ölçüttür. Kod bloklarındaki satırlarda bulunan boşluk, yorum, mantıksal ifadeler gibi satırların sayılmasıyla belirlenen ölçütler ile yapılan ölçme işlemidir. Tablo 1'de McCabe kod ölçütleri verilmiştir.

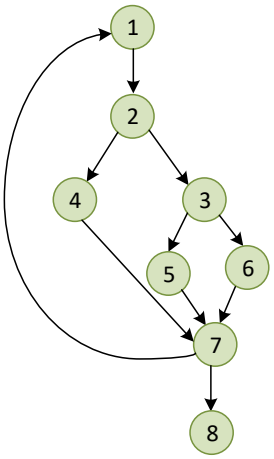
Tablo 1. McCabe'e göre kaynak kodlar için ölçütler [3]

Ölçüt adı	Açıklama
Kod satırları (LOC-Lines Of Code)	Kod bloğundaki satır sayıları.
Kaynak kod satırları (SLOC-Source Lines of Code)	Kod bloklarındaki boşluk satırları hariç ama yorum satırları da dahil olan tüm kod satırlarının sayısıdır.
Koddaki fiziksel Satırlar (PLOC-Physical Lines Of Code)	Kod bloklarındaki açıklama satırları ve boş satırlar dışında kalan tüm fiziksel satırların sayısıdır. Yorum ve boşluk içermeyen satırlar *(Non-Comment Non-Blank-NCNB)* olarak da bilinir.
Koddaki mantıksal satırlar (LLOC-Logical Lines of Code)	Kod bloklarındaki *if, while, for, switch-case, do-while* gibi mantıksal ifadelerin bulunduğu satırların sayısıdır.
Çalıştırılabilir yordam sayısı (EXEC-Executable Statements)	Kod bloklarında bulunan içinde hesaplama adımları bulunduran, fonksiyon, altyordam, altprogram, metot gibi kod parçacıklarının toplam sayısıdır.
Yorum oranı (CP-Comment Percentage)	Program içerisinde bulunan yorum satırlarının, programda bulunan yorum ve boşluk içermeyen toplam satır sayısına oranıdır. Yorum oranının yüksek olması, programın anlaşılabilir ve okunabilirliğini arttırır.
Döngüsel karmaşıklık (CC-Cyclomatic Complexity)	Koddaki koşullu gidiş yollarının doğrusal dallanmaları şeklinde ifade edilir.

Döngüsel karmaşıklık ölçütü, koşullu ifade ve yapıların sayısı ile hesaplanır. Kodda bulunan kontrol blokları ve döngüsel yapılar koşullu dallanmaya sebebiyet veren ölçütler olarak değerlendirilir [7]. Döngüsel karmaşıklığı yüksek olan modüllerin, döngüsel karmaşıklığı daha düşük olan modüllere göre hataya eğilimli olma olasılığı daha yüksektir. Bu nedenle, bir sisteme entegre edilmeden önce bu tür modüllerdeki hataları ortaya çıkarmak için ortalamanın üzerinde çaba sarf edilmelidir. Kodun sahip olduğu ve koşullu dallanmaya neden olabilecek kırılma noktalarının çokluğu, döngüsel karmaşıklığın yüksek olmasına sebep olur. Döngüsel karmaşıklık, yazılımın mantıksal karmaşıklığının bir ölçüsüdür. Bu ölçü (CC veya V(G)), bağımsız yolların sayısını tanımlamak için kullanılır ve aşağıda verilen formüllerden biriyle hesaplanır [8].

V(G) = (Kenar Sayısı) - (Düğüm Sayısı)+2
V(G) = (Dallanan Düğüm Sayısı) + 1

Şekil 2'de verilen çizgesel diyagram Döngüsel Kod karmaşıklığı açısından incelendiğinde, 4 farklı gidiş yolu olduğu görülmektedir.



Şekil 2. Çizgesel akış diyagram örneği

G graf olmak üzere döngüsel kod karmaşıklığı V(G) = 4 olarak bulunur. Hesaplama yaparken yukarıdaki formüllerden herhangi biri kullanılır. Örnek diyagramda, 8 düğüm ve 10 yol olduğundan ilk formüle göre (10-8+2) denklemine göre 4 çıkar. 2, 3 ve 7 birden çok düğüme giden dallanan düğümlerdir. İkinci formüle göre ise (3+1) denklemine göre 4 çıkar. Olası yollar aşağıda listelenmiştir. Burada 4. yol için 7'ye kadar olan yollar tek bir yol olarak değerlendirilir. Diğer olasılıklar göz önüne alınmaz.

- 1. yol: $1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 8$
- 2. yol: $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 8$
- 3. yol: $1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 8$
- 4. yol: $1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 1 \rightarrow \cdots \rightarrow 7 \rightarrow 8$

Formülde bulunan düğümlerden her biri mantıksal bir dalı, kenarlardan her biri düğümler arasındaki bağlantıyı temsil eder. Programın döngüsel karmaşıklığının fazla olması test işlemi için gerekecek durum sayısının fazla olmasına sebep olacaktır. Dolayısıyla bir yöntemin ya da sistemin döngüsel karmaşıklığının düşük olması, kolay test edilebilirliğini gösterir. Yazılım Mühendisliği Enstitüsü (Software Engineering Institute - SEI) tarafından kabul gören döngüsel karmaşıklık ve risk değerlendirmesi Tablo 2'de verilmiştir [9].

