AÇIK KAYNAK YAZILIM PROJELERİ İÇİN İÇSEL ÜRÜN ÖZELLİKLERİNE VE METRİKLERİNE DAYALI BİR TEST EDİLEBİLİRLİK ANALİZİ YÖNTEMİ

A TESTABILITY ANALYSIS METHOD BASED ON INTERNAL ATTRIBUTES AND METRICS FOR OPEN SOURCE SOFTWARE PROJECTS

EBRU HANOĞLU

DR. ÖĞR. ÜYESİ ADNAN ÖZSOY

Tez Danışmanı

Hacettepe Üniversitesi

Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin

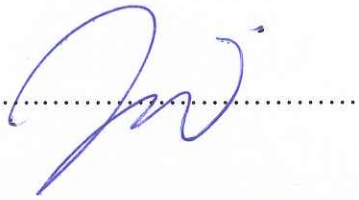
Bilgisayar Mühendisliği Anabilim Dalı için Öngördüğü

YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

2019

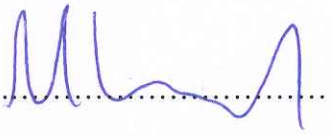
EBRU HANOĞLU'nun hazırladığı "**Açık Kaynak Yazılım Projeleri İçin İçsel Ürün Özelliklerine ve Metriklerine Dayalı Bir Test Edilebilirlik Analizi Yöntemi**" adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Doç. Dr. Oumout CHOUSEINOGLOU
Başkan ..... 

Dr. Öğr. Üyesi Adnan ÖZSOY
Danışman ..... 

Dr. Öğr. Üyesi Tülin ERÇELEBİ AYYILDIZ
Üye ..... 

Dr. Öğr. Üyesi Murat AYDOS
Üye ..... 

Dr. Öğr. Üyesi Mehmet KÖSEOĞLU
Üye ..... 

Bu tez Hacettepe Üniversitesi Fen Bilimleri Enstitüsü tarafından **YÜKSEK LİSANS TEZİ** olarak ..... / ..... /..... tarihinde onaylanmıştır.

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU
Fen Bilimleri Enstitüsü Müdürü

Serdar’a

ETİK

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

17 / 07 / 2019

EBRU HANOĞLU

YAYINLANMA FİKRİ MÜLKİYET HAKKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kâğıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayınlanan "***Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge***" kapsamında tezim aşağıda belirtilen koşullar haricince YÖK Ulusal Tez Merkezi / H. Ü. Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

- ☐ Enstitü / Fakülte yönetim kurulu kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren 2 yıl ertelenmiştir.
- ☐ Enstitü / Fakülte yönetim kurulu gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren .... ay ertelenmiştir.
- ☐ Tezim ile ilgili gizlilik kararı verilmiştir.

17 / 07 / 2019

EBRU HANOĞLU

ÖZET

AÇIK KAYNAK YAZILIM PROJELERİ İÇİN İÇSEL ÜRÜN ÖZELLİKLERİNE VE METRİKLERİNE DAYALI BİR TEST EDİLEBİLİRLİK ANALİZİ YÖNTEMİ

Ebru HANOĞLU

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Danışmanı: Dr. Öğr. Üyesi Adnan ÖZSOY

Haziran 2019, 74 sayfa

Açık kaynak yazılımların önemi ve popülerliği her geçen gün artmaktadır. Bu doğrultuda açık kaynak yazılımlar için kalitenin yüksek tutulması hem geliştiriciler hem de kullanıcılar tarafından beklenmektedir. Yazılım kalitesi ise içsel ve dışsal pek çok faktörden etkilenen çok boyutlu bir kavramdır. Bu kapsamda yazılımın kalitesini etkileyen en önemli faktörlerden birisi de test edilebilirliktir. Bir yazılımın test edilebilirlik düzeyi ne kadar yüksekse, test eforu ve maliyeti o kadar düşük olacak, sonuçta güvenilir ve kaliteli ürünler ortaya çıkacaktır. Ancak, test edilebilirlik yazılım ürünlerinin içsel bir özelliği olmadığı için doğrudan ölçülmesi mümkün değildir. Dolayısı ile test edilebilirliği içsel özellikler ile ilişkilendiren yöntemlere ihtiyaç vardır. Literatürde test edilebilirliği anlamak için pek çok çalışma yapıldığı ve test edilebilirliği tahmin etmek için çeşitli ölçütler önerildiği görülmüştür. Bu tez çalışması kapsamında, test edilebilirliği değerlendirmeye yönelik literatürde önerilmiş olan iç ürün ölçütleri sistematik haritalama metodu kullanılarak derlenmiştir. Daha sonra derlenen bu ölçütlerin kendileri arasındaki ve test süit ölçütleri arasındaki ilişkiler seçilen örnek yazılım kümesi üzerinden deneysel olarak incelenmiştir. Ardından, test edilebilirliğin yazılımlar

açısından neden önemli olduğunun anlaşılması için açık kaynak yazılımların başarısı ile test eforu arasında ilişki olup olmadığı test süit ölçütlerinden faydalanarak incelenmiştir. Bu deneysel incelemeler sonucunda, tez kapsamında kullanılan yazılım seti için kullanımı kolay ve sade bir test edilebilirlik formülü önerilmiş ve bu formülün işlevselliği, test eforu ile olan ilişkisi incelenerek kontrol edilmiştir. Elde edilen korelasyonlar sonucunda test edilebilirliği etkileyen faktörlerin birbirine bağımlı olduğu, test edilebilirliği azaltan faktörlerin harcanan test eforunu artırdığı gözlenmiştir. Açık kaynak yazılımların başarısı ile harcanan test eforu arasında ise orta düzeyde bir ilişki saptanmıştır. Ayrıca, test eforunun yazılan test kodunun kalitesine bağlı olarak artabileceğine dair sonuçlar elde edilmiştir.

Anahtar Kelimeler: açık kaynak yazılım, test edilebilirlik, yazılım kalitesi, yazılım ölçütleri

ABSTRACT

A TESTABILITY ANALYSIS METHOD BASED ON INTERNAL ATTRIBUTES AND METRICS FOR OPEN SOURCE SOFTWARE PROJECTS

Ebru HANOĞLU

Master of Science, Computer Engineering Department

Supervisor: Asst. Prof. Dr. Adnan ÖZSOY

June 2019, 74 pages

Open source software has gained importance and popularity over the last decades. In this sense, high quality open source software products are demanded by developers and users. Software quality is a multi-dimensional concept that is influenced by many internal and external factors. Among them, testability is one of the most important factors affecting the quality of the software. Higher testability of software results in lower testing effort and cost, which leads reliable and qualified products. However, testability is not directly measured since it is not an internal attribute of software products. This makes necessary to correlate testability and internal attributes of the software. There are a lot of studies in the literature about interpreting testability and several metrics have been proposed to predict testability. In the scope of this thesis, internal product metrics intending to predict testability are collected using systematic mapping method. Afterwards, relations between these metrics among themselves and with the test suite metrics are empirically analyzed by utilizing data set which consists of published open source software products. Then, in order to perceive the importance of testability for a software product, the relation between the success of open source software and test effort is examined using test suite metrics.

As a result of these empirical examinations, a simple testability formula is proposed for the software set referenced in this thesis. Moreover, the relation between testability and the test effort is empirically investigated. Gained correlations shows that the factors effecting the testability are dependent to each other. It is observed that the factors that decrease the testability increase the test effort. Furthermore, it is determined that the success of open-source software and the test effort have a mid-level correlation. In addition, there are some results indicating that the test effort might increase based on the quality of the test code.

Keywords: open source software, testability, software quality, software metrics

TEŞEKKÜR

Tez çalışmam boyunca bilgi birikimleri ve tecrübeleri ile bana yol gösteren çok değerli hocalarım Dr. Öğretim Üyesi Ayça Tarhan'a ve Dr. Öğretim Üyesi Adnan Özsoy'a çok teşekkür ederim.

Tez savunma sınavı sırasında önerileri ve yorumlarıyla katkıda bulunan sayın jüri üyelerim Dr. Öğr. Üyesi Tülin Erçelebi Ayyıldız, , Doç. Dr. Oumout Chouseinoglou, Dr. Öğr. Üyesi Murat Aydos ve Dr. Öğr. Üyesi Mehmet Köseoğlu'na saygılarımı ve teşekkürlerimi sunarım.

Seçtiğim mesleği sevmemi sağlayan, yüksek lisansa başlamam için beni motive eden, hayatıma bambaşka bir yön verebilmem için bana güç ve destek olan çok değerli hocam Dr. Kıvanç Dinçer'e sonsuz teşekkür ederim.

Sevgileriyle ve destekleriyle her an yanımda olduklarını hissettiğim Elife anneme, Serpin anneme, Yaşar babama, Metin babama, abilerim Yahya, Dursun, Oğuz ve onların ailemize kattıkları kardeşlerime, yeğenlerim Azra, Furkan, Meryem ve Yiğit'e, mühendis kardeşim Alper'e, artık ailemizden biri olan Satı Abla'ya, bir kız kardeşin yokluğunu hiç hissettirmemiş olan dostum Nilay'a, bana uğur getiren Aras'a ve tüm sabırlarıyla yüksek mühendis olmamı bekleyen bütün aileme çok teşekkür ederim.

Çalışmam boyunca daima yanımda olan Meral, Merve, Ali, Alican ve Utku'ya; Faruk ve Yiğit'e; Ersin ve Burak'a; Eşref ve Arda'ya; Nurefşan ve Ümit'e; İhsan, Sarp ve Eray'a; Nihan, Gökçe ve Ayşe'ye; Ali, Anıl, Cangül, İlke, İnanç, Mustafa, Oğuzhan, Sena, Sinan ve Uygar'a; tez yazım sürecinde tüm sabırlarıyla bana katlanan değerli çalışma arkadaşlarıma teşekkür ederim.

O olmasa asla başaramayacağımı bildiğim, hayatımı güzelleştiren ve ben olmamı, başarmamı sağlayan, her gün bir öncekinden daha çok sevdiğim değerli eşim Serdar'a tüm kalbimle teşekkür ederim.

İÇİNDEKİLER

ÖZET i

ABSTRACT iii

TEŞEKKÜR v

İÇİNDEKİLER vi

ŞEKİLLER DİZİNİ viii

ÇİZELGELER DİZİNİ ix

SİMGELER VE KISALTMALAR x

1. GİRİŞ 1

2. ÖN BİLGİ 3

2.1. Yazılımda Kalite Kavramı 3

2.1.1. McCall Kalite Modeli 3

2.1.2. Boehm Kalite Modeli 4

2.1.3. ISO 9126 Kalite Modeli 5

2.2. Test Edilebilirlik 6

2.3. Yazılım Ölçütleri 6

2.3.1. HS Ölçüt Kümesi 8

2.3.2. Chidamber & Kemerer (CK) Ölçüt Kümesi 8

2.3.3. Diğer Yazılım Ölçütleri 9

2.3.4. Test Süit Ölçütleri 11

2.4. Açık Kaynak Yazılım 11

2.4.1. Açık Kaynak Yazılım ve Tarihsel Gelişimi 11

2.4.2. Açık Kaynak Yazılımın Geleceği 13

2.4.3. Açık Kaynak Yazılımın Sunduğu Avantajlar 14

2.4.4. Açık Kaynak Yazılımların Başarısı 15

3. LİTERATÜR TARAMASI ve İLİŞKİLİ ÇALIŞMALAR 17

4. ARAŞTIRMA TASARIMI 19

4.1. Araştırmanın Amacı ve Araştırma Sorularının Belirlenmesi 19

4.2. Araştırma Planının Hazırlanması 20

4.3. Araştırmanın Uygulanması 20
4.3.1. Ölçütlerin Belirlenmesi 20
4.3.2. Yazılımların Seçilmesi ve Verilerin Toplanması 24
4.3.3. Açık Kaynak Yazılımın Başarı Değerlendirme Yöntemine Karar Verilmesi 26
4.3.4. Toplanan Verilerin Değerlendirilmesi 31
4.3.5. Değerlendirme Sonuçları 34
4.3.6. Değerlendirme Sonuçlarına Dayanarak Bir Yöntem Önerilmesi 37
5. SONUÇLAR VE TARTIŞMA 42
5.1. Genel Sonuçlar 42
5.2. Çalışmaya ve Sonuçlara Yönelik Geçerlilik Tehditleri 43
6. KAYNAKLAR 45
EKLER 51
EK 1 - Proje Seviyesinde Ölçülen Test Edilebilirlik Ölçütleri 51
EK 2 - Proje Seviyesinde Ölçülen Test Süit Ölçütleri 52
EK 3 - Sınıf Seviyesinde Ölçülen Test Edilebilirlik Ölçütleri 53
EK 4 - Sınıf Seviyesinde Ölçülen Test Süit Ölçütleri 54
EK 5 - Metot Seviyesinde Ölçülen Test Edilebilirlik Ölçütleri 55
EK 6 - Metot Seviyesinde Ölçülen Test Süit Ölçütleri 56
EK 7 - Proje Seviyesinde Korelasyon Sonuçları 57
EK 8 - Sınıf Seviyesinde Korelasyon Sonuçları 58
EK 9 - Metot Seviyesinde Korelasyon Sonuçları 59
EK 10 - Yazılım Başarısı ile Test Süit Ölçütleri Arasındaki Korelasyon Sonuçları 60
EK 11 - Tezden Türetilmiş Yayınlar 61
EK 12 - Tez Çalışması Orijinallik Raporu 62
ÖZGEÇMİŞ 63

ŞEKİLLER DİZİNİ

Şekil 2.1. ISO9126 kalite modeli [13] 5
Şekil 2.2. Ölçütlerin sınıflandırılması [19] 7
Şekil 2.3. Açık kaynak yazılımların gelişimi [29] 12
Şekil 2.4. 2009-2019 yıllarında mobil işletim sistemlerinin küresel market paylarındaki değişim [30] 13
Şekil 2.5. DeLone ve McLean IS başarı modeli [33] 15
Şekil 4.1. Dijital kütüphaneler ve makale sayıları 22
Şekil 4.2. Yazılımların başarı değerlendirmeleri 30
Şekil 4.3. İlişkili ölçüt sayısı için istatistiksel değerler 38
Şekil 4.4. Test edilebilirlik indeksleri ve test süit ölçütleri arasındaki korelasyon 40

ÇİZELGELER DİZİNİ

Çizelge 2.1. Açık kaynak kodlu yazılımlarda başarı ölçüm süreci [34] 16
Çizelge 4.1. Makaleler ve önerilen ölçütler 23
Çizelge 4.2. Seçilen ölçütler ve ölçüm düzeyleri 25
Çizelge 4.3. IS literatürü tarafından önerilen başarı kriterleri [34] 27
Çizelge 4.4. Geliştirici sayısının zamana bağlı değişimi üzerinden tanımlanan sınıflandırma sistemi [34] 29
Çizelge 4.5. Açık kaynak yazılımların başarı sıralaması 31
Çizelge 4.6. Metot seviyesinde elde edilen korelasyon sonuçları 32
Çizelge 4.7. Sınıf seviyesinde elde edilen korelasyon sonuçları 32
Çizelge 4.8. Proje seviyesinde elde edilen korelasyon sonuçları 33
Çizelge 4.9. Yazılım başarısı ile test süit ölçütleri arasındaki korelasyon sonuçları 33
Çizelge 4.10. Literatürde test edilebilirliği tahmin etmek için önerilmiş ölçütler 34
Çizelge 4.11. Tez kapsamında incelenen test edilebilirlik ölçütleri 34
Çizelge 4.12. Ölçütlerin birbiriyle ilişkisi 38
Çizelge 4.13. Yazılımlar ve test edilebilirlik indeks değerleri 40

SİMGELER VE KISALTMALAR

Simgeler

p	İstatistik testleri için anlamlılık değeri
rs	Spearman korelasyon katsayı değeri

Kısaltmalar

ACC	Ortalama Çevrimsel Karmaşıklık
CAM	Metotlar Arası Uyumluluk
CBO	Nesne Sınıfları Arasındaki Bağımlılık
CC	Çevrimsel Karmaşıklık
CPM	Bağımlılık Ölçütü
DAM	Veri Erişim Ölçütü
DCC	Doğrudan Sınıf Bağımlılığı
DIT	Kalıtım Ağacı Derinliği
EC	Dışa doğru Bağımlılık
ECS	Koşullu yapıların çıkış sayısı
EF	Yürütülme Frekansı
FOUT	Çıkış Yelpazesi
I	Kararsızlık
IC	İçe Doğru Bağımlılık
IS	Bilgi Sistemleri
LCOM	Metotların Uyumluluğu
LOC	Kod Satır Sayısı
MFA	İşlevsel Soyutlama Ölçüsü
MTMOOD	Metric Based Testability Model for Object Oriented Design

NC	Sınıf Sayısı
NMO	İptal Edilen Metot Sayısı
NNL	İç İçe Döngü Sayısı
NOA	Öznitelik Sayısı
NOC	Alt Sınıf Sayısı
NOM	Metot Sayısı
NOO	Operasyon Sayısı
NOP	Paket Sayısı
NTClass	Test Sınıflarının Sayısı
NUJ	Koşulsuz atlama sayısı
REM	Yeniden Kullanım
RFC	Bir Sınıf İçin Çağrı Sayısı
TI	Test edilebilirlik İndeksi
TLOC	Test Kodunun Satır Sayısı
TM	Test Metotlarının Sayısı
TRFC	Test Sınıflarının RFC Değeri
TWMC	Test Sınıflarının WMC Değeri
WMC	Sınıfın Ağırlıklı Metot Sayısı

1. GİRİŞ

Oxford Üniversitesi ve McKinsey & Company iş birliği ile 2010 yılında 5400 bilişim projesini içeren bir çalışma yürütülmüş ve bu çalışmaya göre, incelenen projelerin %45'inin planlanan bütçeyi, %7'sinin ise belirlenen geliştirme süresini aştığı görülmüştür [1]. Ayrıca bu projelerin %56'sının gereksinimlerin çok azını karşılayarak tamamlanabildiği de ortaya çıkmıştır [1]. Yazılım projelerinde uğranan bu başarısızlıklar yalnızca zaman kayıplarına değil aynı zamanda büyük mali kayıplara da neden olmaktadır. Bu kayıpların önüne geçmek için atılacak ilk adım yazılımın ve yazılım geliştirme sürecinin kalitesinin artırılmasına yönelik çalışmalar yapmaktır.

Yazılım kalitesi kullanıcı ve geliştirici açısından farklı tanımlanabilen çok boyutlu bir kavramdır. Juran kaliteyi "kullanıma uygunluk" olarak tanımlanırken; Crosby kaliteyi sistemin gereksinimleri karşılama ölçüsü ile nitelendirir [2]. Diğer yandan, ISO/IEC 9126 standardı yazılım ürün kalitesini 6 temel karakteristik üzerine inşa eder: İşlevsellik, güvenilirlik, bakım yapılabilirlik, taşınabilirlik, kullanılabilirlik ve verimlilik [3]. Bu standart doğrultusunda test edilebilirlik özelliği, bakım kolaylığı özelliğinin bir alt kategorisi olarak tanımlanır.

Yazılım testi, yazılımdaki hataları bulmak, riskleri tespit etmek ve mevcut uygulamayı tanımlanan en yüksek kalite seviyesine ulaştırmak için yapılan testler bütünüdür [4]. Yazılım test faaliyetleri sayesinde yazılımda yer alan eksiklikler ve hatalar yazılım geliştirme sürecinin erken fazlarında fark edilir ve bunların giderilmesi sonucunda kaliteli, kullanıcıyı ve geliştiriciyi daha çok tatmin eden ürünler ortaya çıkar. Yazılımın test edilebilirliğini artırmanın diğer bir artısı ise daha kaliteli ürünleri daha az maliyetle ortaya çıkmasını sağlamasıdır [5]. Bir yazılım için test edilebilirlik, o yazılım ürününün test faaliyetlerini desteklemesinin ölçüsü olarak tanımlanır ve bu test edilebilirlik düzeyi ne kadar yüksekse, test eforu ve maliyeti o kadar düşük olacaktır [6].

