MOBİL UYGULAMALARIN EVRİMİNDE KALİTENİN GELİŞİMİ

QUALITY IN THE EVOLUTION OF MOBILE APPLICATIONS

BAHAR GEZİCİ

DR. ÖĞR. ÜYESİ AYÇA TARHAN
Tez Danışmanı
DOÇ. DR. OUMOUT CHOUSEINOGLOU
Tez Eş Danışmanı

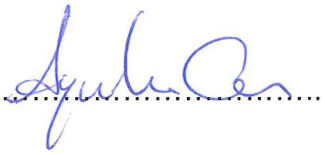
Hacettepe Üniversitesi
Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin
Bilgisayar Mühendisliği Anabilim Dalı için Öngördüğü
YÜKSEK LİSANS TEZİ olarak hazırlanmıştır.

Haziran 2018

BAHAR GEZİCİ' nin hazırladığı **"Mobil Uygulamaların Evriminde Kalitenin Gelişimi"** adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI'** nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. Ali H. DOĞRU
Başkan ..... 

Dr. Öğr. Üyesi Ayça TARHAN
Danışman ..... 

Doç. Dr. Aysu Betin CAN
Üye ..... 

Dr. Öğr. Üyesi Burcu CAN BUĞLALILAR
Üye ..... 

Dr. Öğr. Üyesi Barbaros YET
Üye ..... 

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

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU
Fen Bilimleri Enstitüsü Müdürü

YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) 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 suretlerinin üniversiteye teslim etmeyi taahhüt ederim.

- ☐ **Tezimin/Raporumun tamamı dünya çapında erişime açılabilir ve bir kısmı veya tamamının fotokopisi alınabilir.**
  (Bu seçenekle teziniz arama motorlarında indekslenebilecek, daha sonra tezinizin erişim statüsünün değiştirilmesini talep etseniz ve kütüphane bu talebinizi yerine getirse bile, teziniz arama motorlarının önbelleklerinde kalmaya devam edebilecektir.)
- ☒ **Tezimin/Raporumun** 31/12/2018 **tarihine kadar erişime açılmasını ve fotokopi alınmasını (İç kapak, Özet, İçindekiler ve Kaynakça hariç) istemiyorum.**
  (Bu sürenin sonunda uzatma için başvuruda bulunmadığım takdirde, tezimin/raporumun tamamı her yerden erişime açılabilir, kaynak gösterilmek şartıyla bir kısmı veya tamamının fotokopisi alınabilir.)
- ☐ **Tezimin/Raporumun ..... tarihine kadar erişime açılmasını istemiyorum, ancak kaynak gösterilmek şartıyla bir kısmı veya tamamının fotokopisinin alınmasını onaylıyorum.**
- ☐ **Serbest Seçenek/Yazarın Seçimi**

08/06/2018

(İmza)

Öğrencinin Adı Soyadı

Bahar GEZİCİ

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 tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversitede veya başka üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

04/06/2018

BAHAR GEZİCİ

ÖZET

MOBİL UYGULAMALARIN EVRİMİNDE KALİTENİN GELİŞİMİ

Bahar GEZİCİ
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Danışmanı: Dr. Öğr. Üyesi Ayça TARHAN
Tez Eş Danışmanı: Doç. Dr. Oumout CHOUSEINOGLOU
Haziran 2018, 110 sayfa

Mobil uygulamalar, kullanıcı gereksinimlerini karşılama ihtiyacından hızlı bir şekilde büyüyerek sürekli gelişmekte ve buna bağlı olarak karmaşık yazılım sistemleri haline gelmektedir. Bununla birlikte kullanıcı gereksinimlerinin karşılanması, yazılım kalitesini ve performansını düşürebilecek negatif örüntü (antipattern) olarak bilinen kötü tasarım tercihlerine yol açabilir. Bu sebeple, uygulamaların özelliklerinin algılanması ve izlenmesi, bakım ve geliştirmeyi kolaylaştırmak için önemli faaliyetler olup geliştiricileri uygulamalarını yeniden yapılandırmaya ve böylece kalitelerini yükseltmeye yönlendirebilir. Mobil uygulamaların kalitesi geliştirici ve kullanıcılar ile uygulama mağazaları için büyük bir önem arz etmektedir. Bir mobil yazılım ürününün kalitesini incelemeye yönelik ve mobil uygulamalara özgü birkaç teknik olsa da gerçek bir uygulama mağazasındaki mobil uygulamanın potansiyel başarısını ön görmek için geçerliliği kabul görmüş etkili bir yol bulunmamaktadır. Dolayısıyla, evrim boyunca sürekli gelişmekte olan mobil yazılımların kalitesinin izlenmesi faaliyeti, araştırılması gereken cazip bir problem haline gelmiştir. Bu tez çalışmasında, literatürdeki mevcut çalışmalar ışığında, açık kaynak kodlu mobil uygulamaların evriminde kalite gelişimini izlemek amaçlanmıştır. Bu amaca yönelik olarak, açık kaynak kodlu mobil uygulamaların iç (kod tabanlı) ve dış (toplum tabanlı) kalite özelliklerinin gelişimi, keşifsel bir yaklaşımla değerlendirilmiştir. Kod tabanlı

değerlendirme için, en güncel kalite modeli olan ISO/IEC 25010 standardı temel alınarak nesneye dayalı tasarım metriklerinin ölçümleri gerçekleştirilirken toplum tabanlı değerlendirme için, açık kaynak yazılımların (AKY) başarısını ölçmede rehber niteliğinde olan DeLone ve McLean modeli ışığında Github, Sourceforge vb. veri depolarından çıkarılan toplum tabanlı metrikler analiz edilmiştir. Analiz sonucunda iç kalitenin sürümler boyunca genel olarak arttığı gözlenirken dış kalitenin azaldığı gözlenmiştir. Ayrıca, Spearman korelasyon analizi ile uygulamaların iç ve dış kaliteleri arasındaki ilişki analiz edilmiş ve aralarında anlamlı bir ilişki gözlenmemiştir.