Tablo 2. Döngüsel karmaşıklık değerlerine karşı gelen risklerin değerlendirilmesi

Döngüsel Karmaşıklık	**Risk Değerlendirmesi**
1-10	Düşük riske sahip karmaşık olmayan modül
11-20	Orta riske sahip az karmaşık modül
21-50	Yüksek derecede riske sahip olan karmaşık modül
50-∞	Çok yüksek derece riskli test edilemez modül

Literatür incelendiğinde, yazılım ölçütleri konusunda farklı öneriler ve yaklaşımların olduğu görülmektedir. Maurice Howard Halstead tarafından 1977'de yazılımın ölçülebilir özelliklerini ve aralarındaki ilişkileri belirlemek amacıyla ölçütler belirlemiştir [10]. Önerilen ölçütler, algoritmaların farklı dillerdeki uygulamasını veya ifadesini yansıtmakta, ancak bunların belirli bir platformda yürütülmesinden bağımsız olması gerektiği gözlemlerini yapmıştır.

2.2. Sınıf Tabanlı Tasarım Ölçütleri

Sınıf, bir nesne tabanlı sistemin temel birimidir. Bu nedenle, nesne tabanlı tasarım kalitesinin değerlendirilmesinde, tek bir sınıf, sınıf hiyerarşisi ve sınıf işbirlikleri için ölçüler çok değerlidir. Bir sınıf, verileri ve verileri işleyen işlevleri kapsamaktadır.

Chidamber ve Kemerer Ölçütleri

Chidamber ve Kemerer (CK) tarafından önerilen nesne tabanlı yazılım ölçütleri literatürde en yaygın kullanılan ölçütlerdir [11]. CK ölçütlerinde bir sistemin bütününden ziyade sistemdeki sınıflar dikkate alınır [12]. Nesne yönelimli tasarım için kullanılan bu ölçüt grubunda, sistemdeki sınıflar geniş bir şekilde değerlendirilir [13]. CK ölçüt paketi olarak adlandırılan Chidamber ve Kemerer tarafından sınıf tabanlı ölçütler için 6 (altı) ölçüt önerilmiştir. Önerilen ölçütler Tablo 3'te sunulmaktadır.

Tablo 3. Chidamber ve Kemerer'a göre sınıf tabanlı ölçütleri [3,8,11,14,15]

Ölçüt adı	**Açıklama**
Sınıf Başına Ağırlıklı Metot Sayısı *(WMC-Weighted Methods per Class)*	Bir sınıftaki metotların karmaşıklık derecesinin veya sayısının toplamıdır. WMC, sınıfın geliştirilmesi ve bakımı için harcanacak sürenin tahmininde etkilidir. WMC ile nesne yönelimli bir yazılımın anlaşılırlık, yeniden kullanılabilirlik ve dayanıklılık gibi ölçütleri değerlendirilir. Bundan dolayı, WMC makul olduğu kadar düşük tutulmalıdır.
Kalıtım ağacının derinliği *(DIT-Depth of Inheritance Tree)*	Bu ölçüt, düğümden ağacının köküne kadar olan maksimum uzunluktur. Derin bir sınıf hiyerarşisi, daha fazla tasarım karmaşıklığına yol açar. Büyük DIT değerleri, birçok yöntemin yeniden kullanılabileceğini belirtir. Bu ölçütten yararlanarak yazılımın verimliliği, yeniden kullanılabilirliği, anlaşılırlığı, test edilebilirliği gibi kavramların oranları belirlenir.
Alt Sınıf Sayısı *(NOC-Number Of Children)*	Alt Sınıf Sayısı, bir sınıftan doğrudan türetilen toplam alt sınıfların sayısıdır. Çocukların sayısı arttıkça yeniden kullanım artar. Ancak alt sınıf sayısı yüksek olan sistemler, daha karmaşık ve daha çok hataya sahip, yüksek zaman-çaba-maliyet ile test edilebilirlik özelliklerine sahiptir.

Tablo 3 Devamı

Nesne Sınıfları Arasındaki Bağımlılık *(CBO-Coupling Between Object Classes)*	Bir sınıfın bağımlı olduğu sınıfların sayıları toplamıdır. Genel olarak, her sınıf için CBO değeri düşük tutulmalıdır. CBO, yeniden kullanılabilirlik ve verimliliğin ölçülmesinde kullanılır. Bir sınıfın az bağımlılığa sahip olması, farklı uygulamalarda daha kolay yeniden kullanılabilirliğe sahip olması demektir. Bağımlılığın fazla olması testlerin daha dikkatli bir şekilde yapılması gerektiğini, dolayısıyla test maliyetlerinin de aynı oranda artacağını gösterir.
Sınıfın Tetiklediği Metot Sayısı *(RFC-Response For a Class)*	RFC, bir nesnenin tetikleyebileceği tüm metotların sayısıdır. RFC arttıkça, test dizisi büyüdüğü için gereken efor da artar. RFC, test edilebilirlik, bakım ve test zaman-maliyet değerleri hakkında fikir verir. Bu nedenle RFC değerinin düşük olması, sistemin test edilebilirlik oranının yüksek, test için harcanan zaman-maliyet değerlerinin düşük, sistem dayanıklılığının yüksek ve bakımın kolay olduğunu gösterir.
Metot İçi Uyumsuzluk *(LCOM-Lack of Cohesion in Methods)*	LCOM metriği ile metotların birbirlerine olan benzerliği ölçülür. LCOM yüksekse, yöntemler nitelikler aracılığıyla birbirine bağlanabilir. Bu, sınıf tasarımının karmaşıklığını artırır. LCOM'un düşük tutulması en iyi durumdur. LCOM değeri ne kadar yüksekse, o kadar çok durum test edilmelidir. LCOM yardımı ile verimlilik ve yeniden kullanılabilirlik derecesi hakkında bilgi edinilir.