Öte yandan, açık kaynak yazılımlara artan taleple birlikte bu yazılımların da test edilebilirliğinin artırılması, yazılımın kalitesini artırmak ve sürdürülebilirliğini sağlamak açısından önemli hale gelmiştir. Ancak, test edilebilirlik kapsamında birçok ölçüt analizi yapılmış, modeller geliştirilmiş ve testler yapılmış olmasına rağmen açık kaynak yazılımlar için test edilebilirliği değerlendirmeye yönelik sunulmuş standart bir çözüm literatürde henüz bulunmamaktadır [4]. Bu açıdan bakıldığında test edilebilirliğin sade ve

yalın yöntemlerle, yaygın kullanılan ölçüt değerlerine bağlı olarak analiz edilmesine ihtiyaç duyulmaktadır.

Bu tez çalışmasının amacı, literatürde yazılım test edilebilirliğini tahmin etmeye ve değerlendirmeye yönelik önerilmiş iç ürün özelliklerine dayalı kaynak kod ölçütlerini araştırmak, bu ölçütlerin kendi aralarındaki ilişkileri ve test eforuna etkilerini incelemek, açık kaynak kodlu yazılımlarda test edilebilirliği analiz etmenin ve test etmenin ürün geliştirmesine getirisini ortaya koymaktır.

2. ÖN BİLGİ

2.1. Yazılımda Kalite Kavramı

Yazılım dünyası için "kalite", çok boyutlu ve soyut olması nedeniyle tanımlaması oldukça zor ve tartışmaya açık bir kavramdır. Literatürde çeşitli kalite tanımları ve yaklaşımlar mevcuttur. IEEE Standart Yazılım Mühendisliği Terminolojisi Sözlüğü 'ne göre kalite "bir sistem, bileşen ya da sürecin belirlenen gereksinimleri karşılama derecesi" ve "bir sistem, bileşen ya da sürecin müşterinin ya da kullanıcının ihtiyaçlarını ya da beklentilerini karşılama derecesi" olarak tanımlanmıştır [7]. Kalite kavramının öncülerinden Juran'a göre kalite "kullanıma uygunluk (İng. fitness for use)" olarak tanımlanır, bu tanıma göre kalite, kullanıcıların bir ürün ya da servisin ihtiyaç ve beklentilerini karşılayacağından emin olma derecesidir. Crosby ise kaliteyi "isterlere uygunluk" olarak tanımlar [8].

Yazılımın kalitesi kullanıcılar ve geliştiriciler açısından farklı değerlendirilebilir. Kullanıcılar için kaliteli bir yazılımda kullanım kolaylığı, hata barındırmama, gereksinimleri eksiksiz karşılama, en yüksek performansa sahip olma gibi özellikler öne çıkarken geliştiriciler için ürünün düşük bakım maliyetine sahip olması, yeniden kullanılabilir olması ve kolay yönetilebilir olması gibi özellikler daha ön plandadır [6].

Kalitenin ölçülmesi, üzerinde uzun yıllardır çalışılan bir konu olmasına rağmen kalitenin çok boyutlu ve soyut bir kavram olmasının beraberinde getirdiği zorluklar vardır. Yazılımların ürün ve kod kalitesini değerlendirmek için araştırmacılar ve uygulayıcılar tarafından çeşitli modeller geliştirilmiştir. Bu modellerin temel amacı "kaliteyi daha iyi belirlemek/ölçmek"tir ve bu ölçümler için ürünlerin kalite özelliklerinden, karakteristiklerinden ve çeşitli ölçüt kümelerinden faydalanılmıştır. Takip eden kısımda bazı kalite modellerine ait bilgiler sunulmuştur.

2.1.1. McCall Kalite Modeli

Bu model en eski ve öncü modellerden biridir ve McCall tarafından 1977 yılında sistem geliştiricilerine ve sistem geliştirme süreçlerine yönelik olarak sunulmuştur [9]. Sunulan modelin temel amacı dış faktörler ve ürün kalite kriterleri arasındaki ilişkileri değerlendirmektir. Bu doğrultuda kaliteye hem geliştirici hem kullanıcı açısından odaklanır ve geliştiriciler ile kullanıcılar arasındaki boşluğu kapatmayı hedefler [10].

McCall'un kalite modelinin yapısı ürün revizyonu, ürün geçişi, ürün operasyonu olmak üzere üç ana kategoriden oluşur. Her bir kategori içinde yer alan kalite özellikleri:

- Ürün revizyonu kategorisi için, sürdürülebilirlik, esneklik ve test edilebilirlik;
- Ürün geçiş kategorisi için, taşınabilirlik, yeniden kullanılabilirlik, birlikte çalışabilirlik;
- Ürün operasyonu kategorisi için, doğruluk, güvenilirlik, kullanılabilirlik, bütünlük ve verimliliktir.

McCall modelinde kullanılan ölçütlerin birçoğu öznel bir şekilde ölçülür. Kullanıcıdan her bir kalite faktörü için 0 ile 10 arasında bir not vermesi istenir. Değerlendiricilerin yazılımdaki rolüne ve kişisel özelliklerine göre farklı değerlendirmeler ortaya çıkabilmektedir.

2.1.2. Boehm Kalite Modeli

Günümüzdeki kalite modellerinin ikincisi olan bu model Boehm tarafından 1978 yılında önerilmiştir [11]. İçerisinde 3 yüksek seviye karakteristik, 7 orta seviye karakteristik ve kalite ölçütlerine baz oluşturan çeşitli alt seviye karakteristikler barındırır. Bahsi geçen üç üst seviye karakteristik, yazılım kalitesini araştırırken akla gelmesi muhtemel şu temel soruları yanıtlamayı hedefler [12]:

- Olduğu gibi kullanım (İng. as-is utility): Üründe herhangi bir değişikliğe gidilmeden ne kadar kullanabilirim?
- Bakım yapılabilirlik: Ürünü anlamak, değişiklik yapmak ve test etmek ne kadar kolay?
- Taşınabilirlik: Yazılım ortamı değiştirildiğinde ürün hala kullanılabiliyor mu?

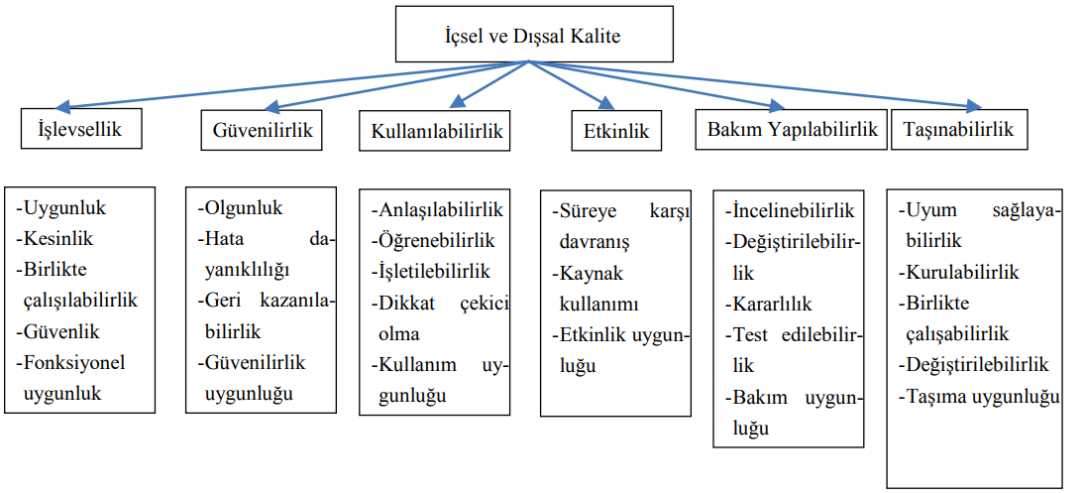
Model çerçevesinde ortaya konulan 7 orta seviye karakteristik ise şu şekildedir [12]:

- Taşınabilirlik
- Güvenilirlik
- Verimlilik
- Kullanılabilirlik
- Test edilebilirlik
- Anlaşılabilirlik
- Esneklik

Model kapsamında bahsedilen alt seviye karakteristikler ise 7 orta seviye özelliğin alt dalıdır ve ölçülebilen yapılara ayrılmıştır.

2.1.3. ISO 9126 Kalite Modeli

Bu modelde yazılım ürün kalitesini etkileyen özellikler hiyerarşik bir ağaç yapısı altında tanımlanmıştır. Yazılım kalitesini içsel kalite, dışsal kalite ve kullanımdaki kalite olarak nitelendirir. Bu standart kapsamında bir yazılımın kalitesi iç özellikleri, dış özellikleri ya da kullanımdaki kalite özellikleri üzerinden yapılan bir ölçüm ile tespit edilebilir [12]. Şekil 2.1'de gösterildiği üzere, iç ve dış kalite özellikleri altında ise şu kalite birimleri toplanmıştır: İşlevsellik, güvenilirlik, kullanılabilirlik, verimlilik, bakım ve taşınabilirlik [12]. Yine her bir birim altında alt birimler yer almaktadır ve bu alt birimler üzerinden yapılacak uygun ölçme ve değerlendirme yöntemleri ile kalite tespiti yapmak mümkündür. Bu standarda göre kalitenin özelliklerinden biri olan bakım yapılabilirlik "yazılımın değişiklik veya düzeltme isteklerine adaptasyon yeteneği" olarak tanımlanmaktadır [9].



Şekil 2.1. ISO9126 kalite modeli [13]

Bu tez kapsamında üzerinde durulan "test edilebilirlik" özelliği, bu modele göre "bakım yapılabilirlik" özelliğinin altında yer almaktadır.

2.2. Test Edilebilirlik

Yazılımda test, hataları bulmak, riskleri anlamak ve gerçekleştirilen ürünü en yüksek kalite seviyesine ulaştırmak için yapılan faaliyetler bütünüdür. Yazılım testi sayesinde hatalar kullanıcıya gitmeden erken evrelerde fark edilir ve daha düşük maliyetlerle telafisi mümkün olur. Bu hataların giderilmesi ile de kullanıcıyı ve geliştirici tatmin eden yüksek kaliteli ürünler ortaya konur. "Daha az maliyetle daha kaliteli ürünler ortaya koymak" için yazılımın test edilebilirliğini artırmak kaçınılmaz bir gerekliliktir. Bu gerekliliğin altında yatan sebep ise yazılımların test edilmesinde hem tasarım hem de uygulama aşamasında büyük zaman ve çaba harcanmasıdır. Öyle ki, Myers ve arkadaşları yazılım geliştirme sürecindeki maliyetin %50'si kadarını test süreçlerinin oluşturduğunu tespit etmişlerdir [14]. Bu açıdan bakıldığında yazılımların geliştirilmesinde test faaliyetlerine yönelik yapılacak iyileştirmelerin önemi anlaşılmaktadır

ISO/IEC 25010–2011 standart tanımına göre test edilebilirlik, bir ürünün işlevini yerine getirdiğini doğrulamak için harcanan çabanın etkisinin ve verimliliğinin derecesidir [15]. IEEE standardında ise test edilebilirlik "bir sistem, bileşen ya da sürecin test kriterlerinin doğrulanmasını ve performans ölçümünün değerlendirmesini destekleme derecesi" olarak tanımlanır [16]. Binder ise test edilebilirliğin test faaliyetlerini kolaylaştırdığını söylemiş ve bu kolaylığı elde etmek doğrultusunda "test edilebilirlik için tasarım" görüşünü savunmuştur. Binder'e göre yazılım geliştirme sürecinin erken safhalarında test edilebilirliği sağlamak önemlidir [17]. Böylelikle yazılım geliştirme sürecinde test edilebilirlik sürekli gözlemlenebilecek ve nihai yazılım için gereken test maliyetleri düşürülebilecektir.

Test edilebilirlik yazılım ürünlerinin içsel bir özelliği olmadığı için doğrudan ölçülmesi mümkün değildir. Bu nedenle yazılımın test edilebilirliğini ölçmek için ölçütler, modeller ve metotlar önerilmiştir.

2.3. Yazılım Ölçütleri

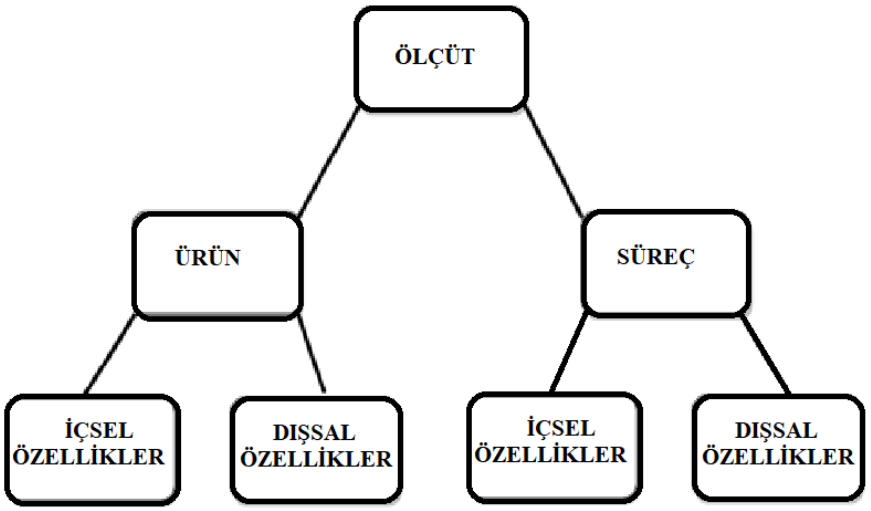
ISO 14598 [18]'e göre ölçüm "bir varlığa bir sayı ve kategori atamaya yönelik süreç" olarak tanımlanır ve bu sayede bir varlığın niteliğini tanımlamış olur. Ölçüt ise "varlığın ölçülecek olan özelliğini ölçme birimi" olarak tanımlanır.

Yazılım ölçütü ise IEEE Standard Glossary [7]'e göre "bir sistemin, bileşenin ya da sürecin belirli bir özellik bakımından sahip olduğu derecenin nicel ölçümü" olarak

tanımlanmıştır. Literatürde yazılımların çeşitli özelliklerini ölçmek için önerilmiş çeşitli ölçütler mevcuttur ve çeşitli açılardan farklı sınıflara ayrılırlar.

Yazılımları ölçmek için önerilen ölçütler ilk olarak ürün ve süreç ölçütleri olarak ikiye ayrılırlar: Süreç ölçütleri geliştirme boyunca o an içinde bulunulan durumu ve kaynakların kullanımını ölçmeyi hedeflerken ürün ölçütleri ürünün sahip olduğu özellikleri anlamaya yöneliktir ve teslim edilebilir ürün ya da dokümanlar üzerinden değerlendirilebilir [19].

Ürün ölçütleri, içsel ve dışsal özellikler olmak üzere iki kategori üzerinden incelenir. Örneğin bir sınıfın büyüklüğü, içerdiği kodun siklomatik karmaşıklığı (İng. cyclomatic complexity), bağımsız yolların sayısı gibi özellikler kaynak kod üzerinden ve kaynak kod çalıştırılmadan değerlendirilebilir. Bu tip özellikler içsel özellikler olarak adlandırılır. Öte yandan kaynak kodun derlenip çalışması ile ölçülebilecek olan yazılımın davranışlarına bağlı olan hata sayısı, kullanıcı dostu olma gibi özellikler dışsal özellikler olarak adlandırılır [19]. Şekil 2.2'de ölçütlerin sınıflandırılmasının bir özeti verilmiştir.



Şekil 2.2. Ölçütlerin sınıflandırılması [19]

Ölçütler, ölçümlerin gerçekleştiği ve verilerin toplandığı zaman açısından da statik ve dinamik ölçütler olarak ikiye ayrılır. Bir diğer gruplama da değerlendirme kriterlerine göre metot, sınıf, paket ve proje ölçütleri olarak 4'e ayırmaktır.

Literatürde pek çok çalışma olsa da test edilebilirliği doğrudan ve tek seferde ölçmeye yarayan ölçütler bulunmamaktadır, çünkü test edilebilirlik yazılımın doğrudan ölçülebilir

olmayan bir dış ürün özelliğidir. Bu nedenle bir yazılımın test edilebilirliği, yazılımın iç ürün özelliklerine dayandırılarak önerilen ölçüm yöntemleri ile analiz edilir. Önerilen ölçüm yöntemlerinin kullandığı ölçütler üzerine yapılan çalışmalar ise ölçütleri analiz ederek gruplamışlardır.

Aşağıda çeşitli ölçüt gruplarına ait örnekler verilmiştir:

2.3.1. HS Ölçüt Kümesi

- Kod Satır Sayısı (İng. Lines of Code – LOC): Bir sınıftaki toplam kod satırı sayısını veren bir büyüklük ölçütüdür. Hesaplanırken yorum satırları ve boş satırlar hariç tutulur [20].
- Sınıf Sayısı (İng. Number of Classes – NC): Yazılımın büyüklüğü hakkında fikir veren yapısal ölçütlerdendir ve bir yazılımın modüllerinde yer alan sınıfların toplam sayısı bulunarak hesaplanır [20].
- Öznitelik Sayısı (İng. Number of Attributes – NOA): Bir sınıf için o sınıfta tanımlanan toplam özniteliklerin sayısıdır [20].
- Metot Sayısı (İng. Number of Methods – NOM): Bir yazılım modülünde yer alan metotların toplam sayısıdır. Bir sınıfın metot sayısı, o sınıfın karmaşıklığı hakkında fikir vericidir [6].
- İptal Edilen Metot Sayısı (İng. Number of Overriden Methods – NMO): Bir altsınıfta yeniden yazılarak değiştirilen metotların sayısıdır [20].

2.3.2. Chidamber & Kemerer (CK) Ölçüt Kümesi

- Sınıfın Ağırlıklı Metot Sayısı (İng. Weighted Methods per Class – WMC): Bir sınıfa ait tüm metotların karmaşıklık toplamıdır [20]. Tüm metotların ağırlığı 1 olarak kabul edildiğinde metot sayısı ile aynı değeri verir [6].
- Kalıtım Ağacı Derinliği (İng. Depth of Inheritance Tree – DIT): Bir sınıfın, kalıtım ağacının köküne uzaklığıdır [20]. Hiçbir sınıftan türemeyen sınıflar için bu değer 0'dır. Kalıtım hiyerarşisinde derinde olan sınıflar, daha fazla metot türettikleri için davranışlarını tahmin etmek zorlaşır ve derin kalıtım ağaçları tasarımın karmaşıklığını artırır [6].
- Alt Sınıf Sayısı (İng. Number of Children – NOC): Bir sınıftan doğrudan türeyen diğer sınıfların sayısıdır [20]. Bir sınıfın çok fazla alt sınıfa sahip olması, kalıtımın

yanlış kullanıldığına işaret ediyor olabilir [6]. Çok fazla sayıda alt sınıfı olan sınıfların metotları daha çok test etmeyi gerektireceği için harcanacak test eforu hakkında fikir vericidir [6].

- Nesne Sınıfları Arasındaki Bağımlılık (İng. Coupling Between Object Classes – CBO): Bir sınıfın bağımlı olduğu diğer sınıfların sayısıdır [20]. Bir sınıf içerisinde yer alan nitelikler ya da metotlar, başka bir sınıf tarafından kullanılıyorsa ve sınıflar arasında kalıtım yoksa, bu iki sınıfın bağımlı olduğu kabul edilir [6].
- Bir Sınıf İçin Çağrı Sayısı (İng. Response for a Class – RFC): Bir sınıfta yer alan bir nesnenin metotları çağrıldığında, bu nesnenin tetikleyebileceği tüm metotların sayısıdır [20]. Sınıfın test maliyeti hakkında fikir verir, bir çağrının çok sayıda metodu tetiklemesi, testin ve hata ayıklamanın zorlaşması demektir [6].
- Metotların Uyumluluğu (İng. Lack of Cohesion in Methods – LCOM): Paylaşılan niteliklere sahip olmayan metot çifti sayısı ile paylaşılan niteliklere sahip metot çifti sayısı arasındaki farktır [20]. Bir sınıfın uyumluluk değerinin düşük olması, sınıfın en az 2 ayrı sınıfa ayrılması gerektiğini göstermektedir [6].