Anahtar Kelimeler: Açık kaynak kodu, mobil yazılım, yazılım evrimi, yazılım kalitesi, iç kalite, dış kalite, C&K metrik seti, Lehman yasaları, ISO/IEC 25010, DeLone ve McLean Modeli

ABSTRACT

QUALITY IN THE EVOLUTION OF MOBILE APPLICATIONS

Bahar GEZİCİ

Master of Science, Computer Engineering Department

Supervisor: Asst. Prof. Dr. Ayça TARHAN

Cosupervisor: Assoc. Prof. Dr. Oumout CHOUSEINOGLOU

June 2018, 110 pages

Mobile applications are becoming complex software systems as they rapidly evolve and grow constantly to meet user requirements. However, satisfying these requirements may lead to poor design choices known as ‘antipatterns’ that can degrade software quality and performance. Therefore, perception and monitoring of the characteristics of mobile applications are important activities to facilitate maintenance and development, so that developers are directed to restructure their practices and upgrade their qualifications. The quality of mobile applications is of great importance for developers, users and application stores. Although there are a number of mobile application specific techniques for analysing the quality of a mobile software product, there is no accepted and valid way to predict the potential success of a mobile application in a real app store. Thus, the monitoring of the quality of ever-evolving mobile software throughout the evolution has become an attractive problem to investigate. In this thesis, we aim to monitor the quality in the evolution of open source mobile applications in the light of existing studies in the literature. For this purpose, the development of internal (code-based) and external quality (community-based) features of open source mobile applications has been evaluated with an exploratory approach. For code-based evaluation, object-based design metrics are measured based on the most recent quality model ISO/IEC 25010. For community-based evaluation, a number of community based metrics which are extracted from data repositories such as

Github and Sourceforge have been analyzed based on the DeLone and McLean model, which is a guideline for measuring the success of open source software. As a result of the analysis, it was observed that the internal quality increased generally during the releases, while the external quality decreased. In addition, the relationship between internal and external qualities of the applications was analyzed by Spearman correlation analysis and no significant relation was observed between them.

Keywords: Open source code, mobile software, software evolution, software quality, internal quality, external quality, C&K metric set, Lehman Laws, ISO/IEC 25010, DeLone and McLean Model

TEŞEKKÜR

Öncelikle bu tez çalışmamın yürütülmesi sırasında desteğini esirgemeyen, tezimin her aşamasında kıymetli bilgi, birikim ve tecrübeleri ile bana yol gösteren, beni her zaman güler yüz ile karşılayan ve gerektiğinde fazla mesai harcayan değerli danışman hocalarım Dr. Öğr. Üyesi Ayça TARHAN ve Doç. Dr. Oumout CHOUSEINOGLOU'na sonsuz teşekkür ve saygılarımı sunarım. Ayrıca değerli tez jüri üyeleri Prof. Dr. Ali H.DOĞRU, Doç. Dr. Aysu Betin CAN, Dr. Öğr. Üyesi Burcu CAN BUĞLALILAR ve Dr. Öğr. Üyesi Barbaros YET'e teşekkürü bir borç bilirim.

Bu çalışmam boyunca yardım ve desteklerini benden esirgemeyen, sevgili çalışma arkadaşlarıma teşekkür ederim.

Hayatım boyunca maddi manevi her türlü destekleriyle beni hiçbir zaman yalnız bırakmayan, bana her koşulda güvenen ve çalışmam sırasında umutsuzluğa kapıldığım her anda moral ve motivasyon kaynağı olan aileme de sonsuz teşekkürlerimi sunarım.

İÇİNDEKİLER

Sayfa

ÇİZELGELER

Sayfa

ŞEKİLLER

Sayfa

SİMGE VE KISALTMALAR

Simgeler

$\sum$	Toplam Sembolü
Kendall tau	Mann-Kendall testi korelasyon katsayı değeri
S	Mann-Kendall testi istatistik değeri
p	İstatistik testleri için anlamlılık değeri
r_s	Spearman korelasyon katsayı değeri

Kısaltmalar

AS	Araştırma Sorusu
AKY	Açık Kaynak Yazılım
GPL	Genel Kamu Lisansı
TCO	Toplam Sahip Olma Maliyeti
FOSS	Özgür Açık Kaynak Kodlu Yazılım
OHA	Open Handset Alliance
C&K	Chidamber ve Kemerer
MCC	McCabe Döngüsel Karmaşıklığı
NOC	Alt Sınıf Sayısı
DIT	Kalıtım Ağacının Derinliği
RFC	Sınıfın Tetiklediği Metot Sayısı
WMC	Sınıfın Ağırlıklı Metot Sayısı
CBO	Nesne Sınıfları Arasındaki Bağımlılık
LCOM	Metotlardaki Uyum Eksikliği
LOC	Kod Satır Sayısı
CLOC	Yorum Satır Sayısı
Ca	İçsel Bağımlılık
Ce	Dışa Bağımlılık
NOR	Eleştirici Sayısı
UR	Kullanıcı Oyları
NOF	Geri bildirim Sayısı
NOD	Katkıda bulunan Geliştirici Sayısı

CN	Değişiklik Sayısı
TBR	Sürümler arasındaki Zaman
SI	Başarı İndeksi

1. GİRİŞ

1.1. Problemin Tanımı

Son on yılda mobil uygulamalara yönelik talep, uygulama mağazalarının mobil uygulama ürünlerinin yayılması için önde gelen bir kanal olmasıyla birlikte artmıştır [1]. Uygulama mağazalarında milyonlarca mobil uygulama mevcuttur ve bu uygulamalar milyonlarca kez indirilmiştir [2]. 2017 yılı itibariyle Google Play'de 3,5 milyon, Apple App Store'da 2,2 milyon, Amazon App Store'da ise yaklaşık 385 bin mobil uygulama bulunmaktadır [3]. Mobil uygulama sayısının bu denli artması ve pazarda uygulamaların devamlılığının sağlanması, kullanıcı beklentilerini karşılayan ve rakiplerin tekliflerini aşan fonksiyonel uygulamalarla pazarda nasıl başarılı olunabileceği sorusunu doğurmaktadır [4].