Brito e Abreu MOOD Ölçütleri

Brito e Abreu MOOD ölçüt kümesi, nesne yönelimli yöntemin yapısal noktalarını esas alır [12]. MOOD ölçüt kümesinin prensipleri, kapsülleme, kalıtım, çok çeşitlilik, mesaj aktarımı olarak sıralanır. Brito e Abreu MOOD tarafından nesneye dayalı sınıf tabanlı ölçütlerin belirlenmesinde 6(altı) ölçüt kullanılmıştır [16]. Önerilen ölçütler Tablo 4'te sunulmaktadır.

Tablo 4. Brito e Abreu MOOD'a göre sınıf tabanlı ölçütleri [3,17,18]

Ölçüt adı	**Açıklama**
Metot Gizleme Faktörü *(MHF-Method Hiding Factor)*	Metot Gizleme Faktörü(Method Hiding Factor-MHF), sistemde tanımlı olan bütün sınıflardaki çağrılabilen metotların, var olan bütün metotlara oranıdır. Kalıtımla gelen metotlar hesaplamaya dahil edilmez. Bu ölçüt, sınıf tanımının görünürlüğü hakkında yorum yapma şansı tanır.
Nitelik Gizleme Faktörü *(AHF-Attribute Hiding Factor)*	Kalıtımla gelen niteliklerin dahil edilmemesi ve sistemde tanımlı olan tüm sınıflardaki görünür niteliklerin, tüm niteliklere oranı olarak hesaplanır. MHF metriğinde olduğu gibi AHF metriğinin hesaplanmasında da kalıtım ile gelen metotlar hesaplamaya dahil edilmez ve MHF gibi sınıfların görünürlüğü hakkında bilgi sahibi olmak için bu ölçütten yararlanılabilir. Bu ölçütle sınıf tanımının görünürlüğü ölçülür.
Metot Kalıtım Faktörü *(MIF-Method Inheritance Factor)*	Var olan bütün sınıflardaki kalıtım ile gelen metot sayısının, sınıflarda bulunan toplam metot sayısına oranı olarak tanımlanır.
Nitelik Kalıtım Faktörü *(AI-Attribute Inheritance Factor)*	Sistemde tanımlı olan tüm sınıflardaki kalıtım yardımıyla gelen niteliklerin, sistemde var olan bütün niteliklere oranı ile belirlenir.
Çok Biçimlilik Faktörü *(PF-Polymorphism Factor)*	Belirlenen bir sınıf için sistemdeki farklı çok şekilli olan durumların, maksimum olası diğer çok şekilli durumlara oranı olarak belirlenir.
Bağımlılık Faktörü *(CF-Coupling Factor)*	Sistemde sınıflar arasındaki bağımlılık sayısının toplamının, oluşabilecek maksimum bağımlılık sayısına oranı olarak belirlenir. Oranların belirlenmesinde kalıtım ile gelen bağımlılıklar dikkate alınmaz.

Bansiya ve Davis QMOOD Ölçütleri

Bansiya ve Davis QMOOD Ölçüt Kümesi yazılım toplam kalite endeksinin hesaplanması için faydalanılan ölçüt kümesidir. 4 farklı seviye şeklinde tanımlanmıştır. Seviyelerin ilk adımı sınıf, metot gibi nesne yönelimli yaklaşımların belirlenmesi yer alır. Bir diğer seviyede QMOOD ölçütleri yardımıyla karmaşıklık, uyumluluk gibi bir üst seviyedeki özelliklerin hesaplanması gerçekleştirilir [12]. Bu özelliklerin kalite değerlendirilmesi dikkate

alınarak yazılım sistemine ait toplam kalite indeksi (Total Quality Index(TQI)) hesaplanır. QMOOD ile yazılım tasarım kalitesinin ölçülmesinde ölçütlerin değer aralıklarının farklı olmasından dolayı, z-puanı normalizasyonu kullanılır [19].

• *Yazılım Kalite Nitelikleri (Attribute):* QMOOD ölçüt kümesinde işlevsellik, verimlilik, genişleyebilirlik, yeniden kullanılabilirlik ve esneklik gibi niteliklerden bahsedilir.

• Nesneye Dayalı Yazılım Özellikleri (Property): QMOOD ölçüt kümesinin 2. seviyesinde artık ölçümler gündeme gelir. Bu aşamada, kalıtım, kapsülleme, çok biçimlilik, soyutlama, bağımlılık, hiyerarşi, yazılımın büyüklük ve karmaşıklığı hakkında bilgilere ulaşılabilir [14].

• *Nesneye Dayalı Yazılım Bileşenleri (Components):* Nesne yönelimli yazılım bileşenleri, nitelik, metot, sınıf, ilişkiler ve sınıf hiyerarşisini içinde barındırır [20].

• *Nesneye Dayalı Yazılım Ölçütleri:* Nesne yönelimli kalite değerlendirilmesinde hiyerarşik bir yaklaşım tanımlanır. Bu yaklaşımda, nesne yönelimli değerlendirme için kullanılan ölçütler sınıflar, nesneler ve bu iki yapı arasındaki ilişkilerin yapısal ve davranışsal özellikleri ile belirlenir [12].