2.3.3. Diğer Yazılım Ölçütleri

- Çevrimsel Karmaşıklık (İng. Cyclomatic Complexity – CC): Yazılım kaynak kodu içerisindeki birbirinden bağımsız, doğrusal dal sayısıdır [20]. Bu ölçüt değerinin büyümesi, çözümlenebilirliği azalttığı için istenmeyen bir durumdur [21].
- İç İçe Döngü Sayısı (İng. Number of Nested Level – NNL): Bir sınıf içerisinde yer alan döngülerin iç içe geçme derinliğidir ve değeri ne kadar büyürse test edilebilirlik azalır [21].
- Çıkış Yelpazesi (İng. Fan Out - FOUT): Bir metodun yerel akışlarının sayısı ile bu metodun güncellediği veri yapılarının sayısının toplamıdır [22].
- Ortalama Çevrimsel Karmaşıklık (İng. Average Cyclomatic Complexity - ACC): Bir sınıftaki tüm metotların karmaşıklığının ortalamasıdır.
- Dışa Doğru Bağımlılık (İng. Export Coupling - EC): Bir sınıftan sistemdeki diğer sınıflara gönderilen metot çağrılarının sayısıdır [23].

- İçe Doğru Bağımlılık (İng. Import Coupling - IC): Bir sınıf tarafından, sistemdeki diğer sınıflardan alınan metot çağrılarının sayısıdır [23].
- Operasyon sayısı (İng. Number of Operations - NOO): Bir sınıftaki operasyonların sayısıdır [24].
- Paket Sayısı (İng. Number of Package - NOP): Bir yazılımdaki paketlerin sayıdır ve yazılımın büyüklüğü hakkında iyi bir fikir vericidir [25].
- Kararsızlık (İng. Instability – I): Dışa doğru olan bağlımlılıkların toplam bağımlılık sayısına oranıdır [25].
- İşlevsel Soyutlama Ölçüsü (İng. Measure of Functional Abstraction - MFA): Sistemde tanımlı tüm sınıflardaki türetilmiş metot sayısının tüm metotların sayısına oranıdır [6].
- Veri Erişim Ölçütü (İng. Data Access Metric - DAM): Sınıfın özel (İng. private) ve korumalı (İng. protected) niteliklerinin tüm niteliklere oranıdır [6].
- Metotlar Arası Uyumluluk (İng. Cohesion Among Methods - CAM): Metotların imzaları arasındaki benzerliğin ölçüsüdür [6].
- Doğrudan Sınıf Bağımlılığı (İng. Direct Class Coupling - DCC): Bir sınıfı parametre olarak kabul eden sınıfların sayısı ile bu sınıfı nitelik değişkeni olarak barındıran sınıfların sayısının toplamıdır [6].
- Yürütülme Frekansı (İng. Execution Frequency – EF): Bir sınıfın içindeki yöntemlerin yürütülme sayısıdır [26].
- Yeniden Kullanım (İng. Reuse Metric – REM): Tasarımdaki sınıf hiyerarşilerinin sayısıdır [27].
- Bağımlılık (İng. Coupling Metric – CPM): Bir sınıf ve bu sınıfla ilişkili farklı sınıfların toplam sayısıdır [27].
- Koşulsuz atlama sayısı (İng. Number of Unconditional Jumps - NUJ): Bir modül içerisinde, bir koşuldan ya da döngüden dışarıya doğru atlamaların sayısıdır [28].
- Koşullu yapıların çıkış sayısı (İng. Number of Exits of Conditional Structs - ECS): Bir modül içerisinde tanımlanan koşulların çıkış dallarının sayısıdır [28].

2.3.4. Test Süit Ölçütleri

Test süit ölçütleri, kaynak koda karşılık yazılmış olan test kodlarından elde edilen ölçütlerdir. Bu ölçütler, birim ve entegrasyon testlerinde harcanacak olan test eforunu analiz etmek ve ölçmek için kullanılır [21]. Yaygın kullanılanları şu şekildedir:

- Test kodunun satır sayısı (TLOC)
- Test sınıflarının sayısı (NTClass)
- Test metotlarının sayısı (TM)
- Test sınıflarının RFC değeri
- Test sınıflarının WMC değeri

2.4. Açık Kaynak Yazılım

2.4.1. Açık Kaynak Yazılım ve Tarihsel Gelişimi

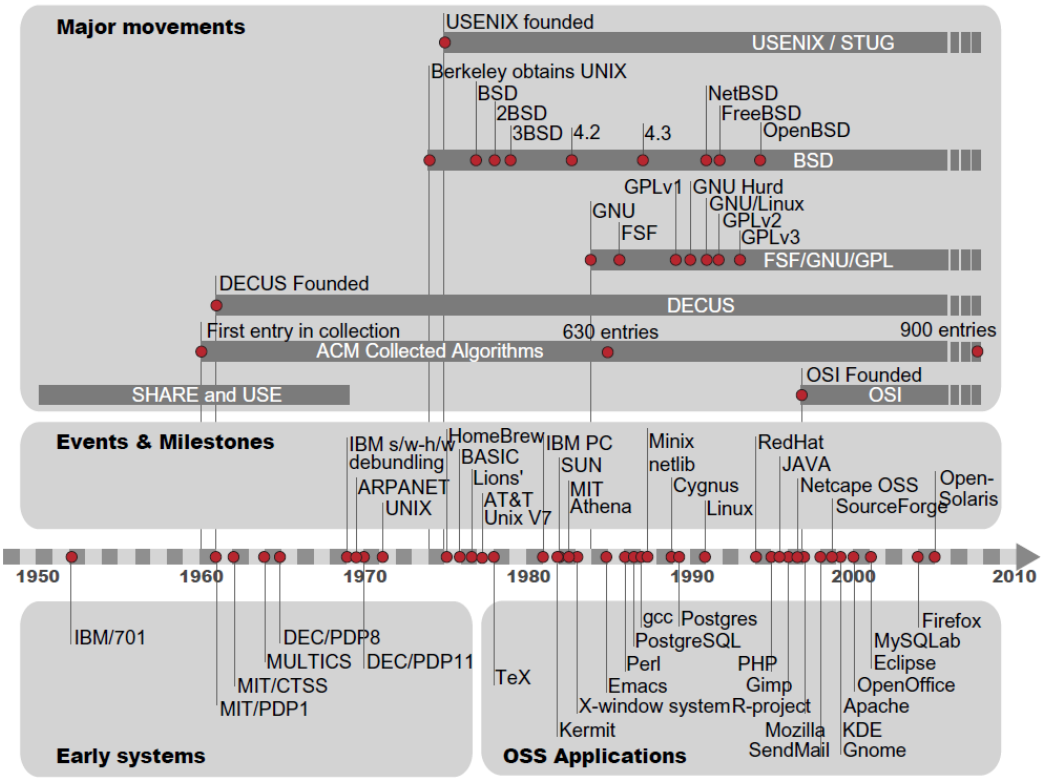
Açık kaynak yazılım, kaynak kodlarını da kapsayacak şekilde ücretsiz olarak dağıtıma sunulan ve kullanıcılarına üzerinde değişim ve paylaşım hakkı veren yazılımdır. "Açık kaynak" kavramı sadece ücretsiz yazılım demek değildir, bunun ötesinde bir anlayış sunar ve açık kaynak yazılımların şu özelliklerini öne çıkarır [23]:

1. Paylaşıma ücretsiz olarak sunulurlar.
2. Kaynak kodları yazılım ile ücretsiz olarak paylaşılır.
3. Paylaşılan yazılım üzerinden türetilecek çalışmalara ve dağıtımına izin verirler.
4. Türetilmiş çalışmalar için kaynak koda atıfta bulunulması talep edilmişse bunun yapılmasını gerektirirler. Böylelikle türetilmiş kodların oluşturabileceği hak ihlallerinden kaynak çalışmanın yazarlarının sorumlu tutulmasının önüne geçerler.
5. Herhangi bir kişiye, gruba veya çalışma alanına paylaşım konusunda ayrımcılık yapmazlar.
6. Sahip olduğu lisanslar paylaşılan tüm partiler için geçerliliğini korur.
7. Lisansları bir ürünle kısıtlı değildir, yazılımın kendisine aittir.
8. Lisansları kendileri ile dağıtıma sürülen diğer yazılımlara karışamaz.

Tarihsel sürece bakıldığında, açık kaynak yazılımların hem yukarda sayılan özelliklerinin belirlendiği 1999 yılına kadar hem de sonrasında birçok önemli gelişme yaşanmıştır. Şekil 2.3 incelendiğinde açık kaynak yazılımların tarihsel gelişimi içerisindeki önemli

hareketler, olaylar ve dönüm noktaları, ortaya konan sistemler ve uygulamalar görülür [29].

1960'larda MIT ve Stanford gibi önde gelen Amerikan üniversitelerinde başlayan yazılım paylaşım kültürü 1980'lerde kurumsal firmaların paylaşımcıları işe almasıyla sekteye uğramıştır. En başlarda paylaşım kültürü çerçevesinde geliştirilen UNIX işletim sisteminin ticari bir yola girmesi ise paylaşım kültürünü savunan yazılımcı Richard Stallman tarafından 1983'te GNU (GNU's Not Unix) projesinin başlatılmasını tetiklemiştir. Ardından 1991 yılında, Linus Torvals Linux projesini başlatmış ve paylaşım mekanizmasını güçlendirerek önemli bir adıma imza atmıştır. 1998 yılında ise Eric Raymond tarafından OSI (Open Source Initiative) kurulmuş ve açık kaynak yazılım savunucularının gücü artmaya başlamıştır. Bu gücün etkileri arasında 1998'de Netscape'in Netspace Navigator 5.0'ın kaynak kodlarını paylaşıma açması, Red Hat'in 4,8 milyar dolarlık bütçe ile piyasada yer alması ve 2000 yılında IBM'in Linux için 1,0 milyar dolarlık yatırım paketi açıklaması gösterilebilir.

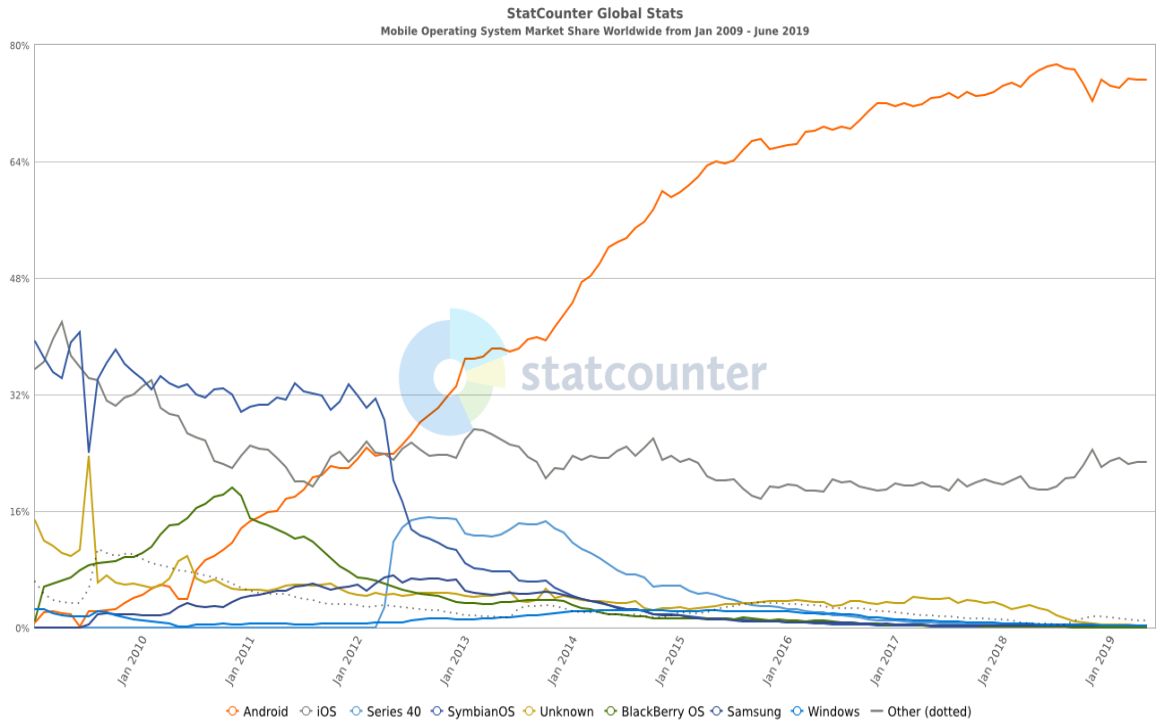


Şekil 2.3. Açık kaynak yazılımların gelişimi [29]

2.4.2. Açık Kaynak Yazılımın Geleceği

İnternetin yaygınlaşması ile paylaşım olanakları genişlemiş, SourceForge ve GitHub gibi yazılım paylaşımına ve yönetimine yardımcı olan platformlar ortaya çıkmıştır. Bu gelişmeler açık kaynak yazılımların yaygınlaşmasını oldukça hızlandırmıştır. Açık kaynak yazılımların geleceği hakkında en çarpıcı projeksiyonu ise Google ve Open Handset Alliance tarafından yürütülen Android projesi vermektedir.

Mobil cihazlar için Linux tabanlı olarak geliştirilen Andorid işletim sistemi açık kaynak bir yazılımdır. Şekil 2.4'teki verilere bakılarak Android'in akıllı telefonlarda tercih edilme payındaki artış göz önünde bulundurulduğunda açık kaynak yazılımların önemi daha da anlaşılmaktadır. Ayrıca, açık kaynak yazılım sadece büyük çaplı firmalar için değil tüm firmalar için gündemde olan bir konudur. 2018 yılında TODO Grup tarafından 748 kişi üzerinden yapılan güncel anket sonuçlarına göre firmaların %37'si hali hazırda açık kaynak yazılım programı yürütmekte iken %16'sı da böyle bir program başlatmayı planlamaktadır [30]. Sonuç olarak, yazılım dünyasının geleceğinde açık kaynak yazılımların büyük söz sahibi olacağı değerlendirilmektedir ve popülerliği gün geçtikçe artmaktadır. Açık kaynak yazılımların artan bu popülerliğinin altında ise sunduğu avantajlar yatmaktadır.



Şekil 2.4. 2009-2019 yıllarında mobil işletim sistemlerinin küresel market paylarındaki değişim [30]

2.4.3. Açık Kaynak Yazılımın Sunduğu Avantajlar

Aralarında Siemens ve Nokia gibi önde gelen şirketlerin de bulunduğu 13 kuruluş ile yapılan çalışmada açık kaynak yazılımların tercih edilme sebepleri şu şekilde listelenmiştir [31]:

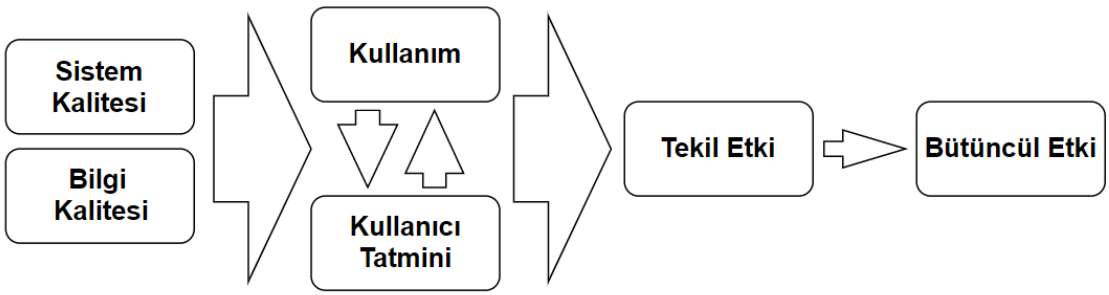
- Güvenilir olması: Uygulamaların kullanılabilirlikleri ve performansları sonucunda elde ettikleri popülerlik, yeni kullanıcılar için güven ortamının oluşmasını sağlar. Bu sayede açık kaynak yazılımın doğası gereği güvenilir olan yazılımlar ön plana çıkar.
- Güvenli olması: Birçok kuruluş açık kaynak yazılımların daha güvenli bir seçenek olduğunu düşünmektedir. Kapalı yazılımlarda kaynak koda erişim sağlanmadığı için zararlı faaliyetler konusunda kullanıcının herhangi bir farkındalığı oluşmaz. Aksine açık kaynak yazılımlar kullanıcının kaynak koda erişmesini sağladığı için hem zararlı faaliyetlerin oluşmadığı garanti altına alınmış olur hem de oluşacak zararlı girişimler en kısa zamanda tespit edilip engellenir.
- Kaliteli olması: Kapalı kaynak yazılımların aksine açık kaynak yazılımlar gelişim sürecinde birçok geliştiricinin ve kullanıcının denetiminden ve değerlendirmesinden geçer. Bu da her yönü ile test edilmiş ve ihtiyaçları karşıladığından emin olunan yazılımların ortaya çıkmasını sağlar.
- Esnek yapısı: Kapalı kaynak yazılımlar üzerine geliştirme yapmak mümkün değilken açık kaynak yazılımlar bu imkânı hiçbir kısıtlama olmaksızın sunar. Bu da kullanıcıların her türlü kişiselleştirmeyi ve geliştirmeyi yapabilmesini sağlar.
- Düşük maliyet: açık kaynak yazılımlı çözümler kapalı kaynak alternatiflerine göre çok daha düşük maliyetli çözümler sunar. Ayrıca, lisanslama, virüs koruması ve servis gibi konularda maliyet oluşturmadığı için de tercih edilirler.
- Bağımsızlık kazandırması: Herhangi bir kuruma veya kurulaşa bağımlı olmaksızın çalışabilme olanağı sağlar. Bu sayede hem geliştirme konusunda bağımsızlık kazanılmış olur hem de zaman anlamında başka firmaların iş takvimine olan bağlılık ortadan kalkar.
- Yaratıcılığı teşvik etmesi: Geliştirme anlamında sunulan bağımsızlık yaratıcılığı da tetiklemektedir.
- Lisans kolaylığı: Hem maliyet anlamında hem de yönetimi anlamında lisanslama zor bir iştir. Açık kaynak yazılımlar ile lisanslama konusunda hiçbir endişe duyulmasına gerek kalmaz.

Lenarduzzi ve arkadaşları tarafından 2010-2016 yılları arasında açık kaynak yazılımların tercih edilme sebeplerindeki değişim üzerine yapılan çalışmada ise ortaya çıkan sonuç kaliteye verilen önemin her sene arttığı yönündedir [32]. Açık kaynak yazılımların kalitesi tercih edilmelerinde her zaman büyük bir öneme sahip olmakla birlikte, özellikle 2016 yılındaki son değerlendirmeye göre kalite parametresi önemini büyük ölçüde artırmıştır ve tercih sebepleri arasında ilk sıralara yerleşmiştir. Bu sonuçlar da göstermektedir ki açık kaynak yazılımlarda kalite her zaman sağlanması gereken bir koşuldur.

2.4.4. Açık Kaynak Yazılımların Başarısı

Açık kaynak yazılımların başarısını değerlendirmek iki açıdan fayda sağlar. Bunlardan birincisi, yazılım geliştiricilerinin kendi yazılımlarının gelişim sürecini takip edebilecek olmasıdır. Bu sayede geliştiriciler başarının düştüğü anlarda gelişim sürecine müdahale ederek süreci daha iyi yönetebileceklerdir. İkincisi ise başarılı açık kaynak yazılımlarının yapısal yönlerinin analiz edilmesi olacaktır. Elde edilen analiz sonuçlarına bakarak başarı için nelere dikkat edilmesi gerektiği ortaya konacaktır ve bu sonuçlar açık kaynak yazılımlara bağlı paydaşlar için büyük önem arz edecektir.

Ancak başarı ölçümünün getireceği bu faydalardan yararlanabilmek için başarıyı ölçecek doğru bir modele ihtiyaç vardır. Literatüre bakıldığında açık kaynak yazılımların başarısının ölçülmesi için pek çok modelin DeLone & McLean tarafından ortaya konan Information Systems (IS) başarı modelinden esinlendiği görülür [33]. Bu modelde bilgi sistemlerinin başarısının birçok açıdan ölçülebildiği fikri ortaya atılmış ve Şekil 2.5'teki başarı modellemesi ortaya çıkmıştır. Şekilde de görüleceği üzere sistem, kullanıcılar ve sonuçlar arasındaki etkileşimler üzerinden başarı tanımlaması yapılmıştır.