Bir mobil yazılım ilk sürümü yayımlandıktan sonra, uzun yıllar geliştirilmeye devam edilir. Bu süre boyunca kaçınılmaz olarak yeni gereksinimleri karşılamak, yeni platformlara uyum sağlamak, hataları düzeltmek veya yazılım tasarımını iyileştirmek için atılan adımlarla başa çıkmak gerekir. Yapılan çalışmalardan bir kısmı yazılım bakım ve geliştirme ile ilişkili maliyetlerin, toplam maliyetin %50'sinden %90'ına değişmekte olduğunu [5-7]; başka çalışmalar ise bu maliyetlerin, ilk yazılım sürümüne ait maliyetin birkaç katına kadar çıktığını göstermektedir [8]. Yazılım üretme maliyetlerini azaltmak için, hem yöneticiler hem de geliştiriciler yazılım geliştirmeyi yönlendiren faktörleri anlamalı ve değişiklikleri kolaylaştıran, yazılımın bozulmaması için gereken önleyici adımlar atmalıdır. Mobil uygulamaların popülerliği katlanarak arttıkça, geliştirme maliyetini düşürmek ve başarılı bir geliştirme ortaya koymak amacıyla, mobil uygulamaların evrimi üzerine araştırma yapma gerekliliği doğmuştur [9-11].

Yazılım kalitesi, müşteri gözünden kalite ve yazılım üreticisi gözünden kalite olmak üzere iki temel bakış açısı ile ele alınmaktadır. Müşteriler genel olarak satın aldıkları yazılımların kolay kullanılabilir, hatasız, tüm isteklerini karşılayan ve yeterli performansa sahip olmasını ister. Buna karşılık yazılım üreticileri ise geliştirme ve bakım maliyetlerinin düşük ve üretilen yazılımın parçalarının bir sonraki projelerinde de kullanılabilir olmasını ister. ISO 25010, yazılım ürünlerinin kalitesini anlatan ve sınıflandıran uluslararası bir standarttır [12] ve bu standartta geliştirici gözünden yazılımın iç kalitesi, "belirlenen ve hedeflenen gereksinimleri karşılama

kabiliyetine sahip bir varlığın özelliklerinin bütünü" olarak tanımlanmıştır. Yazılımın iç kalitesi, kötü tasarım ve uygulama seçeneklerinin yazılım içine istemsizce eklenmesi nedeniyle yazılımın evrimi boyunca gerileyebilir. ‘Memnuniyet’ terimini kullanan ISO 25010’a göre dış kalite, "yazılımın belirli bir kullanım bağlamında kullanıcıları tatmin edebilme kabiliyeti" anlamına gelir. Büyük uygulama mağazalarında, her ürünün mağazada yer alması gereken geliştirme politikaları ve yayınlama kuralları vardır [4]. Bir uygulamanın piyasada onaylanması ve yer alması için bir değerlendirme süreci gerçekleştirilmelidir. Dahası, müşteriler (yani, son kullanıcılar), geri bildirimlerini uygulama mağazalarında sunma olanağına sahip olarak, oyları ve yorumları vasıtasıyla ürünleri değerlendirebilir. Bu bağlamda ürün kalitesi geliştiriciler, kullanıcılar ve uygulama pazarları için büyük bir endişe kaynağıdır. Mobil geliştiriciler, son kullanıcıyı memnun eden uygulamalar üretmek isterler, böylece daha geniş bir müşteri yelpazesini etkileyebilir ve potansiyel olarak daha yüksek gelir elde edebilirler [6]. Kullanıcılar, beklentilerini karşılayan ve cihazlarına zarar vermeyen veya gizliliklerini riske atmayan, yüksek kaliteli uygulamalar almak isterler. Uygulama mağazaları ise satış ve prestijlerinde yüksek bir kalite düzeyini korumak ister; düşük kaliteli uygulamalar pazara kabul edilmeyebilir veya servis sağlayıcısı tarafından kaldırılabilir. Örneğin, 2013 yılında AppBrain [13] Google Play Store'da bulunan uygulamaların yaklaşık %20'sinin “düşük kaliteli uygulamalar” olduğunu belirtmiş ve Google'ın bunları uygulama mağazasından yaklaşık dörtte bir oranında kaldırdığını belirtmiştir. Özetle, mobil uygulamaların popülerliği arttıkça, evrim maliyetini azaltmak ve başarılı bir gelişim sürecini desteklemek için mobil uygulamaların evrimi üzerine bir çalışma yapılması gerekliliği doğmuştur.

1976’da Lehman ve Belady [14], yazılımın evrimi üzerine somut deneysel çalışmalar gerçekleştirmiş ve bir çalışmada yazılım gelişimini karakterize eden bir dizi yasa önermiştir [15, 16]. Bu yasalar; ‘1-Sürekli değişim’, ‘2-Artan karmaşıklık’, ‘3-Öz denetim’, ‘4-Durağanlığın korunması’, ‘5-Benzerliklerin korunması’, ‘6-Sürekli büyüme’, ‘7-Azalan kalite’ ve ‘8-Geri bildirim sistemi’ ile ilişkilidir. Lehman’ın yasaları, o zamanki ticari yazılımların deneysel gözlemlerine dayanarak oluşturulmuştur. Bununla birlikte, yazılım mühendisliği topluluğunda bu yasaları, yazılım evrim problemlerini anlamaya yönelik ve yenilikçi çözümler öneren bir rehber olarak gören bir fikir birliği mevcuttur. Geçen zamanda yazılım mühendisliği

disiplini gelişse de yazılımın evrimi üzerine yapılan deneysel çalışmalarla Lehman yasalarının günümüzdeki geçerliliği sınanmakta ve bazılarının hala geçerli olduğu gözlenmektedir [17, 18]. Lehman'ın yasaları, bir yazılım ürününün ömrü boyunca yazılım geliştirme dinamiklerini temsil etmektedir. Büyük masaüstü uygulamalarının gelişimi açısından Lehman yasaları ile çok fazla araştırma yapılmıştır. Ancak, bu yasalar mobil uygulamaların evrim süreci ile ilişkili çok az sayıda araştırmada gözlenmektedir [11, 17].

Yukarıdaki ihtiyaçlardan hareketle bu tez çalışmasında, mobil uygulamaların evrimi boyunca iç ve dış kalitenin gelişimini anlamak hedeflenmiştir. Bu hedefe ulaşmak amacıyla, iç kalite analizinde ISO 25010 standardı temel alınarak Lehman'ın [15, 16] sekiz yazılım geliştirme yasasından kalite ile ilişkili olan ikinci, altıncı ve yedinci yasası kapsamında oluşturulan hipotezler test edilirken dış kalite için, DeLone ve McLean modeli [19] temel alınarak bir takım toplum tabanlı metriklerin ölçümü gerçekleştirilmiştir. Son aşamada ise iç kalite bulguları ve dış kalite bulguları arasındaki ilişki analiz edilmiştir.