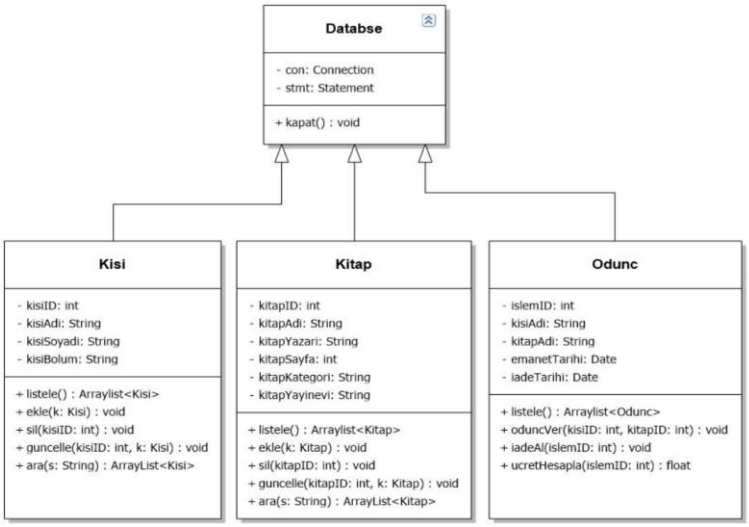
Bansiya ve Davis tarafından irdelenen QMOOD(Quality Model for Object Oriented Design) ölçütlerinde kullanılan değerler Tablo 5'te sunulmaktadır.

Tablo 5. Bansiya ve Davis QMOOD'a göre sınıf tabanlı yazılım ölçütleri [11,15,20,21]

Ölçüt adı	**Açıklama**
Ortalama Ata Sayısı *(ANA-Avarage Number of Ancestors)*	Tüm sınıflarda bulunan ve Chidamber-Kemerer ölçütleriyle hesaplanan DIT değerlerinin ortalaması olarak hesaplanır. Yazılımda soyutlamanın kullanımı hakkında bilgi verir.
Metotlar Arası Uyumluluk *(CAM-Cohesion Among Methods)*	Bansiya tarafından tam olarak tanımı verilmemiş olsa da literatür tanımı olarak metotların imzalarına bakarak ölçülen uyumluluğun hesaplanması olarak ifade edilebilir.
Sınıf Arayüz Boyutu *(CIS-Class Interface Size)*	Sınıfta bulunan erişime açık (public) metotların sayısıdır. Bu sayı sayesinde yazılım sisteminin mesajlaşma özelliği hakkında yorum yapılabilir.
Veri Erişim Metriği *(DAM-Data Access Metric)*	Yazılım sisteminin sınıflarında bulunan özel erişimli (private) ve korumalı (protected) olarak tanımlanan niteliklerin, var olan tüm niteliklere oranı olarak belirlenir. Bu ölçüt ile yazılımın kapsülleme özelliği hakkında bilgi sahibi olunabilir.
Doğrudan Sınıf Bağımlılığı *(DCC-Direct Clas Coupling)*	Bir sınıfın parametre olarak kabul edildiği sınıfların toplam sayısı ile bu sınıf nitelik olarak içeren sınıf sayılarının toplamı olarak değer alır. Sınıflar arası bağımlılık bu ölçüt ile belirlenir.
Kümeleme Ölçüsü *(MOA-Measure Of Aggregation)*	Tanımlanan sınıf bildirimlerinin, tanımlı olan temel veri tiplerine oranı olarak belirlenir.
İşlevsel Soyutlama Ölçüsü *(MFA-Measure of Functional Abstraction)*	Sistemdeki tüm sınıflarda bulunan türetilmiş metot sayıları toplamının, tüm metot sayıları toplamına oranıdır. Yazılım sisteminin kalıtım özelliği bu ölçüt ile değerlendirilir.
Metot Sayısı *(NOM-Number of Methods)*	Sınıfta tanımlı olan metot sayısının toplamıdır. Sınıflarda bulunan metot sayısı, sınıf karmaşıklığının da göstergesidir.

3. Örnek Proje için Kod Ölçütlerinin Hesaplanması

Örnek proje olarak Java ortamında temel kütüphane ihtiyaçlarını karşılayan bir uygulama geliştirilmiştir. Geliştirilen uygulamaya ait sınıf UML diyagramı Şekil 3'te verilmiştir.



Şekil 3. Projenin sınıf UML diyagramı

Java dilinin yanında C ve Python gibi farklı diller kullanılarak da yazılımların geliştirilebileceği Eclipse platformu aracılığıyla kullanılan CodeMR eklentisi sayesinde, geliştirilen yazılım sistemine ait kod ölçütleri hesaplanır. Özellikle son yıllarda bu eklentinin akademik çalışmalarda kullanımı artmaktadır [22-26]. Java tabanlı uygulamalarda CodeMR eklentisi ile ölçütlerin çıkarılmasında, temel olarak Chidamber ve Kemerer ölçüt kümeleri baz alınır. CodeMR ile belirlenen ölçütler ve açıklamaları Tablo 6'da sunulmuştur.

Tablo 6. CodeMR ile analiz edilen ölçütler

Kısaltması	**Açıklama**
CBO (Coupling Between Object Classes)	Nesne sınıfları arasındaki bağımlılığı verir.
RFC (Response For a Class)	Sınıfın tetiklediği metot sayısını verir.
DIT (Depth Of Inheritance Tree)	Kalıtım ağacının derinliğini verir.
NOC (Number Of Children)	Alt sınıf sayısını verir.
WMC (Weighted Methods Per Class)	Sınıf başına ağırlıklı metot sayısını verir.
LOC (Lines Of Code)	Kod satırlarının sayısını verir.
LCOM (Lack of Cohesion in Methods)	Metot içi uyumsuzluklarının sayısıdır.
MCC (McCabe Cyclomatic Complexity)	McCabe döngüsel karmaşıklığı ifade eder.
NBD (Nested Block Depth)	İç içe bulunan metot derinliklerini verir.
#Pa (Number of Parameters)	Toplam parametre sayısını verir.
#MC (Number of Methods Called)	Çağrılan metot sayısını ifade eder.
#AF (Number of Accessed Fields)	Erişilen alan sayısıdır.