Şekil 2.5. DeLone ve McLean IS başarı modeli [33]

Ancak IS modellemesinde odak noktası yazılımın kullanıldığı çevre ve kullanıcının kendisi iken açık kaynak yazılımlarda bu noktalarda gözlem yapmak oldukça zordur. Crowston ve arkadaşları bu problemin üstesinden gelmek için, açık kaynak yazılımların başarısını ölçerken doğası gereği gözlem yapmaya elverişli olan geliştirme sürecini ele almıştır [34].

Crowston ve arkadaşları bu çalışmalarında IS modellemesindeki ölçüm metotlarını açık kaynak yazılımlara uygulanabilirliği açısından incelemişler ve Çizelge 2.1'de görülen çerçeveye indirgemişlerdir. Bu indirgeme sürecinde vardıkları sonuç ise başarı ölçümü konusunda nihai bir ölçütün olmadığı ve her bir ölçütün başarı ölçüsüne farklı bir bakış açısı sunduğu olmuştur. Yani başarı değerlendirilirken farklı açılardan ve bu açıların gerektirdiği ölçütlerle ölçüm yapmak mümkündür.

Çizelge 2.1. Açık kaynak kodlu yazılımlarda başarı ölçüm süreci [34]

Başarı Ölçüsü	**Göstergeler**	**İlgililer**
Proje Çıktısı	Alfadan Betaya kararlı geçiş Tanımlanan Hedefler Geliştirici Memnuniyeti	Geliştiriciler
Süreç	Geliştirici Sayısı Faaliyet Düzeyi Sürümler Arasındaki Süre Hataları Kapatma veya Özellikleri Uygulama Zamanı	Geliştiriciler Kullanıcılar
Proje üyeleri için çıktılar	Bireysel İş Olanakları ve Maaş Bireysel İtibar Bilgi Oluşturma	Geliştiriciler

Açık kaynaklarda başarının değerlendirilmesi literatürde tartışılmaya devam eden konulardan biridir ve pek çok farklı yaklaşım mevcuttur. Bu tez kapsamında başarıyı ölçmek için kullanılan yaklaşım Bölüm 4.3.3'te detaylandırılmıştır.

3. LİTERATÜR TARAMASI VE İLİŞKİLİ ÇALIŞMALAR

Test edilebilirliği oluşturan tüm etmenleri hesaba katarak bir tanımlama, ölçüm ya da değerlendirme yapmak oldukça zordur [35]. Bu nedenledir ki test edilebilirlik kavramının ortaya atıldığı 1975 [36] yılından bu yana test edilebilirlik için farklı tanımlamalar, değerlendirmeler ve ölçüm metotları ortaya konmuştur. Bu süreçte, test edilebilirlik üzerine yapılan çalışmalara olan ilgi, 1990 yılında önemli ölçüde artmış ve test edilebilirlik günümüzün popüler araştırma konuları arasındaki yerini almıştır [37]. Ancak tüm bu ilgiye rağmen test edilebilirliğin uygulanması için gerekli süreçler, rehberler ve araçlar tam olarak ortaya konmuş değildir [38]. Yine de bu amaca hizmet eden birçok test edilebilirlik bakış açısı geliştirilmiştir.

Voas [39] bir yazılım parçasının hatalı olduğu varsayımı altında sonraki yazılım koşturmasında hata çıkarması olasılığı üzerinden bir test edilebilirlik tanımlaması yapmıştır. Hata hassasiyeti üzerinden yürüttüğü çalışmasında ilgili birimlerin kod koşturması sırasında aktif olması ve bunun çıktılara yansıması gibi olasılıkları değerlendirmiştir.

Voas, Miller ile yaptığı bir diğer çalışmasında [40] ise yazılım içerisindeki hata eğilimleri üzerinden ilerlemiş ve test faaliyetleri sırasında oluşabilecek bilgi kayıplarını değerlendirmiştir.

McGregor ve Srinivas [41], yazılım geliştirme sürecinin erken safhalarında eksiksiz ve iyi tanımlanmış dokümanlar gerektiren bir test edilebilirlik hesabı üzerinde çalışmışlardır.

Freedman [42] ise alan test edilebilirliği kavramını tanımlamış ve yazılımın girdileri ve çıktıları arasında ilişki kurmayı gerektiren gözlemlenebilirlik ile kontrol edilebilirlik üzerinden test edilebilirlik ölçümleri sunmuştur.

Test edilebilirliği ölçmeye yönelik diğer bir çalışmada ise Jungmayr [43] yazılımın parçaları arasındaki bağlantıları analiz etmiş ve test-kritik bağlantılar üzerinden hesaplamalara gitmiştir.

Tüm bu çalışmalarda geliştirilen modellerin ortak noktası ise uygulanmaları için kaynak kodun tek başına yeterli olmamasıdır. Diğer bir deyişle, ortaya konan metotlar yazılımların koşturulmasını veya kaynak koda ilave detaylı dokümanlara erişilmesini gerektirmektedir. Aksine, sadece kaynak koda erişim gerektiren ölçütler üzerinden yapılabilecek tanımlamalarda, ölçümlerin daha hızlı alınabilmesi, projelerin erken

safhalarında da elde edilebilmesi ve karmaşıklığa sebebiyet vermemesi gibi avantajlar bulunmaktadır. Bu avantajları değerlendirmek isteyen çalışmalara ise Binder'in [17] test edilebilirliğe dair önerdiği test süit ölçütleri yol göstermiştir. Bu yolda ilerleyen en önemli çalışmalardan birisi de Bruntink ve Deursen tarafından sınıf (İng. class) test edilebilirliği üzerine yapılan deneysel araştırmadır [44]. Bu çalışmada SIG firması tarafından geliştirilmiş DocGen, Jackal, Monitor, ZPC yazılımları ve açık kaynak olan Apache Ant yazılımı sınıf (İng. class) seviyesinde incelenmek üzere, kod satır sayısı ve test sayısı gibi ölçütler üzerinden korelasyon hesapları yapılmıştır. Daha sonrasında ise kullanılan tüm ölçütler arasında elde edilen korelasyonlar ile analizler yapılmıştır. Bu çalışmada eksik olan nokta ise yazılımların sadece kendi içerisinde ve sınıf (İng. class) seviyesinde analiz edilmesidir. Yazarlar da sonuç bölümünde yazılımların tamamına ait ölçütler ve test ölçütleri üzerinden yapılacak bir analizin gerekliliğinden bahsetmişlerdir.

Bruntink ve Deursen'nin çalışmasında [44] yazlım ölçütleri ile test ölçütleri arasındaki ilişkiler analiz edilmiş, test edilebilirliğe dair ölçütler aracılığıyla hesaplanan bir formül verilmemiştir. Böyle bir formül nesne tabanlı yazılımlar için Khan ve Mustafa tarafından MTMOOD (Metric Based Testability Model for Object Oriented Design) yöntemi adı ile geliştirilmiştir [27]. MTMOOD yöntemi de sınıf (İng. class) seviyesinde bir çalışma olup bilgileri paylaşılmayan veriler üzerinden bir formül ortaya koymuştur. Daha sonrasında ise ortaya konan modelin uygulanması ile ortaya çıkan test edilebilirlik sonuçları ve anketlerden elde edilen test edilebilirlik sonuçları kıyaslanmıştır. Bu çalışmada katsayıların bulunması sürecindeki kullanılan veriler ve yöntemler çok da net değildir ve kullanılan ölçüt sayısı oldukça azdır. Ancak, test edilebilirlik formülü sunması açısından alanında öncü bir çalışma olması sebebiyle önemlidir.

Nesne tabanlı yazılımlar için formül sunan diğer bir çalışma ise Huda ve arkadaşları tarafından gerçekleştirilmiştir [45]. Bu çalışmada test edilebilirlik etkilik (İng. effectiveness) ve tekrar kullanılabilirlik (İng. reusability) ile ilişkilendirilmiş ve bu özellikler yerine ölçütlerle ifade edilen formüller koyularak ölçüt tabanlı bir test edilebilirlik formülü elde edilmiştir.

4. ARAŞTIRMA TASARIMI

Araştırma, “Karşılaşılan bir güçlüğün bilimsel yöntem kullanılarak giderilmesi veya verilerin belli bir plana ve sisteme göre toplanması, çözümlenmesi, yorumlanması ve sonucun rapor haline getirilerek problemlere güvenilir çözümler arama çalışması” olarak tanımlanmış bir faaliyettir ve çeşitli yöntemleri bulunmaktadır [46].

Keşifsel çalışma ise araştırmacının fazla bilgi sahibi olmadığı konuları incelediği ya da araştırma konusunun görece yeni olduğu durumlarda yapılan araştırmalardır. Bu çalışmalar, bir konuyla ilgili en genel düzeyde bilgi toplamak için yürütülür. Keşifsel araştırmalar araştırmacının konuyla ilgili mevcut durumu keşfetmesini sağlar ve gelecek çalışmalar için zemin hazırlamasına yardımcı olur [47].

Bu tez çalışması keşifsel bir çalışma olarak tasarlanmış ve araştırma kapsamında birbirini izleyen temel aşamalar şu şekilde tanımlanmıştır:

- Araştırma amacının ve araştırma sorularının belirlenmesi
- Araştırma planının hazırlanması
- Araştırmanın uygulanması
- Uygulama ile elde edilen verilerin değerlendirilmesi
- Değerlendirme sonuçlarının sunulması

4.1. Araştırmanın Amacı ve Araştırma Sorularının Belirlenmesi

Bu tez çalışmasının amacı,

- test edilebilirliği değerlendirmeye yönelik önerilmiş olan iç ürün ölçütlerini literatürden faydalanarak derlemek,
- derlenen bu ölçütlerin kendileri arasındaki ilişkileri incelemek ve birbirlerine olan bağımlılıklarını gözlemlemek,
- derlenen test edilebilirlik ölçütlerinin harcanan test eforuna etkisini gözlemlemek için bu ölçütlerin test süit ölçütleri ile olan ilişkilerini incelemek,
- test edilebilirliğin yazılımlar açısından önemini ortaya koymak için açık kaynak yazılımların başarısı ile harcanan test eforu arasında ilişki olup olmadığını test süit ölçütlerinden faydalanarak incelemek,
- yapılan keşifsel çalışma bulgularına uygun olarak açık kaynak yazılım ürünlerinde test edilebilirliği bu ölçütlere dayandırarak tahmin etmeye yönelik bir yönteme ulaşmaktır.

Bu amaca uygun olarak araştırma soruları şu şekilde belirlenmiştir:

- AS 1: Literatürde test edilebilirliği tahmin etmeye yönelik olarak önerilmiş ölçütler hangileridir?
- AS 2: Araştırma sorusu 1 kapsamında derlenen test edilebilirlik ölçütlerinin birbirleri ile ilişkileri nasıldır?
- AS 3: Araştırma sorusu 1 kapsamında derlenen test edilebilirlik ölçütlerinin harcanan test eforu arasındaki ilişkisi nasıldır?
- AS 4: Literatürde açık kaynakların başarısını ölçmek için geliştirici ve ürün açısından önerilmiş yöntemlerle bakıldığında, yazılımın başarısı ve harcanan test eforu arasında bir ilişki var mıdır?

4.2. Araştırma Planının Hazırlanması

Araştırmanın amacının ve araştırma sorularının belirlenmesinin ardından araştırma tasarımı yapılmıştır. Bu tasarım kapsamında;

- Literatür taranmış ve üzerinde çalışılacak ölçütler belirlenmiştir.
- Üzerinde çalışılacak açık kaynak yazılımların seçilmesi için kriterler belirlenmiş ve ilgili kaynaklardan yazılımların indirilmesi tamamlanmıştır.
- Açık kaynak yazılımın başarısının nasıl değerlendirileceği araştırılmıştır.
- Çalışma kapsamında kullanılacak olan kaynak kod analiz aracı seçilmiş ve ilgili ölçüt değerleri toplanmıştır.
- Elde edilen bulgular istatistiksel yöntemlerle değerlendirilmiştir.
- Keşifsel çalışmanın sonuçlarından faydalanarak bir yöntem önerilmiştir.

4.3. Araştırmanın Uygulanması

4.3.1. Ölçütlerin Belirlenmesi

Araştırma kapsamında ilk olarak literatür taranarak tez çalışması kapsamında incelenecek ölçütlerin seçimi yapılmıştır. Bu seçimi gerçekleştirmek için öncül bir literatür haritalama çalışması yapılmış ve bu çalışma aşağıdaki adımlar takip edilerek gerçekleştirilmiştir:

- Haritalama sorusunun tanımlanması
- Konuyla ilgili akademik yayınlara ulaşmak için dijital kütüphanelerin taranması
- Yayınların havuza eklenmesi için kriterlerin belirlenmesi
- Belirlenen kriterlere göre yayınların seçilmesi

- Yayınların içeriklerinin incelenmesi ve kategorilendirilmesi.

Haritalama sorusunun tanımlanması:

2016 yılında gerçekleştirdiğimiz "Yazılım test edilebilirliği: bir literatür haritalaması" isimli çalışma [4], literatürde test edilebilirliği analiz etmek için az sayıda metot olduğunu ve bu metotların da henüz genel geçer bir formda kabul görmüş metotlar olmadığını ortaya koymuştur. Ayrıca bu çalışma kapsamında test edilebilirliğin yazılım geliştirme sürecinin farklı evrelerinde analiz edilmesine yönelik çalışmalar yapıldığı görülmüştür. Söz konusu açık kaynak yazılımlar olduğunda tasarım dokümanlarına, iyi tanımlanmış bir geliştirme sürecine veya eksiksiz yazılmış test kodlarına erişim ne yazık ki her zaman mümkün olmamaktadır. Tez çalışması kapsamında amaç, iç ürün özelliklerine dayanarak açık kaynak kodların test edilebilirliği hakkında fikir yürütülmesini sağlamaktır. Bu amaçla öncül haritalama çalışması kapsamında tek bir soru belirlenmiş ve makaleler bu açıdan incelenmiştir:

- Test edilebilirliği kaynak kod üzerinden tahmin etmek üzere önerilen ölçütler hangileridir?

Dijital kütüphanelerin taranması:

Kaynakları taramak için bir arama kriteri belirlenmiş ve buna göre 3 dijital kütüphanede arama yapılmıştır. Konu ile ilgili olduğu düşünülen makaleler bir havuza dahil edilmiş ve detaylıca incelenmiştir.

Aramaların gerçekleştirildiği anahtar sözcükler şu şekildedir:

metric AND (evaluate OR assess OR predict OR estimate) AND "software testability"

Arama sözcüklerinin bu şekilde seçilmesinin sebebi, test edilebilirliği yazılım geliştirme sürecinin erken fazlarında gerçekleştiren çalışmaları bulmaktır. Böylece açık kaynak yazılımlar için kullanıcının erişebildiği tek kaynak kod olsa bile, kullanıcının bunun üzerinden test edilebilirlik hakkında fikir yürütebilmesi istenmiştir. Literatürde yazılmış olan test sınıflarından veya tasarım aşamasında hazırlanan tasarım dokümanları üzerinden de test edilebilirlik analizi yapıldığı görülmüş ancak bu çalışmanın dışında bırakılmıştır, çünkü açık kaynak yazılımlar için her zaman yazılmış test kodları ve tasarım dokümanları bulunmamaktadır.

Bu arama sonucunda erişilen makale sayıları Şekil 4.1'de görüldüğü gibidir.

Kaynak	Makale Sayısı
Google Scholar	885
ACM	9
IEEE Xplore	21

Şekil 4.1. Dijital kütüphaneler ve makale sayıları

İlgili makale havuzunun oluşturulması:

Bu kısımda makaleler tek tek incelenerek eleme işlemi gerçekleştirilmiştir. Makaleleri dahil etme ve dışlama kriterleri şu şekilde belirlenmiştir:

- Tez şeklinde ya da dergi, konferans ve workshop kapsamında yayınlanmış olmalıdır.
- İngilizce veya Türkçe olarak yazılmış olmalıdır.
- Konu ile doğrudan alakalı olmalıdır.
- Ön bilgi ya da çalışma kısmında en ez 1 test ölçütünden bahsedilmiş olmalıdır.

Yukarıda bahsedilen dahil etme ve dışlama kriterleri uygulandıktan sonra 143 makaleden oluşan ikinci bir havuz elde edilmiştir. Daha sonra bu havuz şu kriterler de göz önünde bulundurularak ikinci kez gözden geçirilmiştir:

- Birbirinden türetilen çalışmalar havuzdan çıkarılmalıdır.
- Önerilen ölçütler en az bir proje kapsamında sınanmış olmalıdır.
- Test edilebilirliği tahmin etmeye yönelik, kaynak kod üzerinden ölçülebilir en az bir ölçüt önermiş olmalıdır.

Yapılan bu ikinci gözden geçirme sırasında ilişkili çalışmalar, sistematik haritalama çalışmaları ve teorik bilgi konusunda faydalı olabilecek makaleler de seçilerek tezin ön bilgi ve literatür taraması kısmında faydalanmak üzere ayrılmıştır. Bu ikinci gözden geçirme sonucunda, final havuzunda ölçüt öneren ve bu ölçütleri sınayan 15 çalışma kalmıştır.

Çalışmada kullanılacak ölçütlerin belirlenmesi:

Çalışmanın bu aşamasında havuzda kalan 15 makale tek tek incelenmiştir ve çalışmalarda önerilen ölçütler Çizelge 4.1'de gösterilmiştir.

Çizelge 4.1. Makaleler ve önerilen ölçütler

Makale Adı	Önerilmiş Ölçüt	Makale Adı	Önerilmiş Ölçüt
A Study of the Relationship Between Class Testability and Runtime Properties [23]	EC IC	Predicting Class Testability using Object-Oriented Metrics [50]	RFC LOC FOUT
An Empirical Analysis of a Testability Model for Object-Oriented Programs [24]	DIT CBO NOO	Predicting different levels of the unit testing effort of classes using source code metrics a multiple case study on open-source software [51]	CBO LOC WMC
An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes [48]	LCOM	Predicting Testability of Eclipse, A case Study [52]	RFC LCOM NOC
An Empirical Study into Class Testability [44]	DIT LOC FOUT	Testability and Path testing strategies [53]	DIT NMO
Application Of Artificial Neural Networks For Assessing The Testability Of Object Oriented Software [49]	DIT RFC CBO LOC WMC NOC NMO NOA	Testability Assessment of Object Oriented Software Using Internal & External Factor Model and Analytic Hierarchy Process [54]	DIT CBO LCOM WMC NOM NMO
Estimation of a Set of Package Metrics to Determine the Package Testability Efforts on Open Source Software [25]	EC I NOP	The SQO-OSS Quality Model Measurement Based Open Source Software Evaluation [55]	ACC CC NUJ ECS NNL
Metric Based Testability Estimation Model for Object Oriented Design Quality Perspective [45]	CAM DCC MFA DAM	Understanding Class-level Testability through Dynamic Analysis [26]	EC IC EF
Metric Based Testability Model for Object Oriented Design (MTMOOD) [27]	LCOM NOM REM CPM		

Scientific Tools INC. Tarafından geliştirilen Understand [56] Java, C++, C#, Python gibi pek çok farklı programlama dilinde yazılmış kodları ölçmeye ve analiz etmeye yarayan bir statik kod analiz aracıdır ve NASA, US Navy, Apple, Microsoft gibi pek çok kurum tarafından kullanılmaktadır. Bu çizelgede yer alan ölçütlerden Understand aracı ile

ölçülebilen 12 tanesi tez kapsamında kullanmak üzere seçilmiştir. Ölçütlerin test edilebilirlik ile olan ilişkileri yapılan haritalama çalışması kapsamında ölçütlerin önerildiği makalelerden ile bulunmuştur. Bu ölçütlerin listesi şu şekildedir:

- ACC
- CBO
- NOC
- NOM
- RFC
- LOC
- FOUT
- CC
- DIT
- NNL
- LCOM
- WMC

4.3.2. Yazılımların Seçilmesi ve Verilerin Toplanması

Çalışma kapsamında kullanılacak açık kaynak yazılımlar GitHub üzerinden şu arama kurallarına bağlı kalarak seçilmiştir:

- 2013 – 2018 yılları arasında geliştirilmiş olmalıdır. Bu kriterle yazılımların gelişen teknolojiden aynı oranda etkilenmiş olması amaçlanmıştır.
- Test kodları yazılmış olmalıdır.
- Yazılımın büyüklüğü 250.000 satırı geçmemelidir.
- Yazılımlar C# ya da Java dilinde gerçekleştirilmiş olmalıdır.