İç kalite analizi için, toplamda 61 sürümü içeren üç mobil uygulamanın (Keepassdroid, UPM ve PasswdSafe) sürümleri boyunca iç kalitenin gelişimi ve dış kalite analizi için toplamda 47 sürümü içeren aynı üç mobil uygulamanın dış kalite gelişimi incelenmiştir. İç kalite için, Keepassdroid ve PasswdSafe uygulamalarının son beş yılda geliştirilen 12 ve 33 sürümü ile UPM uygulamasının tüm sürümleri (16) analiz edilmiştir. Dış kalite analizi için, kullanıcı yorumları temel alınarak sürüm seçimi gerçekleşmiştir. Buna göre, Keepassdroid'in 9, UPM'in 5 ve PasswdSafe uygulamasının 33 sürümü çalışmaya dâhil edilmiştir.

Çalışmaya özel olarak oluşturulan araştırma soruları (AS) aşağıda verilmiştir. Araştırma sorularının (AS) yanıtlanması için hipotezler oluşturulmuş ve bu hipotezler, yukarıda sözü edilen üç açık kaynak kodlu mobil uygulamanın sürümleri bazında toplanan yazılım tasarım metrikleri ve toplum tabanlı metrikler üzerinde istatistiksel analizler yapılarak test edilmiştir.

- **AS 1.** Mobil uygulamaların evriminde iç kalite nasıl bir gelişim göstermektedir?
  - **AS 1.1.** Mobil uygulamalar evrimleri süresince karmaşıklık, boyut ve iç kalite açılarından nasıl bir gelişim göstermektedir?

    - **AS 1.2.** Lehman'ın artan karmaşıklık, sürekli büyüme ve azalan kalite yasaları mobil uygulamalar için geçerli midir?
- **AS 2.** Mobil uygulamaların evriminde dış kalite nasıl bir gelişim göstermektedir?
- **AS 3.** İç kalite ve dış kalite arasında anlamlı bir ilişki kurulabilir mi?

1.2. Çözüm Önerisi

Yukarıda bahsedildiği gibi, mobil uygulamaların sayısının ve popülerliğinin artmasıyla yazılım evrimi boyunca yazılım kalitesi daha önemli bir konu haline gelmiştir ve kalite çeşitli açılardan değerlendirilebilmektedir. Bu tez çalışmasında, yazılım ürün kalitesi için en güncel standart olan ISO 25010 [12] rehber alınarak, mobil uygulamaların evriminde kalite, iç ve dış olmak üzere iki açıdan incelenmiş ve bu iki perspektif arasındaki ilişki araştırılmıştır. Bu yaklaşıma göre seçilen açık kaynak kodlu mobil uygulamalar, kod tabanlı ve toplum tabanlı olmak üzere, iki farklı keşifsel çalışma ile değerlendirilmiş ve sonuçlar analiz edilmiştir.

İç kaliteyi değerlendirmek amacıyla yürütülen kod tabanlı değerlendirme aşamasında, ISO 25010 standardını temel alarak ve Lehman'ın [15, 16] sekiz yazılım geliştirme yasasından kalite ile ilişkili olan ikinci, altıncı ve yedinci yasası kapsamında, AS1'in yanıtlanması için hipotezler oluşturulmuş ve istatistiksel yöntemler [20, 21] ile test edilmiştir.

- **Yasa 1.** Artan karmaşıklık (Lehman 2): Lehman'ın yazılım karmaşıklığı ile ilgili ikinci yasasına göre "Bir program geliştikçe karmaşıklığı, onu korumak veya azaltmak için proaktif bir çalışma yapılmadıkça artar".
- **Yasa 2.** Sürekli Büyüme (Lehman 6): Lehman'ın altıncı yasasına göre, "Yazılım sisteminin boyutu, yaşam döngüsü boyunca kullanıcı memnuniyetini devam ettirmek için artmaya devam etmelidir".
- **Yasa 3.** Azalan kalite (Lehman 7): Lehman'ın yedinci yasasına göre, "Yazılımın iyileştirilmesi için sıkı bakım önlemleri alınmadığı ve uygun yöntemler standartlaştırılmadığı sürece, sistemler geliştikçe yazılım sisteminin kalitesi düşer".

Dış kaliteyi değerlendirmek amacıyla yürütülen toplum tabanlı değerlendirme aşamasında ise DeLone ve McLean'in [19], açık kaynak kodlu yazılımların başarısını ölçmek için önerdikleri en çok bilinen modeli temel alınarak; öncelikle

bilgi ihtiyaçları belirlenmiş, daha sonra bunları ölçebilmek için uygulama mağazalarındaki veri tabanlarından metrikler çıkartılmış ve metrik verileri üzerinde derinlemesine analizler gerçekleştirilmiştir. AS2'nin yanıtlanması için hipotezler oluşturulmuş ve istatistiksel analizler [20]ile bunların geçerliliği sınanmıştır.

Son aşamada ise AS3'ü yanıtlamak amacıyla iç ve dış kalite bulguları arasındaki ilişki ve anlamlılığı istatistiksel yöntemler [20] kullanılarak analiz edilmiştir.

1.3. Tez Organizasyonu

Bu tez çalışmasının devamı şu şekilde düzenlenmiştir: İkinci bölümde açık kaynak yazılımların (AKY) üzerine genel bir bilgi, kullanımları ve tercih edilme sebepleri, mobil yazılımlar hakkında genel bilgi ve mobil uygulamaların kullanımına yer verilirken sonrasında açık kaynak mobil yazılımlar, yazılım evrimi hakkında genel bilgi, Lehman yasaları, Lehman'ın artan karmaşıklık, sürekli büyüme ve azalan kalite yasalarına yer verilmiştir. İkinci bölümün devamı olarak yazılım kalite modellerinin detaylı açıklaması yapılırken nesneye dayalı tasarım metriklerinin, DeLone ve McLean başarı modeli ve ilişkili toplum tabanlı metriklerin detaylı açıklamasına yer verilmiştir. Üçüncü bölümde mobil uygulamaların evriminde iç ve dış kalite gelişimi üzerine yapılan ilişkili çalışmalar anlatılmış ve tez çalışmasının bu çalışmalarla benzerlik ve farklılıkları tartışılmıştır. Dördüncü bölümde tez çalışmasında başvurulan yönteme genel bir bakış ile yer verilmiş; araştırma hedefinin belirlenmesi, uygulama seçimi, çalışma tasarımı, test edilecek hipotezlerin oluşturulması, metrik verilerinin toplanması ve analizi kısımları anlatılmıştır. Beşinci bölümde değerlendirme ve analiz sonuçları, altıncı bölümde sonuçlara dair genel bir özet verilmiş ve geçerliliği tehdit eden faktörler ile gelecekte yapılması planlanan çalışmalardan bahsedilmiştir.