Bu çalışmada, yazılımda bulunan ürün bazlı ölçütler ile birlikte kaynak kod ölçütlerine ve sınıf tabanlı tasarım ölçütlerine yer verilmiştir. Sınıf tabanlı tasarım ölçütleri başlığı altında Chidamber ve Kemerer, Brito e Abreu MOOD, Bansiya ve Davis QMOOD sınıf tabanlı ölçütleri irdelenmiştir. Sınıf tabanlı ölçütlerin alt başlıkları tablo şeklinde sunulmuştur. Çalışma kapsamında Java diliyle yazılan temel düzeyde bir kütüphane uygulaması geliştirilmiştir. Uygulamada Kisi, Kitap ve Odunc sınıfları yer almaktadır. Uygulamanın kod analizi için Eclipse destekli CodeMR eklentisinden faydalanılmıştır. CodeMR eklentisine göre bir projeye ait ölçütler sınıf, paket ve metot ölçütlerinden oluşmaktadır. Geliştirilen uygulamanın sınıf ölçütleri incelendiğinde genel olarak yardımcı model sınıflarının karmaşıklıkları düşük-orta seviyede risk oluşturmaktadır. Sınıf ölçütlerinin genel bakışla değerlendirilmesi sonucunda elde edilen bu karmaşıklık değerleri, sürükle bırak metodu ile oluşturulan arayüzlerde daha yüksek görülmektedir. Bu durum ise yazılan kodların sınıf bazında daha karmaşık yapıda olduğunu ve kod anlaşılabilirliğinin daha az olduğunu gösterir.

4. Uygulama Sonuçları

Çalışma kapsamında, Java diliyle temel düzeyde bir kütüphane uygulaması geliştirilmiştir. Geliştirilen uygulamanın, Eclipse CodeMR eklentisi ile kod ölçütleri çıkarılmıştır. Projeye ait genel ölçütler Tablo 7'de verilmiştir.

Tablo 7. Projeye ait elde edilen sonuçlar

Toplam Satır Sayısı	1133
Sınıf Sayısı	11
Paket Sayısı	2
Problemli / Yüksek Problemli Paket Sayısı	0 / 0

CodeMR kütüphanesinde uygulamaya ait ölçütler farklı şekillerde kategorize edilmiştir. Bu gruplandırma sınıf, paket ve metot ölçütleridir. Uygulamaya ait sınıf ölçütlerinde CBO, RFC, DIT, NOC, WMC, LCOM gibi değerler yer almakta ve Şekil 4'te gösterilmektedir.

Element	Quality Attributes	CBO	RFC	DIT	NOC	WMC	LOC	LOC	LCOM
kutuphaneotomasyonu									
model						34	187	187	
DataBase		0	6	1	3	2	11	11	1.0
Kisi		0	13	2	0	10	48	48	0.375
Kitap		0	13	2	0	10	59	59	0.361
Odunc		0	26	2	0	12	69	69	0.84
view						148	946	946	
AnaPencere		3	27	6	0	12	66	66	0.0
KisiEkle		2	61	6	0	19	138	138	0.778
KisiPanel		2	66	5	0	17	173	173	0.796
KitapEkle		2	67	6	0	25	163	163	0.542
KitapPanel		2	63	5	0	17	175	175	0.796
OduncPanel		2	57	5	0	10	100	100	0.833
OduncVer		4	67	6	0	14	131	131	0.688

Şekil 4. Projeye ait sınıf ölçütleri

CodeMR eklentisi yardımıyla çıkarılan Kişi sınıfına ait *metot* ölçütleri Şekil 5'te verildiği gibi çıkmaktadır. Kişi sınıfına ait metotların ölçütleri incelendiğinde, ara metodunun en fazla *LOC* değerine yani en fazla satır sayısına sahip metot olduğu görülmektedir. LOC değerinin yüksek olması kodun kontrol edilebilirliğinin az olması anlamına gelir ve bu durumda hata meydana gelme riski artar. LOC değeri, tek başına karmaşıklık ile ilgili fikir vermez. LOC kodu yazan kişiden kişiye değişebilen sübjektif bir ölçüttür. Döngüsel karmaşıklık (*MCC*), en fazla ara, ekle ve listele metotlarında görülmektedir. En fazla toplam parametreye sahip *#Pa* metriğini barındıran metot kisi metodu, en fazla çağrılan metot sayısını belirten ölçüt olan *#MC* metriği 7 olarak belirtilen ara metodu ve listele metodunda yer almaktadır. *#AF* değeri en fazla olan metotlar ara, ekle ve listele metotlarıdır.

Element	LOC	WMC	MCC	NBD	#Pa	#MC	#AF
model	187	34					
DataBase	11	2					
Kisi	48	10					
ara(String): ArrayList	13		2	2	1	7	6
ekle(Kisi): void	5		2	1	1	1	6
guncelle(int, Kisi): void	4		1	1	2	1	4
Kisi( ): void	2		1	1	0	0	1
Kisi(int, String, String, Stri	5		1	1	4	0	4
listele( ): ArrayList	11		2	2	0	7	6
sil(int): void	3		1	1	1	1	1

Şekil 5. Kişi sınıfına ait metot ölçütleri

Kitap sınıfına ait ölçütler değerlerinden bazıları Şekil 6'da mevcuttur. Bu ölçüt sonuçlandırılmasına göre *LOC* yani en fazla satır satısına sahip metot kitap sınıfında bulunan ara metodudur. *MCC* ile döngüsel karmaşıklığı en

yüksek metotlar ara, ekle ve listele metodu olarak belirtilebilir. *NBD* ile iç içe blok derinliklerinin en fazla olduğu metotlar ise ara ve listele metotlarıdır. Toplam parametre sayısı olarak belirtilen *#Pa* en fazla kitap metodunda mevcuttur. *#AF* değeri en yüksek olan metotlar ara, listele ve ekle metotlarıdır.