GitHub üzerinde yapılan arama sonucunda seçilen yazılımlar şunlardır:

- Apache/ant [57]
- Apache/commons-lang [58]
- Jeantessier/dependency-finder [59]
- Google/ExoPlayer [60]
- Findbugsproject/findbugs [61]
- Google/guava [62]
- Jabref/jabref [63]

- Jfree/jfreechart [64]
- Apache/jmeter [65]
- Netty/netty [66]
- Apache/poi [67]
- ReactiveX/RxJava [68]
- Aspnet/AspNetWebStack [69]
- Aspnet/EntityFrameworkCore [70]
- Dotnet-architecture/eShopOnContainers [71]
- Microsoft/msbuild [72]
- Aspnet/Mvc [73]
- Sonarr/Sonarr [74]

Yazılımların seçilmesinin ardından Understand 5.1 [56] aracı kullanılarak ölçütler toplanmıştır. Bu yazılım analiz aracı, proje, sınıf ve metot seviyesinde ölçüm değerleri sunmaktadır. Ölçütler toplandıktan sonra her bir proje için kullanılacak olan ölçüt değeri, Aggarwal ve arkadaşlarının "Empirical Study of Object-Oriented Metrics" isimli çalışmasında [75] bahsedildiği gibi belirlenmiştir. Her bir projenin ayrı sınıfları için bulunan değerlerin ortalaması alınmış ve o proje için sınıf bazında ölçüt değeri belirlenmiştir. Aynı şekilde metot bazında ölçülen ölçütler için de ortalama alınmış ve metot seviyesinde değerlendirilecek olan değerler saptanmıştır.

Yazılımlar için proje, sınıf ve metot seviyesinde test edilebilirlik açısından incelenen ölçütler Çizelge 4.2'deki gibidir.

Çizelge 4.2. Seçilen ölçütler ve ölçüm düzeyleri

Metot	**Sınıf**	**Proje**
LOC	ACC	NOM
FOUT	CBO	RFC
CC	NOC	LOC
NNL	NOM	CC
WMC	RFC	NNL
	LOC	WMC
	DIT	
	NNL	
	LCOM	
	WMC	

Literatürde yapılan çalışmalara bakıldığında, test eforunu ölçmek için sınıflara karşılık gelen test kodunun büyüklüğünün incelendiği görülmüş ve bu amaçla test süit ölçütleri kullanılmıştır [21]. Bu tez çalışması kapsamında da test eforu açısından yazılımları incelemek için test süit ölçütlerinden faydalanılmıştır. Yazılımlar belirlendikten sonra ölçüt toplama evresine geçilmiş, bu ölçümler esnasında kaynak kodlar ve bu kaynak kodlara karşılık gelen test kodları hesaba dahil edilmiştir. Tez kapsamında incelenen test süit ölçütleri yaygın kullanımları sebebi ile şunlardır [21]:

- Test Kodu Satır Sayısı (TLOC)
- Test Sınıflarının Sayısı (NTClass)
- Test Metotlarının Sayısı (TM)
- Test Sınıflarının RFC Değeri (TRFC)
- Test Sınıflarının WMC Değeri (TWMC)

Yazılımların ve ölçütlerin belirlenmesinin ardından proje, sınıf ve metot seviyesinde ölçütler toplanmış ve alt seviye ölçütler için Aggarwal ve arkadaşlarının [75] önerdiği yöntemle son ölçüt değerleri belirlenmiştir. Tüm ölçümlerin sonuçları EK 1, EK 2, EK 3, EK 4, EK 5 ve EK6'da sunulmuştur.

4.3.3. Açık Kaynak Yazılımın Başarı Değerlendirme Yöntemine Karar Verilmesi

DeLone ve McLean tarafından ortaya konan IS modelinin pek çok başarı ölçüm modeline temel oluşturduğundan ön bilgi bölümünde bahsedilmişti. Çizelge 4.3'te bu model üzerinden geliştirilen, literatürde yaygın olarak kullanılan başarı kriterleri gösterilmiştir. Bu çizelgeden de görüldüğü gibi kod kalitesi başarıyı etkileyen temel faktörlerden biridir. Test edilebilirlik ise kodun kalitesi değerlendirilirken göz önünde bulundurulması gereken önemli faktörlerden biridir [33,34]. Bu noktadan hareketle yazılımın test edilebilirliğinin sistem kalitesine ve dolayısı ile yazılımın başarısına katkıda bulunduğu söylenebilir.

Çizelge 4.3. IS literatürü tarafından önerilen başarı kriterleri [34]

Başarı ölçütü	**Göstergeler**	**Hitap ettiği kesim**
Sistem ve bilgi kalitesi	Kod kalitesi (anlaşılabilirlik, eksiksizlik, taşınabilirlik, tutarlılık, bakım yapılabilirlik, güvenilirlik, verimlilik vb.) Dokümantasyon kalitesi	Kullanıcılar, geliştiriciler
Kullanıcı tatmini	Kullanıcı puanları E-posta listesindeki fikirler Kullanıcı anketleri	Kullanıcılar, geliştiriciler
Kullanım	Kullanım Kullanıcı sayısı İndirme Dağıtıma katılım Bilgi sayfasındaki görüntülenme miktarı, Paket bağlılıkları, Kod paylaşım sayısı	Geliştiriciler
Tekil ve bütüncül etkiler	Ekonomik ve diğer etkiler	Kullanıcılar, geliştiriciler

Bu tez kapsamında amaç, test edilebilirliğin ve test faaliyetlerinin önemi göz önünde bulundurularak öncelikle, test edilebilirlik ve test eforu ve ardından, test eforu ve yazılım başarısı arasında ilişki olup olmadığını araştırmaktır. Bu amaçla elde edilen test süit ölçütleri ve yazılım başarısına göre sıralanan veri seti arasında korelasyon olup olmadığına bakılmıştır.

Başarı sıralaması yapılırken kullanılacak ölçütler

Amrollahi ve arkadaşları tarafından yazılımda başarı ölçütleri üzerine yapılan literatür taramasında 17 farklı ölçüt tespit edilmiştir [76]. Bu ölçütlerden 4 tanesinin, çalışma kapsamında incelenen 24 makalenin 18'inde bahsedildiği tespit edilmiştir ve bu ölçütlerin başarıyı ölçmede etkili olduğu görüşünün yaygın olduğu sonucuna ulaşılmıştır. Bu 4 ölçüt indirilme sayısı, geliştirici sayısı, aktiflik seviyesi ve hata giderme gücüdür.

Her ne kadar bu ölçütler başarıyı ölçmek için tek başlarına kullanılsa da tek başlarına kullanılması güvenilir sonuçlar elde edilmesini sağlamaz. Örneğin, bu ölçütler ele alındığında indirilme sayısı tek başına kullanıldığında güvenilir olmayan sonuçlar

doğuracağı söylenebilir. Bunun altında yatan sebeplere ise tüm indirilmelerin gerçek kullanıma dönüşmemesi, indirilme kaynaklarının çeşitliliği sebebi ile toplam indirilme sayısının ölçülemiyor olması ve yazılımların hitap ettiği kitlelerin farklı büyüklükte olması örnek gösterilebilir [34].

Bu çalışma kapsamında, başarıyı ölçmek için ölçütleri tek başına kullanmak yerine, birlikte kullanarak gücünün ve güvenilirliğinin artırılması hedeflenmiştir ve başarıyı, geliştirici ve ürün açısından ölçen "geliştirici sayısı" ve "aktiflik seviyesi" ölçütleri tercih edilmiştir.

Başarının ölçülmesi için kullanılan yöntemin detayları

Crowston ve arkadaşları, geliştirici sayısını doğrudan kullanmak yerine geliştirici sayısının zamana bağlı değişimini kullanarak başarıyı ölçen bir yöntem önermişlerdir. [34]. Bu yöntem temel olarak açık kaynak yazılımların başarısı ile geliştiricilerin sürece olan ilgisinin doğru orantılı olduğu sonucuna dayanmaktadır. Yani bir yazılım için geliştiricilerin ilgisini çekiyor olmak başarı göstergesi iken, geliştiricilerin ilgisini kaybetmesi başarısızlık göstergesidir. Crowston ve arkadaşları ilgili makalesinde [34], bu süreci analiz etmiş ve yazılımları geliştirici sayısındaki eğilime göre 6 adet kategoriye ayırmıştır: sürekli gelişen, gelişen, durağan, gerileyen, sürekli gerileyen ve tükenmiş. Çizelge 4.4'te görüleceği üzere ilgili kategoriler, geliştirici sayısındaki artış ve azalış eğilimlerinin sıklığı üzerinden tanımlanmıştır.

Çizelge 4.4. Geliştirici sayısının zamana bağlı değişimi üzerinden tanımlanan sınıflandırma sistemi [34]

Kategori	**Tanımı ***
1. Sürekli Gelişen	Düşüş yok, artıyor: ardışık artışlar en azından serinin yarısından fazla ise.
2. Gelişen	Genellikle artıyor ama en azından bir tane düşüş var.
3. Durağan	Değişmiyor ya da artış veya azalış göstermiyor
4. Gerileyen	En fazla bir artış var ama genellikle düşüyor: ardışık azalışlar en azından serinin yarısından fazla ise.
5. Sürekli Gerileyen	Sürekli düşüyor, hiç artış gözlemlenmiyor.
6. Tükenmiş	Proje sisteminden kaldırılmış.
***Artış (azalış) sonrasındaki örnek noktası değişim göstermedi ise artış (azalış) olarak değerlendirilir.	

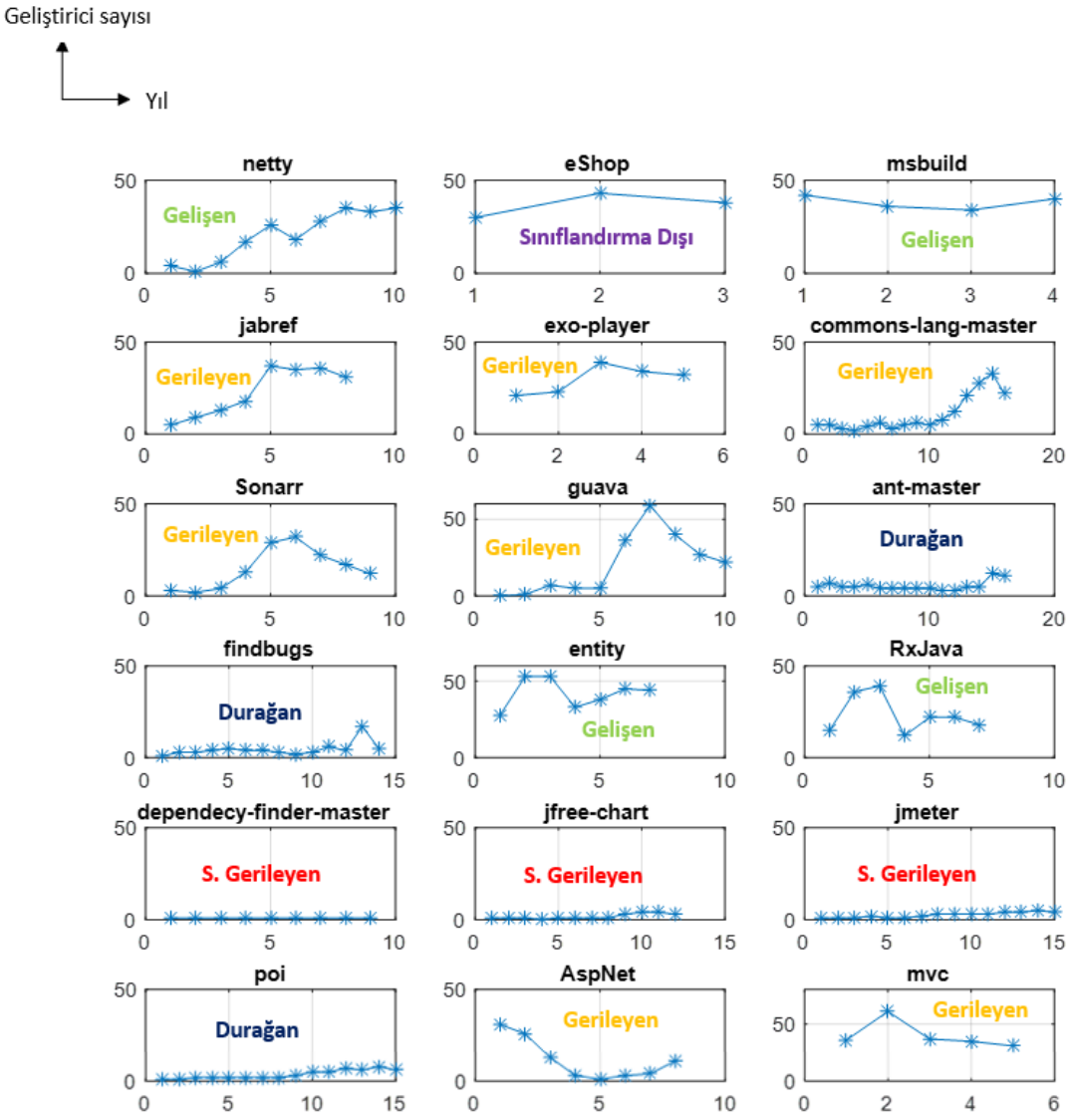
Çalışma kapsamında değerlendirilen yazılımlar ilk olarak bu çizelgeye göre gruplanmıştır. Daha sonrasında ise gruplar içerisinde aktiflik seviyesine göre sıralama yapılmıştır.

Açık kaynak yazılımların aktiflik seviyeleri ise geliştirici başına düşen katkı (İng. commit) sayılarına bakılarak değerlendirilmiştir. Bu çalışma kapsamında katkı sayısının doğrudan kullanılmamasının sebebi, incelenen yazılımların farklı büyüklükler de olması ve farklı amaçlara hizmet ediyor olmalarıdır. Bu farklılıkların yaratacağı sorunu ortadan kaldırmak ve daha dengeli bir kıyaslama yapabilmek adına katkı sayıları geliştirici sayılarına oranlanarak normalleştirme yapılması sağlanmıştır. İlk olarak, başarı kriterine göre kategorilere ayrılan yazılımlar, daha sonra aktiflik seviyelerine göre kendi içlerinde sıralanmıştır. Aktivite miktarı çok olan yazılımlar, daha başarılı olarak değerlendirilmişlerdir [77]. Ayrıca katkı sayısı geliştirici sayısına bölünerek başarı ölçütünün normalleştirilmesi sağlanmıştır.

Başarının ölçülmesi için kullanılan yöntemin uygulanması

Veri setinde yer alan yazılımlar ilk olarak uygun kategorilere konularak sınıflandırılmıştır. Sınıflandırma işlemine başlamadan önce, ilgili veriler GitHub üzerinden toplanmıştır. Buradan alınan verilerle her bir yazılım için geliştirici sayısının zamana yayılımına ait grafikler çizilmiştir. Bu grafiklerde yatay eksen yazılımın geliştirilmeye başlanmasından itibaren geçen yılları; dikey eksen ise projeye katkı sağlayan geliştirici sayılarını göstermektedir.

Şekil 4.2'de veri setindeki yazılımlara ait grafikler verilmiştir. Bu aşamada yazılımlardan biri olan Dotnet-architecture/eShopOnContainers yeterli veri noktasına sahip olmadığı için değerlendirme dışı tutulmuş ve başarı incelemesi bir eksik yazılımla yapılmıştır.



Şekil 4.2. Yazılımların başarı değerlendirmeleri

Başarı ölçüm modelinin ikinci aşamasında ise yine GitHub üzerinden veriler toplanmış; aktivite seviyesi, geliştirici başına düşen katkı sayısı üzerinden ölçülerek bir başarı sıralaması elde edilmiştir.

İki aşamalı bu model sonunda, geliştirici sayısının zamana bağlı değişimini ve aktiflik seviyesini hibrit olarak ele alan yöntem ile Çizelge 4.5'teki başarı sıralaması elde edilmiştir.

Çizelge 4.5. Açık kaynak yazılımların başarı sıralaması

Yazılım	Toplam geliştirici sayısı	Toplam katkı sayısı	Geliştirici başına düşen katkı sayısı	Başarı kategorisi	Nihai Sıralama
Apache/ant	43	14479	336.72	steady	7
Apache/commons-lang	114	5430	47.63	faller	10
Jeantessier/dependency-finder	1	2116	2116.00	const faller	16
Google/ExoPlayer	127	5589	44.01	faller	11
Findbugsproject/findbugs	32	15375	480.47	steady	6
Google/guava	180	4924	27.36	faller	14
Jabref/jabref	203	12500	61.58	faller	9
Jfree/jfreechart	12	3670	305.83	const faller	17
Apache/jmeter	7	16127	2303.86	const faller	15
Netty/netty	369	9258	25.09	riser	3
Apache/poi	12	9807	817.25	steady	5
ReactiveX/RxJava	241	5529	22.94	riser	4
Aspnet/AspNetWebStack	66	2112	32.00	faller	12
Aspnet/EntityFrameworkCore	154	7362	47.81	riser	1
Microsoft/msbuild	161	4418	27.44	riser	2
Aspnet/Mvc	150	4596	30.64	faller	13
Sonarr/Sonarr	104	7918	76.13	faller	8

4.3.4. Toplanan Verilerin Değerlendirilmesi

Spearman'ın sıralı korelasyon katsayısı, biri artan sırada olan 2 değişken arasındaki doğrusal ilişkinin gücü ve yönü hakkında fikir veren bir istatistiksel yöntemdir [78]. Tez çalışması kapsamında üzerinde çalışılan ölçütlerin aralarında ilişki olup olmadığı ve bu ilişkinin derecesi Spearman korelasyonu kullanılarak değerlendirilmiştir.

Korelasyon sonuçları JASP yazılımı [79] ile hesaplanmıştır. Elde edilen sonuçlar aşağıdaki çizelgelerde sunulmuştur. Çizelgelerin daha büyük halleri eklerde sunulmuştur.