2. ÖN BİLGİ

2.1. Açık Kaynak Yazılımlar

2.1.1. Açık Kaynak Yazılımlar Hakkında Genel Bilgi

Çeşitli amaçlarla kullanıcılara yazılımı çalıştırma, ihtiyaçları doğrultusunda değiştirme ve yazılımın temel ya da değiştirilmiş versiyonunu dağıtma özgürlüğü sağlayan lisanslara sahip yazılımlar, “açık kaynak kodlu yazılım” olarak adlandırılır [22].

1960'larda Massachusetts Teknoloji Enstitüsü, Stanford, Kaliforniya (Berkeley) ve Carnigie Mellon gibi önde gelen araştırma üniversitelerinin bilgisayar ve yapay zekâ laboratuvarında AKY’nın kavramsal temelleri atılırken o dönemde, Ken Thompson ve Dennis Ritchie tarafından çok kullanıcılı ve çok görevli yapıya sahip olan UNIX işletim sistemi geliştirilmiştir. AKY’nın kurumsallaşma hareketi ise 1985 yılında Richard Stallmann [23] tarafından GNU manifestosu [24] ile başlatılmış ve 1989 yılında Genel Kamu Lisansı (İng. General Public Licence - GPL) geliştirilmiştir. Böylece açık kaynak yazılım hareketinin çerçevesi çizilmiş ve zamanla bu hareket sağlam ilkelere kavuşmuştur. 1990’lı yıllarda Linux işletim sisteminin de geliştirilmesiyle birlikte, açık kaynak girişimine olan destek iyice artmıştır. Son olarak, açık kaynak yazılımların gelişimi icin radikal bir karar alan Netscape, Microsoft firmasının web tarayıcısı olan Internet Explorer karşısında büyük ölçüde pazar kaybı yaşadığından 1998 yılında web tarayıcısının kaynak kodlarını kamuya açmıştır [25]. Tüm bu gelişmeler, AKY için önemli birer adım olmuştur.

1998 yılının başlarında, açık kaynaklı yazılım, yazılım kullanıcıları ve uygulayıcıları arasında en çok tartışılan konulardan biri haline gelmiştir [26]. Haziran 2010 verilerine göre Linux ve Apache gibi ürünlerin kendi pazarlarında (işletim sistemleri ve http sunucuları) artan paylar kazanma başarısı; yazılım endüstrisinde Microsoft‘a karşı duyulan güvensizlik; geleneksel yazılım geliştirme yaklaşımlarının, etkili ve güvenilir yazılım uygulamaları için artan talebe, tatmin edici bir cevap verememesi gibi sebepler, açık kaynak yazılıma olan ilginin artmasına neden olmuştur [22]. Açık kaynağa ilgi farklı seviyelerde ve farklı bağlamlarda görülebilir:

- Açık kaynağı destekleyen ve organize eden çok sayıda bireysel kullanıcı topluluğu vardır. Bu olgu, özellikle endüstriyel ve akademik araştırmalarda mevcuttur. Öyle ki, AKY'ların birçok ticari ürünün maliyeti, karmaşıklığı ve kısıtlamaları konusunda kullanıcıların artan rahatsızlıklarının cevabı olarak ortaya çıktığı söylenebilir.
- Birçok şirket dikkatini ve gayretini açık kaynak yazılımlara odaklamaktadır. Buna, açık kaynakları, Microsoft'un tekelini zayıflatmak ve açık bir operasyonel platformun kurulmasını zorunlu kılmak için stratejik bir fırsat olarak değerlendiren Sun ve IBM gibi önemli bilgisayar üreticileri örnek verilebilir.
- Son olarak, özellikle Avrupa'daki kamu kurumları ve devlet kurumları, iki ana nedenden ötürü AKY'larla giderek daha fazla ilgilenmektedir. İlk olarak, açık kaynaklı yazılım, ABD'deki teknolojinin baskınlığını dengelemek ve Avrupa'da daha güçlü bir yazılım endüstrisinin gelişimini desteklemek için uygun bir strateji olarak değerlendirilmektedir. İkincisi, hükümetlerin ve kamu idarelerinin yazılım sistemlerine artmakta olan güvenleri, bu yazılımların güvenliği ve güvenilirliği konularında bazı endişeler doğurmuştur. Kamu yönetimi ve hükümetler, belirli yazılım sağlayıcılarına olan bağımlılıkları konusunda endişe duymakta ve dolayısıyla bağımsızlıklarını arttırmalarına yardımcı olabilecek yaklaşımları desteklemektedir. Bu bağlamda, açık kaynak taraftarları, kaynak kodun kısıtlanmadan kullanılabilirliğinin, bu sorunları etkin bir şekilde ele almayı sağladığını savunmaktadır.

2.1.2. Açık Kaynak Yazılımların Tercih Edilme Sebepleri

AKY'lar başta kamu kurumları olmak üzere maliyeti düşürme, lisanslama giderlerini azaltma, belirli bir firma ya da grubun tekelinden çıkarma, güvenli ve daha kaliteli yazılım sunma gibi faydalardan ötürü sıklıkla tercih edilmektedir. AKY kullanıcılarının geniş bir yelpazede olması dolayısıyla bu yazılımların tercih edilme faktörleri de çeşitlilik göstermektedir. Aşağıda bu faktörlerin detaylı açıklamasına yer verilmiştir:

Özgürlük - Özgür açık kaynak yazılım (Free Open Source Software - FOSS) ile kullanıcı yazılımı kontrol edebilir ve yazılımı kendi gereksinimlerine göre değiştirebilir. Açık kaynak, dünya çapında bir geliştirici ve kullanıcı topluluğu ortaya

çıkarır, böylelikle dünyanın farklı yerlerindeki insanlar yazılımla ilgili tüm soruları tartışabilir. Kullanıcı sadece açık kaynaklı yazılımlara erişim sağlamakla kalmaz, aynı zamanda yazılımı kontrol eder ve değiştirir.