Element	LOC	WMC	MCC	NBD	#Pa	#MC	#AF
kutuphaneotomasyonu							
model	187	34					
DataBase	11	2					
Kisi	48	10					
Kitap	59	10					
ara(String): ArrayList	15		2	2	1	9	8
ekle(Kitap): void	7		2	1	1	1	8
guncelle(int, Kitap): void	5		1	1	2	1	6
Kitap( ): void	2		1	1	0	0	1
Kitap(int, String, String, ir	7		1	1	6	0	6
listele( ): ArrayList	13		2	2	0	9	8
sil(int): void	3		1	1	1	1	1

Şekil 6. Kitap sınıfına ait metot ölçütleri

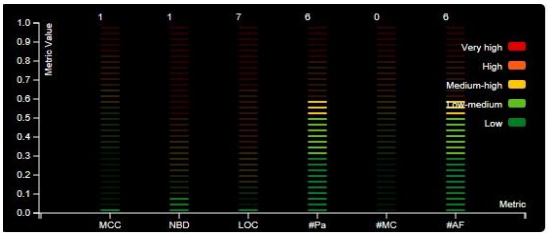
Ödünç sınıfına ait ölçütlerden bazıları Şekil 7'de görüldüğü gibidir. CodeMR eklentisi aracılığıyla farklı ölçütler filtrelenerek tabloda gösterilebilir. Sonuçların değerlendirilmesi adına seçilen ölçütlerde *LOC* ile belirtilen kod satır sayısı en fazla olan metotlar emanet gün hesaplaması ve ödünç işleminin listelendiği metotlar olarak belirtilmiştir. Yazılan kod bloklarında bulunan toplam parametre sayısı *#Pa* en yüksek olan metot 4 olarak belirlenen Odunc metodudur. En fazla çağrılan metot sayısı değeri olan *#MC*'ye sahip kod bloğu, 12 olarak belirlenen emanet günün hesaplandığı kod bloğudur. *#AF* değeri en yüksek olan metot 7 ile listele metodudur.

Element	LOC	WMC	MCC	NBD	#Pa	#MC	#AF
kutuphaneotomasyonu							
model	187	34					
DataBase	11	2					
Kisi	48	10					
Kitap	59	10					
Odunc	69	12					
emanetGunHesapla(Strin	15		2	2	1	12	1
iadeal(String): void	12		2	1	1	6	1
listele( ): ArrayList	15		2	2	0	8	7
Odunc( ): void	1		1	1	0	0	0
Odunc(String, String, Strir	5		1	1	4	0	4
oduncSil(String): void	3		1	1	1	1	1
oduncver(int, int): void	11		2	1	2	4	3
ucretHesapla(String): do	3		1	1	1	1	0

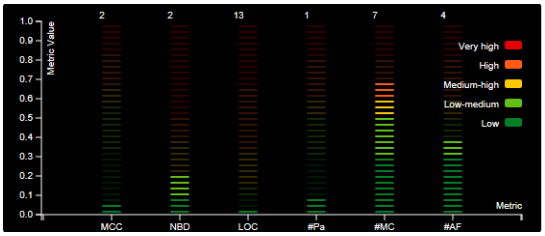
Şekil 7. Ödünç sınıfına ait metot ölçütleri

Ölçüt değerleri yukarda verilen metotların problemli olup olmadıkları ölçüt değerlerinin varsayılan sınırlar içerisinde olup olmaması ile kontrol edilir. Ödünç, kitap ve kişi sınıflarına ait ölçüt değerleri bu açıdan değerlendirilmiştir. Bu sınıflarda bulunan metotlar için seçilen ölçüt değerlerinin risk derecelerine göre kritiklik durumu bazı fonksiyonlar için Şekil 8'de belirtilmiştir. Grafikler göz önüne alındığında *#MC* olarak belirtilen çağrılan metot sayısı metriğinin, ödünç sınıfında bulunan ve Şekil 8.d ile verilen emanet gün hesaplama metodunda yüksek değerde olduğu söylenebilir. *#MC* ifadesinin yüksek olduğu sıralamada ikinci ve üçüncü metot Şekil 8.c ve Şekil 8.b ile verilen ödünç listele ve kişi ara metotlarıdır. Çağrılan metot sayısının yüksek olması yazılan kodun başka kod bloklarına daha çok bağlı olduğu anlamına gelir. Fonksiyonun doğruluğu, tetiklenen tüm fonksiyonların doğru çalışmasına bağlıdır. Kitap sınıfında oluşturulan ve Şekil 8.a ile verilen dolu yapılandırıcıda, *#Pa* ile ifade edilen parametre sayısı ve *#AF* ile ifade edilen erişilen alan sayısı uygulama içerisindeki en yüksek risk oranına sahiptir. Erişilen alan sayısının yüksek olduğu durumları Şekil 8.b ve Şekil 8.c ile verilen kişi ara metodu ve ödünç listele ve kişi ara metotları takip etmektedir. Erişilen alan sayısı, fonksiyonun bir sınıftaki genel parametrelerin yani özelliklerin kaç tanesini kullandığı anlamına gelir. Parametre sayısı metriği ise fonksiyonun aldığı parametre