Çizelge 4.6. Metot seviyesinde elde edilen korelasyon sonuçları

Spearman Correlations

		LOC	FOUT	CC	NNL	WMC	TLOC	TVMC
LOC	Spearman's rho	—						
	p-value	—						
FOUT	Spearman's rho	0.762***	—					
	p-value	< .001	—					
CC	Spearman's rho	0.863***	0.547*	—				
	p-value	< .001	0.019	—				
NNL	Spearman's rho	0.789***	0.486*	0.950***	—			
	p-value	< .001	0.041	< .001	—			
WMC	Spearman's rho	0.860***	0.583*	0.987***	0.925***	—		
	p-value	< .001	0.011	< .001	< .001	—		
TLOC	Spearman's rho	0.321	0.186	0.250	0.261	0.218	—	
	p-value	0.194	0.460	0.317	0.294	0.385	—	
TVMC	Spearman's rho	0.035	0.003	−0.001	0.116	−0.051	−0.050	—
	p-value	0.889	0.990	0.997	0.646	0.839	0.844	—

* p < .05, ** p < .01, *** p < .001

Çizelge 4.7. Sınıf seviyesinde elde edilen korelasyon sonuçları

Spearman Correlations

		ACC	CBO	NOC	NOM	RFC	LOC	DIT	NNL	LCOM	WMC	TLOC	TM	TRFC	TVMC
ACC	Spearman's rho	—													
	p-value	—													
CBO	Spearman's rho	−0.006	—												
	p-value	0.981	—												
NOC	Spearman's rho	0.094	−0.190	—											
	p-value	0.712	0.450	—											
NOM	Spearman's rho	0.195	−0.009	0.468	—										
	p-value	0.438	0.974	0.050	—										
RFC	Spearman's rho	−0.002	0.181	0.584*	0.509*	—									
	p-value	0.994	0.472	0.011	0.033	—									
LOC	Spearman's rho	0.388	0.001	0.385	0.773***	0.527*	—								
	p-value	0.111	1.000	0.114	< .001	0.026	—								
DIT	Spearman's rho	0.085	−0.613**	0.773***	0.291	0.279	0.223	—							
	p-value	0.738	0.007	< .001	0.241	0.263	0.374	—							
NNL	Spearman's rho	0.655**	−0.007	0.491*	0.560*	0.426	0.785***	0.356	—						
	p-value	0.003	0.980	0.039	0.017	0.079	< .001	0.147	—						
LCOM	Spearman's rho	0.417	0.073	0.394	0.752***	0.430	0.775***	0.178	0.754***	—					
	p-value	0.085	0.773	0.106	< .001	0.076	< .001	0.481	< .001	—					
WMC	Spearman's rho	0.487*	−0.034	0.478*	0.880***	0.385	0.901***	0.314	0.792***	0.820***	—				
	p-value	0.041	0.895	0.045	< .001	0.116	< .001	0.205	< .001	< .001	—				
TLOC	Spearman's rho	0.426	0.141	0.083	0.614**	0.044	0.707**	−0.020	0.657**	0.682**	0.742***	—			
	p-value	0.078	0.575	0.744	0.008	0.863	0.001	0.938	0.004	0.002	< .001	—			
TM	Spearman's rho	0.391	0.063	0.129	0.462	−0.035	0.578*	0.156	0.621**	0.575*	0.635**	0.876***	—		
	p-value	0.109	0.804	0.611	0.054	0.890	0.012	0.537	0.006	0.013	0.005	< .001	—		
TRFC	Spearman's rho	−0.116	0.633**	−0.343	0.042	0.112	−0.063	−0.577*	−0.133	0.096	−0.098	0.298	0.219	—	
	p-value	0.648	0.006	0.164	0.869	0.656	0.805	0.012	0.598	0.705	0.699	0.229	0.383	—	
TVMC	Spearman's rho	0.250	0.022	−0.028	0.478*	−0.104	0.459	0.044	0.470	0.488*	0.519*	0.895***	0.849***	0.337	—
	p-value	0.317	0.934	0.913	0.047	0.680	0.057	0.861	0.051	0.042	0.029	< .001	< .001	0.171	—

* p < .05, ** p < .01, *** p < .001

Çizelge 4.8. Proje seviyesinde elde edilen korelasyon sonuçları

Spearman Correlations

		ACC	NOM	RFC	LOC	CC	NNL	WMC	TLOC	NTC	TM	TRFC	TVMC
ACC	Spearman's rho	—											
	p-value	—											
NOM	Spearman's rho	0.264	—										
	p-value	0.289	—										
RFC	Spearman's rho	−0.145	0.662	—									
	p-value	0.567	0.004	—									
LOC	Spearman's rho	0.133	0.796	0.746	—								
	p-value	0.598	< .001	< .001	—								
CC	Spearman's rho	0.031	0.736	0.787	0.967	—							
	p-value	0.903	< .001	< .001	< .001	—							
NNL	Spearman's rho	0.603	0.559	0.459	0.641	0.630	—						
	p-value	0.008	0.016	0.055	0.004	0.005	—						
WMC	Spearman's rho	0.125	0.825	0.777	0.971	0.975	0.645	—					
	p-value	0.621	< .001	< .001	< .001	< .001	0.004	—					
TLOC	Spearman's rho	−0.254	0.146	0.529	0.453	0.455	0.061	0.379	—				
	p-value	0.309	0.563	0.026	0.061	0.059	0.811	0.122	—				
NTC	Spearman's rho	−0.130	0.278	0.604	0.412	0.393	0.054	0.360	0.866	—			
	p-value	0.607	0.264	0.009	0.091	0.107	0.830	0.143	< .001	—			
TM	Spearman's rho	−0.085	0.430	0.657	0.585	0.531	0.218	0.486	0.870	0.901	—		
	p-value	0.738	0.076	0.004	0.012	0.025	0.384	0.043	< .001	< .001	—		
TRFC	Spearman's rho	−0.346	0.375	0.659	0.552	0.622	0.184	0.544	0.779	0.690	0.781	—	
	p-value	0.159	0.126	0.004	0.019	0.007	0.464	0.021	< .001	0.002	< .001	—	
TVMC	Spearman's rho	−0.270	0.234	0.635	0.534	0.562	0.150	0.480	0.961	0.878	0.897	0.837	—
	p-value	0.279	0.348	0.006	0.024	0.017	0.552	0.046	< .001	< .001	< .001	< .001	—

Çizelge 4.9. Yazılım başarısı ile test süit ölçütleri arasındaki korelasyon sonuçları

Spearman Correlations

		TVMC	TRFC	TM	NTC	TLOC	SUCCESS
TVMC	Spearman's rho	—					
	p-value	—					
TRFC	Spearman's rho	0.806***	—				
	p-value	< .001	—				
TM	Spearman's rho	0.877***	0.740**	—			
	p-value	< .001	0.001	—			
NTC	Spearman's rho	0.855***	0.632**	0.882***	—		
	p-value	< .001	0.008	< .001	—		
TLOC	Spearman's rho	0.941***	0.721**	0.855***	0.848***	—	
	p-value	< .001	0.002	< .001	< .001	—	
SUCCESS	Spearman's rho	0.569*	0.561*	0.505*	0.407	0.488*	—
	p-value	0.019	0.021	0.041	0.106	0.049	—

* p < .05, ** p < .01, *** p < .001

4.3.5. Değerlendirme Sonuçları

Araştırmanın başında belirlenen araştırma soruları, araştırmanın uygulanmasının ardından elde edilen verilere uygun olarak izleyen alt başlıklarda yorumlanmıştır.

AS 1 – Test edilebilirlik ölçütleri

Yapılan çalışmalar sonucunda literatürde test edilebilirliği tahmin etmeye yönelik olarak önerilmiş ve sınanmış ölçütler, Çizelge 4.10'da da görüldüğü gibi, aşağıda listelenmiştir.

Çizelge 4.10. Literatürde test edilebilirliği tahmin etmek için önerilmiş ölçütler

- ACC
- CAM
- CBO
- CC
- CPM
- DAM
- DCC
- LOC
- MFA
- NMO
- NNL
- NOA
- NOC
- NoCS
- DIT
- EC
- EF
- FOUT
- I
- IC
- LCOM
- NOM
- NOO
- NOP
- NoUS
- REM
- RFC
- WMC

Çalışma kapsamında incelenen ölçütler ise Çizelge 4.11'deki gibidir.

Çizelge 4.11. Tez kapsamında incelenen test edilebilirlik ölçütleri

Metot	**Sınıf**	**Proje**
LOC FOUT CC NNL WMC	ACC CBO NOC NOM RFC LOC DIT NNL LCOM WMC	NOM RFC LOC CC NNL WMC

AS 2 – Test edilebilirlik ölçütlerinin birbirleri ile olan ilişkisi

Test edilebilirlik ölçütlerinin birbiriyle ilişkisini incelemek için elde edilen veriler Spearman istatistiği ile incelenmiştir ve sonuçları "4.3.4. Toplanan Verilerin Değerlendirilmesi" bölümünde ve eklerde sunulmuştur.

Korelasyonlar metot, sınıf ve proje seviyelerinde incelenmiştir. Korelasyon sonuçlarına göre elde edilen bulgular şu şekildedir:

Metot seviyesinde değerlendirme:

Bu aşamada LOC, FOUT, CC, NNL ve WMC ölçütleri incelenmiştir. Bu ölçütlerden LOC, CC ve NNL ölçütlerinin birbirleri ile güçlü korelasyon gösterdiği saptanmıştır. Ölçütlerin hemen hepsinin güçlü korelasyon göstermesi, metotların kaynak kodun en küçük birimlerinden biri olmasından kaynaklanıyor olabilir.

Ölçütler arasında güçlü bir korelasyon olması, geliştirme aşamasında bu ölçütlerden birinin değerini kontrol ederek diğerlerinin de ideal düzeyde tutulabileceğini ve test edilebilirlik değeri yüksek ürünler geliştirirken yol gösterici olabileceğini göstermektedir. Örneğin geliştirici kod geliştirme aşamasının çeşitli evrelerinde, geliştirdiği kodun NNL ölçüt değerine bakarak LOC, CC ve WMC değerleri hakkında da bilgi sahibi olabilecektir.

Sınıf seviyesinde değerlendirme:

Bu aşamada ACC, CBO, NOC, NOM, RFC, LOC, FOUT, CC, DIT, NNL, LCOM ve WMC ölçütleri incelenmiştir.

- ACC ve NNL ölçütleri arasında güçlü korelasyon görülmüştür.
- CBO ve DIT ölçütleri arasında güçlü korelasyon görülmüştür.
- NOC ve DIT ölçütleri arasında güçlü korelasyon görülmüştür.
- NOM ve LOC, LCOM, WMC ölçütleri arasında güçlü korelasyon görülmüştür.
- RFC ve LOC ölçütleri arasında güçlü korelasyon görülmüştür.
- LOC ve NNL, LCOM, WMC ölçütleri arasında güçlü korelasyon görülmüştür.
- NNL ve LCOM, WMC ölçütleri arasında güçlü korelasyon görülmüştür.
- LCOM ve WMC ölçütleri arasında güçlü korelasyon görülmüştür.

Ölçütler arasında güçlü bir korelasyon olması, geliştirme aşamasında bu ölçütlerden birinin değerini kontrol ederek diğerlerinin de ideal düzeyde tutulabileceğini ve test edilebilirlik değeri yüksek ürünler geliştirirken yol gösterici olabileceğini göstermektedir.

Proje seviyesinde değerlendirme:

Bu aşamada ACC, NOM, RFC, LOC, CC, NNL ve WMC ölçütleri incelenmiştir. Proje seviyesinde de ölçütlerin birbirleri ile güçlü korelasyonu olduğu ve birbirilerinden etkilendiği görülmüştür.

- ACC ve NNL ölçütleri arasında güçlü korelasyon görülmüştür.
- NOM ölçütü ile RFC, LOC, CC, WMC ölçütleri arasında güçlü korelasyon görülmüştür.
- RFC değerinin sınıf ve metot seviyelerinin aksine diğer ölçütlerden daha çok etkilendiği görülür. RFC ve LOC, CC, WMC ölçütleri arasında güçlü bir korelasyon vardır.
- LOC ve CC birbirleriyle güçlü korelasyon göstermiştir.

Proje seviyesinde kodu ve tasarımı değiştirerek ölçütlerin ideal düzeylere getirilmesi daha zor olduğu için, yazılımın en erken evrelerinden ve en küçük birimlerinden itibaren aralıklı ölçümler yapılarak ölçütlerin ideal seviyelerde tutulması, son ürün kalitesinin artmasında oldukça etkili rol oynayabilir.

Test edilebilirlik ölçütlerinin kendi aralarında güçlü bağımlılıkları olması, kaliteyi etkileyen faktörlerin aslında birbirlerini de etkilediğini göstermiştir. Örneğin büyüklüğü ve buna bağlı olarak karmaşıklığı artan yazılımların, aynı zamanda kalitesi ve test edilebildiği de etkilenmektedir. Yazılımda test edilebilirlik düzeyi artırıldığında, kalitenin de artacağı unutulmamalıdır. Ölçütler arası bağımlılıklar ürün seviyesine gelmeden yapılan iyileştirmelerin ve test edilebilirliği ideal düzeyde tutma çalışmalarının gerekliliğini göstermektedir.

AS 3 – Test edilebilirlik ölçütlerinin harcanan test eforu ile ilişkisi

Test süit ölçütleri, test eforunu tahmin etmek için önerilmiş ölçütlerdir. Bu ölçütlere bakarak harcan test eforu hakkında fikir yürütülebilir [2]. Proje seviyesinde bakıldığında, RFC, CC ve LOC ölçütlerinin test süit ölçütleri ile korelasyon gösterdiği görülmektedir. Bu durum yazılımın büyüklüğü ve karmaşıklığı arttıkça harcanan test eforunun da artacağını göstermektedir.

Sınıf seviyesinden bakıldığında NOM, LOC, NNL, LCOM ve WMC ölçütlerinin TLOC ve TM ölçütleri ile ilişkili olduğu görülür. Sınıfların büyüklükleri ve karmaşıklıkları arttıkça, sınıflara karşılık gelen test kodları büyümüş, test için harcanan efor artmıştır.

Test süit ölçütlerinin kendi aralarındaki korelasyonlara bakıldığında TLOC be TM arasında güçlü bir ilişki çıkması oldukça doğaldır. Anca, sınıf düzeyinde TLOC ve TWMC; proje düzeyinde TLOC ve RFC arasında çıkan korelasyon ilişkisi, yazılan test kodunun kalitesi ile test eforu arasında bir ilişki olabileceğini göstermektedir. Bu korelasyonların sonucuna göre, yazılan test kodunu karmaşıklığı arttıkça ve buna bağlı olarak kalitesi düştükçe, test eforu da artıyor olabilir.

AS 4 – Açık kaynak yazılımların başarısı ile harcanan test eforu arasındaki ilişki

Açık kaynak yazılımlar başarılarına göre sıralandıktan sonra, harcanan test eforu ile başarı arasında ilişki olup olmadığı proje seviyesinde incelenmiştir. Korelasyon sonuçları, yazılımın başarısı ile TLOC ve TM arasında orta düzeyde korelasyon olduğunu göstermiştir. Ne var ki başarıyı etkileyen çok fazla değişken olduğu için "test edilen yazılımlar daha başarılıdır" sonucuna kesin olarak ulaşmak mümkün değildir. Yine de ortaya çıkan korelasyon, test edilebilirlik, test eforu ve yazılım başarısı arasındaki ilişkiyi incelemek açısından anlamlıdır.

4.3.6. Değerlendirme Sonuçlarına Dayanarak Bir Yöntem Önerilmesi

Yapılan çalışmalar ve elde edilen korelasyonlar sonucunda, bir yazılımın test edilebilirliğini tahmin etmek için formül çıkarımına gidilmiştir. Test edilebilirlik indeks değerini bulmak için, sınıf düzeyinde diğer ölçütler üzerinde en çok etkiye sahip olan ölçütlerin bulunması ve ilişkili olduğu ölçütlerin sayılarına bağlı olarak bir katsayı atanması hedeflenmiştir. Elde edilen test edilebilirlik indeksinin (TI) sınanması için, test süit ölçütleri ile korelasyonuna bakılmış ve bulunan indeks değerleri ile test eforu arasındaki ilişkinin incelenmesi hedeflenmiştir.

Bu amaçla sınıf seviyesinde görülen korelasyonlar incelenmiştir. Test edilebilirlik ölçütlerinden bir küme oluşturulmuş ve bu küme içerisinde hangi ölçütlerin birbirinden etkilendiği araştırılmıştır. Sonuçlar Çizelge 4.12'de gösterilmiştir. Bu aşamada sadece test edilebilirlik ölçütlerinin kullanılmasının ve test süit ölçütlerinim devre dışı bırakılmasının sebebi, test edilebilirlik indeksinin bu ölçütler kullanılarak sınanacak olmasıdır.

Çizelge 4.12. Ölçütlerin birbiriyle ilişkisi

Ölçüt	**Etkilediği Diğer Ölçütler**	**İlişkili Ölçüt**
CBO	DIT	1
ACC	NNL, WMC	2
DIT	CBO, NOC	2
NOC	DIT, NNL, WMC	3
RFC	LOC, NOC, NOM	3
LCOM	WMC, NOM, LOC, NNL	4
NOM	RFC, LOC, NNL, LCOM, WMC	5
LOC	NNL, LCOM, WMC, NOM, RFC	5
NNL	LCOM, WMC, ACC, NOC, NOM, LOC	6
WMC	ACC, NOC, NOM, LOC, NNL , LCOM	6

Yukarıda verilen çizelgeden faydalanarak bir test edilebilirlik indeksi hesaplanması amaçlanmıştır. Modelin sade ve kolay olması için olabildiğince az ölçüt kullanılması hedeflenmiştir.

İndeks hesaplaması için Çizelge 4.12'den faydalanılmıştır. Bu amaçla ölçütler arasındaki tüm ilişkiler bir havuz olarak düşünülmüş ve 1 ölçüt için ortalama ilişki sayısı hesaplanmıştır.

Descriptive Statistics

	İlişkili Ölçüt Sayısı
Valid	10
Missing	0
Mean	3.700
Std. Deviation	1.767
Minimum	1.000
Maximum	6.000

Şekil 4.3. İlişkili ölçüt sayısı için istatistiksel değerler

Ortalama değer 3.7 olduğu için kırılım noktası 4 olarak belirlenmiş ve indeks değeri hesaplanırken kullanılmak üzere LCOM, NOM, LOC, NNL ve WMC ölçütleri seçilmiştir. Bu ölçütler test edilebilirlik ile negatif korelasyon gösterdikleri için elde ettiğimiz indeks değeri büyüdükçe test edilebilirlik azalacak ve test eforu artacaktır. Bu

nedenle test edilebilir indeks değeri ile test eforu arasında güçlü ve pozitif bir korelasyon çıkması beklenmektedir.

Ölçütlerin katsayıları belirlenirken Bansiya ve arkadaşlarının "A Hierarchical Model for Object-Oriented Design Quality Assessment" [80] isimli çalışmasında yer alan kalite matris tablosu yönteminden faydalanılmıştır. Bu yöntem doğrultusunda, bir özelliğe etki eden faktörler etki derecelerine göre puanlanır ve toplamları üzerinden bir normalleştirme yapılır.

Yöntem kapsamında bir ölçütün katsayısı belirlenirken, etkiledikleri diğer ölçütlerin sayısına dikkat edilmiştir. Bunun için basit bir yöntem izlenmiş ve ölçütler arasındaki tüm ilişkiler bir havuz gibi düşünülmüş, ölçütlerin etki derecesi ise diğer ölçütlerle olan ilişki sayılarına bakılarak değerlendirilmiştir. Önce ölçütlerin toplam ilişki sayısı bulunmuştur.

indeks değeri için toplam ilişki sayısı: 4+5+5+6+6 =26

Daha sonra seçilen her ölçüt için kendi ilişki sayısına bağlı olarak son katsayısı belirlenmiştir. Kat sayıların 0-1 aralığında dağılması için

(1 / Toplam İlişki Sayısı) * Ölçütün Diğer Ölçütlerle İlişki Sayısı

formülünden faydalanılmıştır.

Bu adımlar sonucunda test edilebilirlik indekslerini bulmaya yarayan indeks formülü şu şekilde belirlenmiştir:

TI = (4/26) * LCOM + (5/26) * (NOM+LOC) + (6/26) * (NNL+WMC)

Formülün nihai hali şu şekildedir:

TI = 0.15 * LCOM + 0.19 (NOM+LOC) + 0.23 * (NNL+WMC)

Sınıf seviyesinde gerçekleştirilen ölçümlerden yola çıkarak yazılımların TI değerleri Çizelge 4.13'teki gibi bulunmuştur:

Çizelge 4.13. Yazılımlar ve test edilebilirlik indeks değerleri

Yazılım	TI
Apache/ant	31.82
Apache/commons-lang	41.05
Jeantessier/dependency-finder	25.39
Google/ExoPlayer	33.04
Findbugsproject/findbugs	14.70
Google/guava	17.66
Jabref/jabref	22.01
Jfree/jfreechart	49.91
Apache/jmeter	29.78
Netty/netty	27.09
Apache/poi	31.63
ReactiveX/RxJava	26.15
Aspnet/AspNetWebStack	24.46
Aspnet/EntityFrameworkCore	25.71
Dotnet-architecture/eShopOnContainers	14.83
Microsoft/msbuild	48.27
Aspnet/Mvc	23.10
Sonarr/Sonarr	15.29

Elde edilen test edilebilirlik indeks değerinin yazılım test süit ölçütleri ile olan ilişkisi Spearman korelasyonu kullanılarak incelenmiştir. Korelasyon sonuçları Şekil 4.4'te gösterilmiştir.

Spearman Correlations

		TLOC	TM	TRFC	TVMC	TI
TLOC	Spearman's rho	—				
	p-value	—				
TM	Spearman's rho	0.876***	—			
	p-value	< .001	—			
TRFC	Spearman's rho	0.298	0.219	—		
	p-value	0.229	0.383	—		
TVMC	Spearman's rho	0.895***	0.849***	0.337	—	
	p-value	< .001	< .001	0.171	—	
TI	Spearman's rho	0.744***	0.620**	0.003	0.517*	—
	p-value	< .001	0.006	0.993	0.030	—

* p < .05, ** p < .01, *** p < .001

Şekil 4.4. Test edilebilirlik indeksleri ve test süit ölçütleri arasındaki korelasyon

Korelasyon sonuçları incelendiğinde, elde edilen test edilebilirlik indeksi ile test süit ölçütleri arasında güçlü bir korelasyon olduğu görülmüştür. Bu sonuç kullandığımız indeks formülünün veri setimiz için doğru çalıştığını göstermektedir.