Kalite - Deneysel olarak, açık kaynak yazılımlar, diğer yazılımlara nazaran daha kaliteli bir seyir izlemektedir. Geleneksel yazılımlarda, yazılımı tespit edebilen, tanıyan, triyaj edebilen ve çözebilen tek kişi, yazılımı geliştirenlerdir. Açık kaynakla sadece ücret karşılığında değil, sırf üretme ve geliştirme aşkıyla bile en iyi geliştiriciler yazılım üretimine dâhil edilebilir. Ayrıca, potansiyel olarak yazılıma katkıda bulunabilecek geliştirici sayısı, açık kaynak yazılımda daha fazladır. Bu sebeple, AKY'ların yazılım geliştiricilerin dikkatli incelemesinden geçip ortaya çıkan hataların kısa sürede çözülmesi kaçınılmazdır.

Destek - Şu anda İnternet platformunun %80'i açık kaynaklı yazılıma dayanmaktadır. AKY'lar büyük bir kullanıcı ve geliştirici topluluğuna sahiptir ve bu topluluklar, önerilerini ve önemli gördükleri noktaları paylaşmakta özgürdür. Linux dağıtımlarının çoğu, çevrimiçi bir topluluğa ve Ubuntu, openSUSE, ClearOS vb. forumlara sahiptir.

Güvenlik - Açık kaynaklı yazılımlar kullanıldığında, kullanıcı ya da geliştirici, programın yazarı tarafından gözden kaçırılan veya ihmal edilen bir sorunu tespit edebilir ve düzeltebilir. Orijinal yazarın iznini beklemek yerine, birden fazla programcı programın düzeltilmesini, güncellenmesini ve yükseltilmesini sağlayabilir. AKY'nın kaynak kodlarına farklı kullanıcılar erişebildiği için, kullanıcıların yazılımın arka planında neler olup bittiğinden haberdar olma şansı çok yüksektir. Bu yüzden AKY'lar kullanıcılar tarafından daha güvenli görülüp daha fazla tercih edilmektedir.

Maliyet – AKY'ların, hiçbir bedelinin olmadığı yönünde yanlış bir kanaat mevcuttur. Bu inanç ancak belli bir oranda doğrudur. Çoğu AKY için herhangi bir lisanslama bedeli söz konusu değildir. Birçok AKY İnternet üzerinden herhangi bir ücret ödenmeksizin indirilebilir. Lisanslama bedelleri üzerinden bir değerlendirilme yapıldığında AKY'lar özel-mülk yazılımlara göre ucuzdur [27]. Ancak, lisanslama bedeli, bir yazılım projesinin maliyetini oluşturan yegâne kalem değildir. Proje maliyetinin hesaplanmasında personel harcamaları, donanım gereksinimleri, eğitim giderleri ve fırsat maliyetleri de göz önüne alınmalıdır. Genel olarak, bu

şekilde yapılan bir maliyet analizi, Toplam Sahip Olma Maliyeti (Total Cost of Ownership – TCO) olarak adlandırılır ve AKY'ların sağladıkları tasarrufu doğru değerlendirmede daha faydalı olmaktadır [27]. Toplam sahip olma maliyetleri açısından bir değerlendirmeye girildiğinde, AKY'ların maliyetsiz oldukları ileri sürülemez. Ancak, yapılan araştırmalar AKY'ların özel-mülk yazılımlara göre lisanslama, bakım ve destek, güncelleme maliyetleri açısından tasarruf sağladıklarını ortaya koymaktadır.

2.2. Mobil Yazılımlar

2.2.1. Mobil Yazılımlar Hakkında Genel Bilgi

Mobil uygulama akıllı telefonlar, tablet bilgisayarlar ve diğer mobil cihazlarda çalışmak üzere tasarlanmış bir yazılımdır [28]. Mobil uygulamaların geçmişi, sesli aramalar göndermek ve almak için mikroçiplerin en basit yazılımlara gerek duyduğu cep telefonları ile başlamıştır. İlk halka açık hücresel cep telefonu çağrısı, 1,1 kg ağırlığında ve 23 x 13 x 4,45 cm boyutlarındaki bir cihazla 3 Nisan 1973'te, Motorola'dan Martin Cooper tarafından New York'ta bir tanıtım gösterisinde yapılmıştır. Akıllı telefonlar için ilk mobil uygulamayı elde etmek amacıyla yapılan araştırma ve geliştirme ise yirmi yıl sürmüş; hesap makinesi, dünya saati, takvim ve iletişim defteri gibi özelliklere sahip ilk akıllı telefon, 1993 yılında IBM tarafından duyurulmuş ve tüm kazanç, akıllı telefonlara yönelik ilk mobil uygulamayı dünyaya tanıtan IBM Simon'a [29] gitmiştir. 2002'de piyasaya sürülen BlackBerry Smartphone, mobil uygulama geliştirme alanında bir sonraki büyük başarı olup daha önce Research In Motion Limited olarak bilinen ve yenilikçi kablosuz e-posta konsepti ile entegre olan BlackBerry Limited tarafından duyurulmuştur.

En büyük üç mobil uygulama mağazası Android için Google Play, iOS için App Store, Windows 10, Windows 10 Mobile ve Xbox One için Microsoft'tur.

Google Play (eskiden Android Market olarak bilinirdi), Android cihazlar için Google tarafından geliştirilen, uluslararası ve çevrimiçi bir yazılım mağazasıdır. Ekim 2008'de açılan uygulama mağazası, Mayıs 2016 itibariyle mağazasından toplam 65 milyar uygulama indirildiğini duyurmuştur [30]. Google Play Store'daki mevcut uygulamaların sayısı Temmuz 2013'te 1 milyon uygulamayı aştıktan sonra, Aralık 2017 itibariyle 3,5 milyon uygulamaya ulaşmıştır [31]. Mağazanın 2015 yılında 6 milyar ABD doları gelir elde ettiği raporlanmıştır [32].

App Store, IOS için 10 Haziran 2008 tarihinde geliştirilen bir uygulama mağazasıdır. Haziran 2017 itibariyle Apple, App Store'dan 180 milyar uygulamanın indirildiğini duyurmuştur [33]. App Store, 2,2 milyon kullanılabilir uygulama ile Google Play'den sonra ikinci en büyük uygulama mağazasıdır.