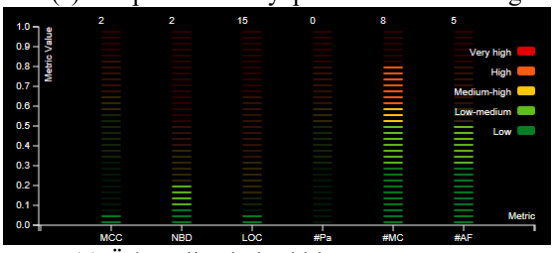
sayısını ifade eder. Kitap sınıfı dolu yapılandırıcısı için, sınıfa ait tüm özellikler parametre olarak istendiğinden bu değer yüksek çıkmıştır.



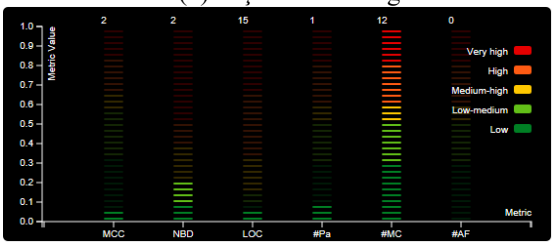
(a) Kitap sınıfı dolu yapılandırıcısı kod bloğu



(b) Kişi ara kod bloğu



(c) Ödünç listele kod bloğu



(d) Emanet gün hesaplama kod bloğu

Şekil 8. Kütüphane otomasyonu modüllerindeki kod ölçütlerine göre risk seviyelerinin grafiksel gösterimi

Şekil 8'de verilen ölçütlerin renklendirilmesi aşamasında CodeMR tarafından kullanılan eşik değerleri Tablo 8'de verilmiştir. Renklendirmede maksimum değer üstündeki değerler kırmızı olarak gösterilmektedir. Maksimum değerin altında kalanlar için ise değerin maksimum değere bölünmesi ile 0-1 aralığında normalize edilmiş değerleri hesaplanır. Normalize değerler için üst sınırlar tabloda gösterilmiştir.

Tablo 8. Şekil 8'de yer alan ölçütlerin risk eşik değerleri

	Max	Low	Low-med	Med-high	High	Very high
MCC	30	0,33	0,57	0,66	0,84	1
Ned	10	0,09	0,21	0,39	0,5	1
Loc	330	0,09	0,15	0,33	0,66	1
Pa	10	0,3	0,51	0,6	0,81	1
Mc	10	0,3	0,51	0,6	0,81	1
Af	10	0,3	0,51	0,6	0,81	1

Uygulamaya ait Odunc sınıfı incelendiğinde, RFC değerinin yüksek çıktığı görülmektedir. Bunun nedeni, Odunc sınıfındaki metotların emanetGunHesapla ve ucretHesapla gibi birbiri ile bağlı çalışan fonksiyonlara sahip olmasıdır. Bağlı fonksiyonların fazla olması, fonksiyonların birinde bulunan hatanın diğer fonksiyonlardaki çalışmayı etkilemesine sebep olduğu için bu tarz fonksiyonlar daha dikkatle incelenmelidir. Alt sınıf sayısını ifade eden NOC değeri, Database modelinde en yüksek değere sahiptir. Bunun nedeni, Kitap, Kisi ve Odunc modellerinin Database sınıfından türetilmesidir. Database sınıfında var olan bir hatanın modelleri etkileyeceği için daha riskli olarak değerlendirilebilir. #MC değeri ile ifade edilen çağrılan metot sayısı parametresi Ödünç Listele ve Emanet Gün Hesaplama kod bloklarında yüksek risk değeri içermektedir. Daha fazla alt fonksiyona bağlı olması, bu kod bloklarında #MC parametresinin yüksek risk içermesine neden olmaktadır.

5. Sonuç

Geliştirilen yazılım sistemlerinde yazılan kodun kalitesini ölçmek için ölçütlerden faydalanılır. Bir yazılım sisteminde belirli özelliklere sahip olmanın derecelendirilmesi, yazılım ölçütünü ifade eder. Literatürde metrik olarak da ifade edilen ölçütler yardımıyla yazılım sisteminin kod analizi gerçekleştirilebilir. Kod analizinin yorumlanması ile kod kalitesini arttırmak hedeflenir. Yazılım sistemlerindeki kodların kalitesi, yazılımın sürdürülebilir olması, tekrar kullanılabilir olması, esnek ve güncellenebilir olması gibi kalite niteliklerini arttırır. Literatürde yazılım ölçütleri hakkında farklı araştırmacılar, birçok yaklaşımlar sunmuşlardır.

Bu çalışmada, yazılım ölçütlerinin sınıflandırılması irdelenmiş, farklı başlıklar altında yer alan ölçütler tablo halinde sunulmuştur. Java programlama dili ile geliştirilen kütüphane uygulamasının kod ölçütleri CodeMR eklentisi yardımıyla analiz edilmiştir. Geliştirilen uygulamanın ölçüt analizleri dikkate alındığında, bir yazılımı oluşturan kodların tamamında birbiriyle bağlantılı yapıların artması riski arttırmaktadır. Yüksek oranda risk içeren yapıların daha özenli geliştirilmesi ve daha az hataya sahip olması gerekmektedir. Bu çalışmada gerçekleştirilen örnek Java Swing projesi için CodeMR tarafından kod ölçütlerin hesaplanması mevcut projeye özgüdür.