5. SONUÇLAR VE TARTIŞMA

5.1. Genel Sonuçlar

Test edilebilirlik, yazılımın kalitesini doğrudan etkileyen en önemli faktörlerden biridir ve yüksek test edilebilirlik değerine sahip olan yazılımlar için harcanacak test eforu daha düşüktür. Doğası gereği yazılımın doğrudan ölçülebilir bir özelliği olmadığı için, test edilebilirliği analiz etmek için pek çok ölçüt ve yöntem önerilmiştir. Bu tez çalışması kapsamında, test edilebilirliğin kendisini ve önemini anlamak, yazılımların test edilebilirlik endeksini belirleyerek test eforunu kestirmek üzerinde durulmuştur. Çalışma boyunca 4 araştırma sorusu belirlenmiş ve bunlar üzerinden ilerlenmiştir.

Birinci araştırma sorusu kapsamında, literatür kapsamlı bir şekilde taranarak test edilebilirliği tahmin etmeye yönelik ölçütler bulunmuştur. Daha sonra üzerinde çalışmak içi 12 tanesi seçilmiştir.

İkinci araştırma sorusu kapsamında, önerilen test edilebilirlik değerlerinin farklı yazılım seviyelerinde birbirleri ile olan ilişkileri incelenmiştir. Yazılımın en küçük seviyesinden en büyük seviyesine kadar birbirlerini yüksek derecede etkileyen ölçütler tespit edilmiştir. Bu durum, yazılımın test edilebilirlik ve kalite düzeyini yüksek tutmak için her ölçüte tek tek bakmak yerine, birbirleri ile olan ilişkilerini kullanarak birini kontrol ederken diğerini de ideal seviyelerde tutabileceğimizi göstermektedir. Geliştirici tasarım evresinden başlayarak test edilebilirlik ve kalite seviyesini kontrol edebilir ve bu durum ürün tamamlandıktan sonra yapılan düzenleme ve kontrollere göre, daha az çaba ve maliyet gerektirebilir.

Üçüncü araştırma sorunu kapsamında, test edilebilirlik ölçütlerinin test süit ölçütleri ile ilişkisi incelenerek gereken test eforunu hangi düzeyde tahmin ettikleri, korelasyon sonuçlarına bakılarak incelenmiştir. Çıkan sonuçlar, karmaşık ve kontrol edilmesi güç kaynak kodların daha çok test eforu gerektirdiğini göstermiştir.

Dördüncü araştırma sorusu kapsamında ise harcanan test eforu ile yazılımın başarısı arasında bir ilişki olup olmadığı sorgulanmıştır. Bu amaçla açık kaynak yazılımlar, kullanılan hibrit bir modelle başarılarına göre sıralanmış, elde edilen sıralama ile test süit ölçütlerinin korelasyonu incelenmiştir. Sonuçta başarı ile harcanan test eforu arasında orta düzeyde bir korelasyon gözlenmiştir. Başarıyı etkileyen çok fazla faktör olduğu göz önünde bulundurulduğunda, “harcanan test eforu ne kadar yüksekse başarı o kadar

iyidir.” çıkarımını yapmak kesin olarak mümkün değildir; ancak bulunan sonuçlar bu ilişkinin sorgulamaya değer olduğunu göstermiştir.

Son olarak ölçütlerin birbirleri ile olan ilişkileri göz önünde bulundurularak veri setinde test edilebilirlik değerini anlamaya yönelik, ölçütlere dayalı basit bir formüle gidilmiştir. Bu formül sonucunda elde edilen değer ile test eforu arasındaki korelasyon incelenmiş ve elde edilen formülün, harcanacak test eforu hakkında fikir verici olduğu görülmüştür.

5.2. Çalışmaya ve Sonuçlara Yönelik Geçerlilik Tehditleri

Tez çalışması kapsamında, ölçütler belirlenirken bir sistematik haritalama çalışması yürütülmüştür. Bu çalışma kapsamında araştırmanın geçerliliğini tehdit eden bazı sorunlar, tez çalışmasını geçerliğini de etkilemektedir. Bu sorunlardan bazıları, anahtar sözcükler nedeniyle makale havuzu dışında kalan kaynakların varlığı, dahil etme/dışlama kriterlerinin yeterlilik derecesi ve veri çıkarımına yönelik önyargılar ve kişisel hatalardır. Yapılan bu çalışmanın tekrarlanabilir olduğunu doğrulamak için arama motoru, dijital kütüphaneler ve dâhil etme/dışlama kriterleri titizlikle tanımlanmış ve raporlanmıştır. Ayrıca makale havuzunun ilgili tüm kaynakları içermesi için kartopu yönteminden faydalanılmış, havuza eklenen makalelerin referansları ve bu makaleyi referans gösteren çalışmalar incelenmiştir.

Ölçüt kümesi belirlenirken, literatürden elde edilen test edilebilirlik ölçütlerinin büyük çoğunluğunun nesneye yönelik programlaya yönelik olması, seçilen yazılımların da Java, C# gibi nesneye yönelik programlama dilleri ile geliştirilmiş yazılımlar olmasına olmuş ve yapılan çalışmanın bu kapsamda yoğunlaşmasına neden olmuştur. Gelecek çalışmalarda veri setinin genişletilmesi ve çeşitli özelliklerinin daraltılarak incelenmesi ile nesneye yönelik programlar özelinde yapılarak nesneye yönelik programlanmış ürünlere özgü sonuçlar elde edilmesi sağlanabilir.

Çalışma kapsamında incelenen yazılımlar, açık kaynak kodlu uygulamalar olup Github gibi güvenilir ve yaygın bir veri deposundan indirilmiştir. Ölçüt ölçümleri, Understand statik kod analiz aracı [56] kullanılarak otomatik olarak gerçekleştirilmiştir. Bu araç sayesinde, insan kaynaklı hataların ortaya çıkması engellenmiştir. Yazılım ürünleri analiz edilirken seçilen klasör ve özellikler defalarca gözden geçirilmiş, yazılımlar 4 kez analiz edilerek çıkan sonuçların tutarlılığı kontrol edilmiştir. Yazılım sonucunda elde edilen ölçütlerin istatistiksel hesaplamaları JASP [79] aracı ile yapılmış ve hesaplamalarda insan kaynaklı hatanın önüne geçilmiştir.

Çalışma kapsamında programlama dilleri açısından zengin ve yeterliliğini kanıtlamış tek bir analiz aracı kullanılmıştır. Bu şekilde elde edilen ölçüt değerlerinin analizinde araçlardan ve araçların kullandığı ölçüm yöntemlerinden doğabilecek olan farklılıklardan kaçınılması hedeflenmiştir. Ancak bu durumda, elde edilen veri setindeki tüm ölçütler yerine sadece seçilen analiz aracı ile ölçülebilen değerler kullanılabilmiştir. Gelecek çalışmalarda güvenilirlik düzeyleri eşit farklı analiz araçları bulunarak tüm ölçüt değerlerinin elde edilmesi ve daha geniş bir ölçüt seti ile çalışılması hedeflenebilir.

İncelenen yazılım setinin büyüklüğü ve özellikleri nedeniyle elde edilen bulguları tüm açık kaynak yazılımlara genellemek mümkün değildir. Elde edilen sonuçlar çalışma kapsamında incelenen yazılımlar için geçerlidir.

Çalışma tasarlanırken izlenen adımların yinelenebilir ve doğrulanabilir olmasına dikkat edilmiştir. Bu şekilde çalışmanın farklı araştırmacılar tarafından da tekrarlanabilmesi güvence edilmiştir. Çalışmanın sonuçları, güvenilirliğini artırmak için istatistiksel yöntemlerle incelenip doğrulanmıştır.

6. KAYNAKLAR

1. Delivering large-scale IT projects on time, on budget, and on value, McKinsey & Company, https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value (Erişim tarihi: 13 Haziran 2019)
2. Study.com, Deming, Juran & Crosby: Contributors to TQM - Video & Lesson Transcript Study.com, https://study.com/academy/lesson/deming-juran-crosby-contributors-to-tqm.html (Erişim tarihi: 13 Haziran 2019)
3. M. Azuma, Applying ISO / IEC 9126-1 Quality Model to Quality Requirements Engineering on Critical Software Department of Industrial and Management Systems Engineering, Requir. Eng. 12, 2011
4. E. Hanoğlu, A. Tarhan, V. Garousi, Yazılım test edilebilirliği : bir sistematik literatür haritalaması Software testability : a systematic literature mapping, Ulusal Yazılım Mühendisliği Sempozyumu, 2015
5. Quandary Peak Research, Measuring Software Maintainability, https://quandarypeak.com/2015/02/measuring-software-maintainability/ (Erişim tarihi: 13 Haziran 2019)
6. U. Erdemir, U. Tekin, F. Buzluca, Nesneye Dayalı Yazılım Metrikleri ve Yazılım Kalitesi, Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu, 9-10 Ekim, İstanbul, 2008
7. Institute of Electrical and Electronics Engineers. IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Std. 610.12-1990. New York: IEEE, 1990
8. A. March, D.A. Garvin, A Note on Quality: The Views of Deming, Juran, and Crosby, Harvard Bus. Sch. Cases. (1986). doi:10.13140/RG.2.2.28698.13766
9. A. McCall, K. Richards, F. Walters, Factors in Software Quality, Volume I. Concepts and Definitions of Software Quality.
10. J.P. Miguel, D. Mauricio, G. Rodriguez, A Review of Software Quality Models for the Evaluation of Software Products, 5, 2014
11. B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod, and M. J. Merrit, Characteristics of software quality. North-Holland Publishing Company, 1978
12. T. Gorschek, K. Henningsson, P. Jönsson, S. Kågström, D. Milicic, F. Mårtensson, K. Rönkkö, P. Tomaszewski, L. Lundberg, M. Mattsson, C. Wohlin, Software quality

attributes and trade-offs Authors : Software quality attributes and trade-offs Editors :, 2005

13. S.A. Koçak, G.I. Alptekin, A.B. Bener, Bir Yazılımın Çevre ve Kalite Açısından Değerlendirilmesi için Önerilmiş Karar Alma Modeli, Ulusal Yazılım Mühendisliği Sempozyumu, 2015
14. G.J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, p. 240. Wiley Publishing, New York, 2011
15. ISO, Software engineering - Product quality-Part 1. In: Quality model 2001, International Organization for Standardization Geneva, 2001
16. ISO/IEC/IEEE International Standard - Systems and software engineering--Vocabulary," in ISO/IEC/IEEE 24765:2017(E) , vol., no., pp.1-541, 28 Aug. 2017
17. R.V Binder, Design for testability in object-oriented systems, ACM 37(9), 87–101, 1994
18. ISO, Software engineering - Product evaluation - Part 4: Process for acquirers. ISO/IEC 14598-4, Geneva, Switzerland: International Organization for Standardization, 1999
19. Y. Singh, Object-oriented software engineering, PHI Learning; 1st edition, 2012
20. R. Pushpa, H. Ratnani, Object Oriented Software Testability (OOST) Metrics Analysis, International Journal of Computer Applications Technology and Research, 2005
21. N. Yılmaz, Açık Kaynak Yazılımlarda Bakım Yapılabilirliği ve Güvenilirliği Ölçmek İçin İki Boyutlu Değerendirme Metodu, Yüksek Lisans Tezi, Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, Ankara, 2017
22. P.R. Suri, H. Singhani, Object Oriented Software Testability (OOSTe) Metrics Analysis, Int. J. Comput. Appl. Technol. Res. 4, 359–367. doi:10.7753/IJCATR0405.1006, 2005
23. A. Tahir, S. MacDonell, J. Buchan, A Study of the Relationship Between Class Testability and Runtime Properties, Maciaszek L., Filipe J. (eds) Evaluation of Novel Approaches to Software Engineering, ENASE 2014, Communications in Computer and Information Science, vol 551. Springer, Cham, 2015
24. A. Kout, F. Toure, M. Badri, An empirical analysis of a testability model for object-oriented programs, ACM SIGSOFT Softw. Eng. Notes. 36, 2011
25. A.K. Alvi, Estimation of a Set of Package Metrics to Determine the Package Testability Efforts on Open Source Software, Comput. Eng.

26. A. Tahir, S.G. Macdonell, J. Buchan, Understanding Class-level Testability Through Dynamic Analysis, (2014) 38–47. doi:10.5220/0004883400380047, 2014
27. R.A. Khan, K. Mustafa, Metric based testability model for object oriented design (MTMOOD), ACM SIGSOFT Softw. Eng. Notes. 34, 2009
28. E. Lancon, Software metrics to improve software quality in HEP, Singapore: World Scientific, 1996
29. S. Androutsellis-Theotokis et al., "Open Source Software: A Survey from 10,000 Feet,", Foundations and Trends in Technology, Information and OM, vol. 4, nos. 3–4, pp. 187–347, 2011
30. StatCounter Global Stats. (2019). Mobile Operating System Market Share Worldwide | StatCounter Global Stats. [online] Available at: http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-200901-201906 (Erişim tarihi: 13 Haziran 2019)
31. L. Morgan, P. Finnegan, Benefits and Drawbacks of Open Source Software: An Exploratory Study of Secondary Software Firms, in: Open Source Development, Adoption and Innovation, Springer, pp. 307-312, 2007
32. V. Lenarduzzi, D. Tosi, L. Lavazza, S. Morasca, Why do developers adopt open source software? Past, present and future, 15th International Conference on Open Source Systems (OSS2019) , 2019
33. W.H . DeLone, E.R. McLean, Information systems success: The quest for the dependent variable. Information Systems Research 3(1): 60–95, 1999
34. K. Crowston, J. Howison, H. Annabi, Information systems success in free and open source software development: Theory and measures. Softw. Process Improv. Practice 11, 2, 123–148, 2006
35. S. Mouchawrab, L. C. Briand and Y. Labiche, A Measurement Framework for Object-Oriented Software Testability, Journal of Information and Soft Technology, vol. 47, no. 15, pages 979-997, 2005
36. N.P.Edwards. The effect of certain modular design principles on software testability. ACN SIGPLAN NOTICES, 10(6):401–410, 1975
37. L Zhao, A new approach for software testability analysis, International Conference on Software Engineering, In Proceedings of the 28th international conference on Software engineering (ICSE '06). ACM, New York, NY, USA, 985-988, 2006
38. S. Jungmayr, Design for Testability, CONQUEST 2002
39. J. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Trans. Software Eng., vol. 18, no. 8, pp. 717–727, 1992

40. J. Voas, K. Miller. Semantic metrics for software testability. Journal of Systems and Software, 20:207–216, 1993
41. J. McGregor, S. Srinivas. A measure of testing effort. In Proceedings of the Conference on Object-Oriented Technologies, pages 129–142. USENIX Association, June 1996
42. R. Freedman, Testability of software components, IEEE Transactions on Software Engineering, 17(6):553–564, 1991
43. S. Jungmayr, Identifying test-critical dependencies. In Proceedings of the International Conference on Software Maintenance, pages 404–413, IEEE Computer Society, 2002
44. M. Bruntink and A. van Deursen, An Empirical Study into Class Testability, Journal of Systems and Software, 79(9): p. 1219–1232, 2006
45. M. Huda, Y.D.S. Arya, M.H. Khan, Metric Based Testability Estimation Model for Object Oriented Design: Quality Perspective, Journal of Software Engineering and Applications, 8, 234-243, 2015
46. Özgür Güldü, Araştırma Yöntem ve Teknikleri, https://acikders.ankara.edu.tr/pluginfile.php/88342/mod_resource/content/1/KONU%201.pdf, (Erişim tarihi: 13 Haziran 2019)
47. Araştırma Yöntem ve Tekniklerinin Seçimi, http://www.bingol.edu.tr/media/205521/sayt-bolum9-Arastirma-Yontem-ve-Tekniklerinin-Secimi.pdf, (Erişim tarihi: 13 Haziran 2019)
48. L. Badri, M. Badri, F. Toure, An empirical analysis of lack of cohesion metrics for predicting testability of classes, Int. J. Softw. Eng. Its Appl. 5, 2011
49. Y. Singh, A. Saha, Application of Artificial Neural Networks for Assessing the Testability of Object Oriented, 3,2012
50. M. Bruntink, A. van Deursen, Predicting class testability using object-oriented metrics, Proc. - Fourth IEEE Int. Work. Source Code Anal. Manip. 2004
51. F. Toure, M. Badri, L. Lamontagne, Predicting different levels of the unit testing effort of classes using source code metrics: a multiple case study on open-source software, Springer London, 2018
52. Y. Singh, A. Saha, Predicting Testability of Eclipse: A Case Study. Journal of Software Engineering, 4: 122-136, 2010.
53. G. Cantone, A. Cimitile, U. Carlini, Testability and path testing strategies, Microprocessing and Microprogramming. 21, 1987

54. H. Singhani, P.R. Suri, Testability Assessment of Object Oriented Software Using Analytic Hierarchy Process, Int. J. Comput. Sci. Issues. 12, 2015
55. S.I. Samoladas I, Gousios G, Spinellis D, The SQO-OSS quality model: measurement based open source software evaluation. Open source development, communities and quality, Int. Fed. Inf. Process. IFIP. 275 (2008) 237–48, 2008
56. Scitools.com. (2019). Understand 5.1 Release | SciTools.com. [online] Available at: https://scitools.com/understand-5-1-release/ (Erişim tarihi: 13 Haziran 2019)
57. Apache/ant, https://github.com/apache/ant, (Erişim tarihi: 13 Haziran 2019)
58. Apache/commons-lang, https://github.com/apache/commons-lang, (Erişim tarihi: 13 Haziran 2019)
59. Jeantessier/dependency-finder, https://github.com/jeantessier/dependency-finder, (Erişim tarihi: 13 Haziran 2019)
60. Google/ExoPlayer, https://github.com/google/ExoPlayer, (Erişim tarihi: 13 Haziran 2019)
61. Findbugsproject/findbugs, https://github.com/findbugsproject/findbugs, (Erişim tarihi: 13 Haziran 2019)
62. Google/guava, https://github.com/google/guava, (Erişim tarihi: 13 Haziran 2019)
63. Jabref/jabref, https://github.com/JabRef/jabref, (Erişim tarihi: 13 Haziran 2019)
64. Jfree/jfreechart, https://github.com/jfree/jfreechart, (Erişim tarihi: 13 Haziran 2019)
65. Apache/jmeter, https://github.com/apache/jmeter, (Erişim tarihi: 13 Haziran 2019)
66. Netty/netty, https://github.com/netty/netty/, (Erişim tarihi: 13 Haziran 2019)
67. Apache/poi, https://github.com/apache/poi, (Erişim tarihi: 13 Haziran 2019)
68. ReactiveX/RxJava, https://github.com/ReactiveX/RxJava, (Erişim tarihi: 13 Haziran 2019)
69. Aspnet/AspNetWebStack, https://github.com/aspnet/AspNetWebStack, (Erişim tarihi: 13 Haziran 2019)
70. Aspnet/EntityFrameworkCore, https://github.com/aspnet/EntityFrameworkCore, (Erişim tarihi: 13 Haziran 2019)
71. Dotnet-architecture/eShopOnContainers, https://github.com/dotnet-architecture/eShopOnContainers, (Erişim tarihi: 13 Haziran 2019)
72. Microsoft/msbuild, https://github.com/microsoft/msbuild, (Erişim tarihi: 13 Haziran 2019)
73. Aspnet/Mvc, https://github.com/aspnet/Mvc, (Erişim tarihi: 13 Haziran 2019)
74. Sonarr/Sonarr, https://github.com/Sonarr/Sonarr, (Erişim tarihi: 13 Haziran 2019)