Microsoft Store (eski adı Windows Store), Microsoft tarafından Windows 8 ve Windows RT (Runtime) platformları için 2012 yılında tanıtılmıştır. Öncelikle, tabletlerde ve diğer dokunmatik tabanlı cihazlarda kullanılmak üzere inşa edilmiş olan Windows Store uygulamalarını dağıtmak için kullanılır. Bununla birlikte, klavye ve fare ile masaüstü bilgisayarlarda ve dizüstü bilgisayarlarda da kullanılabilir [34, 35].

Amazon Appstore, Android işletim sistemi için alternatif diğer bir uygulama mağazasıdır. Mart 2011'de açılmıştır ve 2017 itibarıyla uygulama mağazasının yaklaşık 384.537 uygulaması vardır [36].

2.2.2. Mobil Uygulamaların Kullanımı

Bugün çoğu insan akıllı telefonlara sahiptir, bu nedenle uygulamalara erişmek kolaydır ve bu uygulamalar hayatımızı daha iyi hale getirmektedir. An itibarıyla sosyal ağlar, seyahat, sağlık, bankacılık, fitness, takvimler, oyunlar, haberler ve daha fazlası için milyonlarca mobil uygulama mevcut durumdadır. Mobil uygulama pazarı, 2017 yılı itibarıyla Apple'ın 2,2 milyon uygulamaya sahip App Store'unun yanı sıra Google'ın 3,5 milyon uygulamaya sahip Android Market'i ile de büyük bir patlama yaşamıştır. 5 milyara hızla yaklaşan toplam küresel mobil kullanıcı sayısıyla dünya nüfusunun neredeyse dörtte üçü, artık bir cep telefonu kullanmaktadır ve Ericsson'un yayınladığı rapora göre [37], cep telefonları artık dünyadaki çoğu insan için gündelik hayatın vazgeçilmez bir parçası haline gelmiştir. En yeni veriler [37], dünya nüfusunun yarısından fazlasının bir akıllı telefon kullandığını ve bunun, bu yılın raporunda bir başka önemli kilometre taşı olduğunu göstermektedir. Tüm dijital medyalara ayrılan zamanın %52'den fazlasını mobil uygulamalar oluşturmakta ve kullanıcılar mobil uygulamaları kullanmayı web sitesi gibi diğer dijital araçlardan çok daha fazla tercih etmektedir. Kullanıcılar tarafından en çok tercih edilen 500 şirketin satışlarının yüzde 42'si mobil uygulamalar üzerinden gerçekleşmekte ve iyi bir kullanıcı deneyimi sunan

mobil uygulamalar, bu satış oranlarında oldukça etkili olmaktadır. Ayrıca, mobil uygulama gelirlerinin 2020 yılında 189 milyar doları bulması beklenmektedir [38].

2.3. Açık Kaynak Mobil Yazılımlar

Kasım 2007'de arama motoru devi Google, Open Handset Alliance olarak bilinen bir grup mobil üreticiyle birlikte Android olarak bilinen Linux tabanlı bir akıllı telefon işletim sistemi üzerinde çalışacaklarını açıklamıştır. 2007 yılında Google, Android işletim sistemini geliştirdiğini açıkladıktan kısa süre sonra kurulan ve adı Open Handset Alliance (OHA) olan dev birliğin amacı, piyasadaki standartları belirlemekti [39]. Google önderliğindeki bu birlikte HTC, Samsung, LG, Motorola, Sony Ericsson gibi telefon üreticilerinin yanı sıra Intel ve Qualcomm gibi işlemci üreticileri, Nvidia gibi grafik yongası üreticileri yer alırken birliğe birçok operatör de destek vermiştir. OHA, neredeyse tüm bilişim sektörünü kapsayacak şekilde yeni standartlar getirmiş ve bu birlik kurulduğu sırada Linux 2.6 çekirdeğini kullanan ilk Android işletim sistemi sürümü de piyasaya sunulmuştur [40]. 2008 yılında Google, bu işletim sisteminin kaynak kodlarını açık hale getirmiş ve telefon üreticileri ilk Android telefonu üretmek için yarışa girmişlerdir.

İlk Android telefon olan HTC Dream, Android 1.0 kullanmaktaydı ve İnternet tarayıcısı, medya oynatıcısı, Gmail senkronizasyonu özelliklerine sahip olup Android Market uygulaması da mevcuttu [41]. Haziran 2008'de ise Nokia, yeni ve ortak bir işletim sisteminin meydana getirilmesi amacıyla kurulmuş olan topluluk şirketi Symbian'ı tamamen satın aldığını ve yazılımı, Symbian'ın gelişimini açık kaynaklı bir işletim sistemi olarak takip eden ve kar amacı gütmeyen bir altyapıya dönüştürdüğünü açıklamıştır. Bunun sonucu olarak Nokia ve Apple gibi büyük telefon üreticileri Symbian ve iOS gibi işletim sistemleri geliştirmişlerdir; ancak bu işletim sistemlerini yalnızca kendi telefonlarında kullanmışlar ve piyasada bir standart yakalayamamışlardır. Linux Mobile Foundation (LiMo), cep telefonu için Linux dağıtımı yapmaya adanmış telefon üreticilerinin ve ağ sağlayıcılarının bir üyesi olarak, iPhone'un duyurulduğu ay olan Ocak 2007'de yeniden oluşturulmuştur. Ancak zaman geçtikçe bu girişimlerin çoğu sekteye uğramıştır. Örneğin Symbian, Android ve iOS ile rekabet edebilmek için etkileyici bir kullanıcı ara yüzünü yeniden tasarlamayı başaramamış ve Nokia, 2011'in başlarında Microsoft ile anlaşarak bu yazılımı geliştiremeyeceğini belirtmiştir.