Kaynaklar

[1] Lo, Sin Kit, et al. "A systematic literature review on federated machine learning: From a software engineering perspective." ACM Computing Surveys (CSUR) 54.5 (2021): 1-39.

[2] Haindl, Philipp, and Reinhold Plösch. "Value-oriented quality metrics in software development: Practical relevance from a software engineering perspective." IET Software 16.2 (2022): 167-184.

[3] Pressman, R. S.; Maxim, B. R. (2015). Software engineering: A practitioner's approach. Eighth, E., editor, *Software Engineering*, C. 1. Raghu Srinivasan, 8 edition.

[4] Bhatia, P. K. (2022). A Survey of Static and Dynamic Metrics Tools for Object Oriented Environment. *Emerging Research in Computing, Information, Communication and Applications*, ss.. 521-530, Springer, Singapore.

[5] Rashid, J., Toqeer M., and Muhamad W.N. (2019). "A study on software metrics and its impact on software quality." arXiv preprint arXiv:1905.12922.

[6] William F. and Carol T., "Software reuse: metrics and models," ACM Computing Surveys, vol. 28(2), pp. 415-435, 1996.

[7] Lee, M.-C. (2014). Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance. *British Journal of Applied Science & Technology*, C. 4, Sayı 21, 3069–3095.

[8] Gezici, Bahar, Ayça Tarhan, Oumout Chouseinoglou. "Mobil uygulamaların evriminde karmaşıklık, boyut ve iç kalite gelişimi: Keşifsel bir çalışma." Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi 34.3 (2018): 1483-1500.

[9] Murphy, J.; Robinson, J. (2007). Design of a research platform for en route conflict detection and resolution. 7th AIAA ATIO Conf, 2nd CEIAT Int'l Conf on Innov and Integr in Aero Sciences, 17th LTA Systems Tech Conf; followed by 2nd TEOS Forum, 7803.

[10] Halstead, M. H. (1977). Elements of software science (operating and programming systems series). Elsevier Science Inc.

[11] Chidamber, S. R.; Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, C. 20, Sayı 6, 476–493.

[12] Erdemir, U.; Tekin, U.; Buzluca, F. (2008). Nesneye dayalı yazılım metrikleri ve yazılım kalitesi. Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, 9–14, Kültür Üniversitesi, Istanbul, Turkey.

[13] Calp, M. H., Arıcı, N. (2011). Nesne Yönelimli Tasarım Metrikleri ve Kalite Özellikleriyle İlişkisi. Politeknik Dergisi, 14(1), 9-14.

[14] Ozdemir, N.; Dinçer, K.; Gezici, B. (2016). Measurement of the Quality Characteristics of Mobile Applications. *10. Türkiye Ulusal Yazılım Mühendisliği Sempozyumu*, 337–348, Onsekiz MartUniversity, Canakkale, Turkey.

[15] Alakus, T. B.; Das, R.; Turkoglu, I. (2019). An overview of quality metrics used in estimating software faults. *IEEE International Artificial Intelligence and Data Processing Symposium (IDAP)*, 9– 14, Inonu University, Malatya, Türkiye.

[16] Yılmaz, Nebi, and Ayça Tarhan. "Açık kaynak yazılımlarda bakım yapılabilirliği ve güvenilirliği ölçmek için iki boyutlu değerlendirme metodu." Journal of the Faculty of Engineering & Architecture of Gazi University 34.4 (2019).

[17] Brito e Abreu, F.; Carapuça, R. (1993). Candidate metrics for object-oriented software within a taxonomy framework. *Journal of Systems and Software*, C. 26, Sayı 1.

[18] Brito e Abreu, F.; Melo, W. (1996). Evaluating the impact of object-oriented design on software quality. *Evaluating the Impact of Object-Oriented Design on Software Quality*, C. 26, 90–99. IEEE.

[19] Chawla, M. K.; Chhabra, I. (2013). Capturing OO Software metrics to attain quality attributes–a case study. International Journal of Scientific & Engineering Research, 4, Sayı 6, 359-363.

[20] Demirbaş, R. M. (2014). Nesne yönelik yazılım projelerinde öncelikli olarak test edilecek sınıfların yazılım ölçütleri yardımıyla belirlenmesine yönelik bir yöntem. YL Tezi, Maltepe Üniversitesi Fen Bilimleri Enstitüsü, Tez No: 373774.

[21] Bansiya, J.; Davis, C. G. (2002). A Hierarchical Model for Object-Oriented Design Quality Assessmentn. *IEEE Transactions on Software Engineering*, C. 28, Sayı 1, 4–17.

[22] Ramamohan, H.; Raminaidu, S. (2020). Expending Aspect Oriented Methodology to Extent of Dynamic Complexity. *International Journal of Scientific Research inEngineering & Technology*, C. 1, Sayıs 1, 48-52.

[23] Lahti, J. R.; Tuovinen, A. P.; Mikkonen, T. (2021). Experiences on Managing Technical Debt with Code Smells and AntiPatterns. IEEE/ACM International Conference on Technical Debt (TechDebt).

[24] Oussalah, M. C.; Brohan, R.; Moustafa, O. (2021). Object Metrics for Green Software. J. Softw., C. 16, Sayı: 6, 285-305.

[25] Gupta, A.; Chauhan, N. K. (2022). A severity-based classification assessment of code smells in Kotlin and Java application. *Arabian Journal for Science and Engineering*, Springer, C. 47, Sayı: 2, 1831-1848.

[26] Bhatia, P. K. (2021). Dynamic Reusability Measurement Using Machine Learning Algorithms in Object-Oriented Environment. *Intelligent Computing and Communication Systems*, Springer, Singapore.