75. K. Aggarwal, Y. Singh, A. Kaur, R. Malhotra, Empirical Study of Object-Oriented, Metrics. Journal of Object Technology, 5, 149-173, 2006
76. A. Amrollahi, M. Khansari, A. Manian, How Open Source Software Succeeds? A Review of Research on Success of Open Source Software, International Journal of Information & Communication Technology Research, 6(2), 67-77, 2014
77. C. Subramaniam, R. Sen, M.L. Nelson, M. L, Determinants of open source software project success: A longitudinal study, Decis. Supp. Syst. 46, 2, 576–585, 2009
78. Spearman, C., The proof and measurement of association between two things. The American journal of psychology, 15(1): sf. 72-101. 1904
79. JASP, https://jasp-stats.org/, (Erişim tarihi: 13 Haziran 2019)
80. J. Bansiya, C. G. Davis, A hierarchical model for object-oriented design quality assessment, IEEE Transactions on Software Engineering, vol. 28, no. 1, pp. 4-17, 2002

EKLER

EK 1 - Proje Seviyesinde Ölçülen Test Edilebilirlik Ölçütleri

Yazılım	ACC	NOM	RFC	LOC	CC	NNL	WMC
Apache/ant	2.06	338213	35843	112036	24022	20	24022
Apache/commons-lang	2.23	9656	5180	79577	7206	9	7206
Jeantessier/ dependency-finder	1.54	11552	7919	26520	6149	5	6149
Google/ExoPlayer	2.1	21691	9706	88091	15688	8	15688
Findbugsproject/ findbugs	2.17	55105	57488	195080	40457	11	40457
Google/guava	1.28	39681	32473	101446	17119	5	17119
Jabref/jabref	1.8	21665	9991	84652	15491	8	15491
Jfree/jfreechart	2.03	26623	32179	98335	18470	9	18470
Apache/jmeter	1.75	31774	29250	110565	19073	8	19073
Netty/netty	1.57	57238	62333	151222	30651	9	30651
Apache/poi	1.81	46092	30197	139090	28369	8	28369
ReactiveX/RxJava	1.77	27878	190509	90305	16585	8	16585
Aspnet/ AspNetWebStack	1.32	23343	38141	121469	35354	9	22380
Aspnet/ EntityFrameworkCore	1.34	22776	37153	142232	37735	8	24761
Dotnet-architecture/ eShopOnContainers	1.19	4060	12658	43332	6439	5	6375
Microsoft/msbuild	1.54	28946	42051	249585	50018	9	36292
Aspnet/Mvc	1.74	7502	23073	80094	14721	7	14657
Sonarr/Sonarr	1.28	9844	31588	73355	15872	7	15872

EK 2 - Proje Seviyesinde Ölçülen Test Süit Ölçütleri

Yazılım	TLOC	NTC	TM	TRFC	TVMC
Apache/ant	30721	495	10280	3871	4009
Apache/commons-lang	47792	503	11707	4065	5467
Jeantessier/dependency-finder	33227	865	8435	5025	3038
Google/ExoPlayer	29768	276	6411	2142	2800
Findbugsproject/findbugs	47513	2702	17070	5938	9080
Google/guava	145690	3321	56655	22962	24613
Jabref/jabref	28439	372	8882	2978	3279
Jfree/jfreechart	41986	399	7581	2527	2962
Apache/jmeter	23584	244	5244	2433	2438
Netty/netty	85995	1788	22889	21606	9971
Apache/poi	130471	1184	21570	9134	11212
ReactiveX/RxJava	184524	9210	64448	22617	25487
Aspnet/AspNetWebStack	182880	2405	27507	52450	26558
Aspnet/EntityFrameworkCore	365801	4438	43486	145094	72998
Dotnet-architecture/ eShopOnContainers	746	35	49	371	75
Microsoft/msbuild	218361	1097	24672	27651	25775
Aspnet/Mvc	170339	2801	14079	1017	22052
Sonarr/Sonarr	41440	485	3188	19488	6481

EK 3 - Sınıf Seviyesinde Ölçülen Test Edilebilirlik Ölçütleri

Yazılım	ACC	CBO	NOC	NOM	RFC	LOC	DIT	NNL	LCOM	WMC
Apache/ant	1.71	3.96	0.58	9.09	29.38	95.12	2.32	1.67	47.44	19.64
Apache/ commons-lang	1.57	1.73	0.31	13.34	21.81	123.58	1.60	1.23	51.89	30.32
Jeantessier/ dependency-finder	1.43	4.99	0.51	9.99	22.40	69.16	1.98	1.15	40.64	17.35
Google/ ExoPlayer	1.63	5.42	0.23	7.87	11.25	108.96	1.42	1.28	42.62	18.06
Findbugsproject/ findbugs	1.66	2.57	0.17	4.02	13.05	46.25	1.51	0.96	18.84	9.15
Google/guava	1.24	1.77	0.39	7.62	19.13	63.74	2.39	0.57	11.11	10.01
Jabref/jabref	1.74	4.96	0.32	5.89	8.38	64.05	1.61	1.31	36.34	12.91
Jfree/jfreechart	1.69	4.93	0.51	15.00	57.70	163.25	2.13	1.76	54.34	32.57
Apache/jmeter	1.52	3.91	0.46	8.42	23.87	84.05	2.14	1.47	55.37	15.51
Netty/netty	1.59	4.82	0.43	9.27	32.73	82.45	2.13	1.16	38.17	15.97
Apache/poi	1.69	3.65	0.45	10.73	21.73	98.45	1.73	1.30	39.42	20.31
ReactiveX/RxJava	1.44	4.33	0.47	5.95	125.81	89.68	2.04	1.26	34.65	10.82
Aspnet/ AspNetWebStack	1.51	9.77	0.31	8.46	25.20	74.92	0.48	1.17	38.71	11.05
Aspnet/ EntityFrameworkCore	1.50	11.41	0.44	7.48	22.72	81.76	0.75	0.92	37.95	12.37
Dotnet-architecture/ eShopOnContainers	1.06	4.81	0.12	4.36	13.82	46.01	0.43	0.47	27.50	4.45
Microsoft/msbuild	1.59	12.59	0.27	12.89	29.45	178.48	0.47	1.66	44.67	20.99
Aspnet/Mvc	1.70	8.80	0.18	7.09	22.00	68.91	0.35	1.14	39.90	10.48
Sonarr/Sonarr	1.29	6.48	0.30	6.27	20.11	38.54	0.55	0.78	33.85	6.60

EK 4 - Sınıf Seviyesinde Ölçülen Test Süit Ölçütleri

Yazılım	TLOC	TM	TRFC	TVMC
Apache/ant	60.06	6.99	7.90	8.18
Apache/commons-lang	108.87	8.06	8.47	11.40
Jeantessier/ dependency-finder	41.37	3.33	5.97	3.61
Google/ExoPlayer	105.95	7.80	7.82	10.22
Findbugsproject/findbugs	21.67	2.19	2.31	3.55
Google/guava	50.25	6.27	6.27	7.65
Jabref/jabref	67.65	8.06	8.11	8.92
Jfree/jfreechart	96.22	6.33	6.33	7.42
Apache/jmeter	90.27	7.17	10.06	10.05
Netty/netty	53.04	4.30	12.21	5.62
Apache/poi	103.03	6.12	7.79	9.56
ReactiveX/RxJava	25.87	2.32	2.45	2.74
Aspnet/AspNetWebStack	56.23	5.19	16.05	7.30
Aspnet/EntityFrameworkCore	59.21	6.19	20.70	5.50
Dotnet-architecture/eShopOnContainers	19.69	1.40	10.60	2.11
Microsoft/msbuild	218.43	13.84	27.08	21.96
Aspnet/Mvc	54.59	4.47	15.05	5.53
Sonarr/Sonarr	65.45	5.80	35.43	11.22

EK 5 - Metot Seviyesinde Ölçülen Test Edilebilirlik Ölçütleri

Yazılım	LOC	FOUT	CC	NNL	WMC
Apache/ant	8.32	3.46	2.13	0.57	2.16
Apache/commons-lang	7.96	2.80	2.32	0.62	2.32
Jeantessier/ dependency-finder	5.68	2.94	1.56	0.30	1.56
Google/ExoPlayer	9.58	3.38	2.24	0.58	2.25
Findbugsproject/findbugs	8.52	3.41	2.26	0.60	2.26
Google/guava	5.89	2.56	1.29	0.22	1.29
Jabref/jabref	7.67	4.04	1.88	0.46	2.06
Jfree/jfreechart	9.61	3.47	2.13	0.48	2.13
Apache/jmeter	8.11	3.92	1.82	0.50	1.85
Netty/netty	6.24	2.75	1.60	0.37	1.60
Apache/poi	7.15	3.38	1.85	0.42	1.86
ReactiveX/RxJava	7.46	3.11	1.91	0.59	1.91
Aspnet/AspNetWebStack	6.78	3.00	1.63	0.35	1.69
Aspnet/EntityFrameworkCore	7.53	3.58	1.67	0.32	1.80
Dotnet-architecture/ eShopOnContainers	6.14	2.68	1.20	0.16	1.46
Microsoft/msbuild	10.89	3.76	1.95	0.52	1.97
Aspnet/Mvc	7.23	2.97	1.75	0.41	1.80
Sonarr/Sonarr	4.02	2.56	1.30	0.23	1.41

EK 6 - Metot Seviyesinde Ölçülen Test Süit Ölçütleri

Yazılım	TLOC	TVMC
Apache/ant	6.61	1.17
Apache/commons-lang	8.47	1.31
Jeantessier/dependency-finder	10.84	1.09
Google/ExoPlayer	11.07	1.31
Findbugsproject/findbugs	5.8606	1.6086
Google/guava	7.1438	1.3065
Jabref/jabref	6.59	1.1
Jfree/jfreechart	13.823	1.173
Apache/jmeter	10.94	1.4
Netty/netty	10.32	1.31
Apache/poi	15.06	1.58
ReactiveX/RxJava	9.3452	1.88
Aspnet/AspNetWebStack	6.3781	1.3863
Aspnet/EntityFrameworkCore	5.42	1.43
Dotnet-architecture/eShopOnContainers	8.19	1.45
Microsoft/msbuild	12.24	1.73
Aspnet/Mvc	7.24	1.39
Sonarr/Sonarr	5.38	1.52

EK 7 - Proje Seviyesinde Korelasyon Sonuçları

Spearman Correlations

		ACC	NOM	RFC	LOC	CC	NNL	WMC	TLOC	NTC	TM	TRFC	TVMC
ACC	Spearman's rho	—											
	p-value	—											
NOM	Spearman's rho	0.264	—										
	p-value	0.289	—										
RFC	Spearman's rho	−0.145	0.662	—									
	p-value	0.567	0.004	—									
LOC	Spearman's rho	0.133	0.796	0.746	—								
	p-value	0.598	< .001	< .001	—								
CC	Spearman's rho	0.031	0.736	0.787	0.967	—							
	p-value	0.903	< .001	< .001	< .001	—							
NNL	Spearman's rho	0.603	0.559	0.459	0.641	0.630	—						
	p-value	0.008	0.016	0.055	0.004	0.005	—						
WMC	Spearman's rho	0.125	0.825	0.777	0.971	0.975	0.645	—					
	p-value	0.621	< .001	< .001	< .001	< .001	0.004	—					
TLOC	Spearman's rho	−0.254	0.146	0.529	0.453	0.455	0.061	0.379	—				
	p-value	0.309	0.563	0.026	0.061	0.059	0.811	0.122	—				
NTC	Spearman's rho	−0.130	0.278	0.604	0.412	0.393	0.054	0.360	0.866	—			
	p-value	0.607	0.264	0.009	0.091	0.107	0.830	0.143	< .001	—			
TM	Spearman's rho	−0.085	0.430	0.657	0.585	0.531	0.218	0.486	0.870	0.901	—		
	p-value	0.738	0.076	0.004	0.012	0.025	0.384	0.043	< .001	< .001	—		
TRFC	Spearman's rho	−0.346	0.375	0.659	0.552	0.622	0.184	0.544	0.779	0.690	0.781	—	
	p-value	0.159	0.126	0.004	0.019	0.007	0.464	0.021	< .001	0.002	< .001	—	
TVMC	Spearman's rho	−0.270	0.234	0.635	0.534	0.562	0.150	0.480	0.961	0.878	0.897	0.837	—
	p-value	0.279	0.348	0.006	0.024	0.017	0.552	0.046	< .001	< .001	< .001	< .001	—

EK 8 - Sınıf Seviyesinde Korelasyon Sonuçları

Spearman Correlations

		ACC	CBO	NOC	NOM	RFC	LOC	DIT	NNL	LCOM	WMC	TLOC	TM	TRFC	TVMC
ACC	Spearman's rho	—													
	p-value	—													
CBO	Spearman's rho	−0.006	—												
	p-value	0.981	—												
NOC	Spearman's rho	0.094	−0.190	—											
	p-value	0.712	0.450	—											
NOM	Spearman's rho	0.195	−0.009	0.468	—										
	p-value	0.438	0.974	0.050	—										
RFC	Spearman's rho	−0.002	0.181	0.584*	0.509*	—									
	p-value	0.994	0.472	0.011	0.033	—									
LOC	Spearman's rho	0.388	0.001	0.385	0.773***	0.527*	—								
	p-value	0.111	1.000	0.114	< .001	0.026	—								
DIT	Spearman's rho	0.085	−0.613**	0.773***	0.291	0.279	0.223	—							
	p-value	0.738	0.007	< .001	0.241	0.263	0.374	—							
NNL	Spearman's rho	0.655**	−0.007	0.491*	0.560*	0.426	0.785***	0.356	—						
	p-value	0.003	0.980	0.039	0.017	0.079	< .001	0.147	—						
LCOM	Spearman's rho	0.417	0.073	0.394	0.752***	0.430	0.775***	0.178	0.754***	—					
	p-value	0.085	0.773	0.106	< .001	0.076	< .001	0.481	< .001	—					
WMC	Spearman's rho	0.487*	−0.034	0.478*	0.880***	0.385	0.901***	0.314	0.792***	0.820***	—				
	p-value	0.041	0.895	0.045	< .001	0.116	< .001	0.205	< .001	< .001	—				
TLOC	Spearman's rho	0.426	0.141	0.083	0.614**	0.044	0.707**	−0.020	0.657**	0.682**	0.742***	—			
	p-value	0.078	0.575	0.744	0.008	0.863	0.001	0.938	0.004	0.002	< .001	—			
TM	Spearman's rho	0.391	0.063	0.129	0.462	−0.035	0.578*	0.156	0.621**	0.575*	0.635**	0.876***	—		
	p-value	0.109	0.804	0.611	0.054	0.890	0.012	0.537	0.006	0.013	0.005	< .001	—		
TRFC	Spearman's rho	−0.116	0.633**	−0.343	0.042	0.112	−0.063	−0.577*	−0.133	0.096	−0.098	0.298	0.219	—	
	p-value	0.648	0.006	0.164	0.869	0.656	0.805	0.012	0.598	0.705	0.699	0.229	0.383	—	
TVMC	Spearman's rho	0.250	0.022	−0.028	0.478*	−0.104	0.459	0.044	0.470	0.488*	0.519*	0.895***	0.849***	0.337	—
	p-value	0.317	0.934	0.913	0.047	0.680	0.057	0.861	0.051	0.042	0.029	< .001	< .001	0.171	—

* p < .05, ** p < .01, *** p < .001

EK 9 - Metot Seviyesinde Korelasyon Sonuçları

Spearman Correlations

		LOC	FOUT	CC	NNL	WMC	TLOC	TVMC
LOC	Spearman's rho	—						
	p-value	—						
FOUT	Spearman's rho	0.762***	—					
	p-value	< .001	—					
CC	Spearman's rho	0.863***	0.547*	—				
	p-value	< .001	0.019	—				
NNL	Spearman's rho	0.789***	0.486*	0.950***	—			
	p-value	< .001	0.041	< .001	—			
WMC	Spearman's rho	0.860***	0.583*	0.987***	0.925***	—		
	p-value	< .001	0.011	< .001	< .001	—		
TLOC	Spearman's rho	0.321	0.186	0.250	0.261	0.218	—	
	p-value	0.194	0.460	0.317	0.294	0.385	—	
TVMC	Spearman's rho	0.035	0.003	−0.001	0.116	−0.051	−0.050	—
	p-value	0.889	0.990	0.997	0.646	0.839	0.844	—

* p < .05, ** p < .01, *** p < .001

EK 10 - Yazılım Başarısı ile Test Süit Ölçütleri Arasındaki Korelasyon Sonuçları

Spearman Correlations

		TVMC	TRFC	TM	NTC	TLOC	SUCCESS
TVMC	Spearman's rho	—					
	p-value	—					
TRFC	Spearman's rho	0.806***	—				
	p-value	< .001	—				
TM	Spearman's rho	0.877***	0.740**	—			
	p-value	< .001	0.001	—			
NTC	Spearman's rho	0.855***	0.632**	0.882***	—		
	p-value	< .001	0.008	< .001	—		
TLOC	Spearman's rho	0.941***	0.721**	0.855***	0.848***	—	
	p-value	< .001	0.002	< .001	< .001	—	
SUCCESS	Spearman's rho	0.569*	0.561*	0.505*	0.407	0.488*	—
	p-value	0.019	0.021	0.041	0.106	0.049	—

* p < .05, ** p < .01, *** p < .001

EK 11 - Tezden Türetilmiş Yayınlar

- Hanoğlu, E., Tarhan A. Ve Garousi V. "Yazılım test edilebilirliği: bir sistematik literatür haritalaması", Ulusal Yazılım Mühendisliği Sempozyumu'16, Vol-1721, Çanakkale, Türkiye (2016)
- Hanoğlu E., Tarhan A. "Açık Kaynak Yazılımlarda Başarı ve Test Eforu Arasındaki İlişki: Deneysel Bir Çalışma", Uluslararası Bilgisayar Bilimleri ve Mühendisliği Konferansı UBMK 2019, Türkiye (2019) (Makale kabul edildi, düzeltme aşamasında)

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS TEZ ÇALIŞMASI ORJİNALLİK RAPORU

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI BAŞKANLIĞI'NA

Tarih: 17../07/2019

Tez Başlığı / ~~Konusu~~: AÇIK KAYNAK YAZILIM PROJELERİ İÇİN İÇSEL ÜRÜN ÖZELLİKLERİNE VE METRİKLERİNE DAYALI BİR TEST EDİLEBİLİRLİK ANALİZİ YÖNTEMİ

Yukarıda başlığı/~~konusu~~ gösterilen tez çalışmamın a) Kapak sayfası, b) Giriş, c) Ana bölümler d) Sonuç kısımlarından oluşan toplam ....74... sayfalık kısmına ilişkin, 12../07./2019 tarihinde ~~şahsım~~/tez danışmanım tarafından *Turnitin* adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı %...8..... 'dir.

Uygulanan filtrelemeler:

1- Kaynakça hariç
2- Alıntılar hariç / ~~dahil~~
3- 5 kelimeden daha az örtüşme içeren metin kısımları hariç

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü Tez Çalışması Orjinallik Raporu Alınması ve Kullanılması Uygulama Esasları'nı inceledim ve bu Uygulama Esasları'nda belirtilen azami benzerlik oranlarına göre tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Gereğini saygılarımla arz ederim.

Tarih ve İmza
17.07.2019

Adı Soyadı:	Ebru Hanoğlu
Öğrenci No:	N14325319
Anabilim Dalı:	Bilgisayar Mühendisliği
Programı:	Bilgisayar Mühendisliği Tezli Yüksek Lisans
Statüsü:	☒ Y.Lisans ☐ Doktora ☐ Bütünleşik Dr.

DANIŞMAN ONAYI

UYGUNDUR.

Dr. Öğr. Üyesi Adnan Özsoy
(Unvan, Ad Soyad, İmza)

ÖZGEÇMİŞ

Adı Soyadı	: Ebru Hanoğlu
Doğum yeri	: Ankara
Doğum tarihi	: 04.05.1990
Medeni hali	: Evli
Yazışma adresi	: Hacettepe Üniversitesi Bilgisayar Mühendisliği Bölümü
Telefon	: (0312) 297 75 00
Elektronik posta adresi	: ebruhanoglu90@gmail.com
Yabancı dili	: İngilizce

EĞİTİM DURUMU

Lise	: Ankara Atatürk Anadolu Lisesi
Lisans	: Hacettepe Üniversitesi / Bilgisayar Mühendisliği
Yüksek Lisans	: Hacettepe Üniversitesi / Bilgisayar Mühendisliği

İş Tecrübesi

2018- Halen	SIEMENS – Kalite Güvence Test Mühendisi