Sonuç olarak, Android işletim sistemi [41] başta olmak üzere bazı AKY işletim sistemleri ile çok sayıda mobil AKY uygulaması geliştirilmektedir. Android işletim sisteminin mobil cihazlarda 2010 yılında %17 olan pazar payının 2011 yılında yaklaşık %43'e yükseldiği ifade edilmiştir [42]. Bu işletim sisteminin kısa sürede bu denli büyük bir pazar payına sahip olmasında hem kaynak kodunun açık oluşu hem de küresel ölçekli bir bilişim firması olan Google tarafından yürütülüyor oluşu büyük rol oynamaktadır. Endüstrinin, özgür ve açık kaynak kodlu yazılımın ne yaptığı ve yapamadığı konusunda olgun bir anlayışa yaklaşması iyi olabilir. Sektörde herkese açık bu sistemlerin milyonlarca kullanıcısı olduğundan, sektör günümüzde oldukça kârlı bir yazılım işi haline gelmiştir.

2.4. Yazılım Evrimi

2.4.1. Yazılım Evrimi Hakkında Genel Bilgi

Evrim, "bir varlığın öz niteliklerinde veya kurucu unsurlarının bir veya daha fazlasında devam eden değişim sürecini" yansıtır [43]. Bu tanıma göre, bir yazılım sisteminin zaman içindeki değişimlere kolayca uyum sağlaması, "yazılım evrimi" olarak adlandırılır. Yazılım değişikliği desteklemiyorsa, yavaş yavaş işe yaramaz hale gelir [15]. Yazılım evrimi, paydaşların iş hedeflerini karşılayabilmek için, yazılım sistemlerine yeni gereksinimleri dâhil etmeyi sağlar. Tüm yazılım geliştirme süreçlerine göre yazılımın, çevreye uyum veya kullanıcı memnuniyetinin sürdürülmesi ihtiyacına cevap olarak gelişmesi gerekmektedir [44].

AKY sistemlerinin evriminin incelenmesi ilginçtir, çünkü bu tür sistemler, sıkı geliştirme süreçleri olmadan ve bağımsız bireylerin veya ekiplerin katılımıyla, uzun süreler gelişmeye devam etmektedir. Yazılım evrimi özellikle, yazılım sistemlerinin zaman içinde nasıl geliştiği ile ilgilidir [45]. Bu sistemler, tekrarlanan değişikliklerden sonra evrimleşmekte ve bu da karmaşıklığın artmasına neden olmaktadır. Sistemler değişikliklere kolayca uyum sağlayacak şekilde tasarlanmamışsa karmaşıklık, büyük değişiklik maliyetlerine de yol açabilir. Bu nedenle, yazılım evrimindeki temel zorluklardan biri, yazılımın, paydaşların değişen gereksinimlerini karşılamak için zaman içinde gelişebilme yeteneğidir [46]. AKY sistemlerinin evrimini analiz etmek için araştırmacılar, gelişmekte olan çok sayıda yazılım sisteminin kaynak kodlarına, sürümlerin ve değişiklik kayıtlarının geçmişine erişmişlerdir. AKY projelerinin farklı yönleri üzerine kolayca erişilebilen veriler, araştırmacılara yazılım evrimi ile ilgili önceki çalışmaları doğrulamak ve

AKY evrim problemini adreslemek için çok sayıda fırsat sunmaktadır [15, 16, 47-52].

Yazılım evriminin en önemli çalışmaları M.M. Lehman ve arkadaşları tarafından 1970'lerin ortalarında başlatılan ve günümüze kadar uzanan 40 yıllık bir sürece yayılmaktadır. Bu çalışmalar uzun yıllar çalışmalara konu olan sekiz yazılım geliştirme yasasının ortaya çıkmasını sağlamıştır [15, 16, 47-51]. Bu yasalar, Lehman ve arkadaşları tarafından tanımlanan yazılım sürümleri, sistemler ve E-Tipi uygulamaların (örneğin, gerçek dünya, iş süreçleri ve kişiler ile iç içe geçmiş sistemler- Oyun geliştirme sistemi) gelişimi ile ilgili gözlemleri sürekli olarak raporlamayı amaçlamaktadır. Yasalar ve dayandığı teoriler, uygun bir şekilde yeniden formüle edilebilir ve geçerlenebilir ya da reddedilebilir [53, 54]. Bu sebeple Lehman yasaları, yazılım evrimini bütünsel olarak anlamada önemli ilerlemeler sağlamış olup günümüzde de pek çok çalışmaya konu olmaktadır [17, 55-60].

2.4.2. Lehman Yasaları

Lehman'ın yazılım evrimi üzerine oluşturduğu yasalar Çizelge 1'de verilmiştir:

Çizelge 1. Lehman Yasaları ve Tanımları

Yasa	Tanımı
1-Sürekli değişim (1974)	Bir yazılımı uzun süre kullanabilmek için, kullanıcı ve çevre ihtiyaçlarına göre yazılımın sürekli değişmesi gerekir.
2-Artan Karmaşıklık (1974)	Bir yazılım sistemi geliştikçe karmaşıklığı, onu korumak veya azaltmak için proaktif bir çalışma yapılmadıkça artar.
3-Özdenetim (1974)	Bu kanuna göre, yazılım evrimi kendi kendini düzenleyen bir süreçtir; bu da yazılımın yaşamı boyunca boyutunu ayarlayacağı anlamına gelir.
4-Durağanlığın korunması (1980)	Bu yasa, bir yazılım projesindeki ortalama etkinlik (Bir yazılımın sürümleri boyunca bir ya da daha fazla noktadaki ortalama ve varyansın değişim oranı - change rate) oranının istatistiksel olarak değişmez olmasını şart koşmaktadır.
5-Benzerliklerin korunması (1980)	Bu yasa, ardışık sürümler arasındaki değişikliklerin sınırlı olduğunu göstermektedir.
6-Sürekli büyüme (1980)	Bir yazılımın boyutu, yaşam döngüsü boyunca kullanıcı memnuniyetini devam ettirmek için artmaya devam etmelidir.
7-Azalan kalite (1996)	Bir yazılımın iyileştirilmesi için sıkı bakım önlemleri alınmadığı ve uygun yöntemler standartlaştırılmadığı sürece, sistemler geliştikçe yazılımın kalitesi düşer.
8-Geribildirim sistemi (1996)	Bu kanuna göre, evrim süreci çok katmanlı, çok döngülü ve çok yönlü geri bildirim sistemleri tarafından oluşturulmuş olmalı ve yazılım, sağlam bir temel üzerinde önemli iyileşmeler elde edecek şekilde muamele edilmelidir.

Bu tez çalışması kapsamında analiz edilecek olan ikinci, altıncı ve yedinci Lehman yasalarına ait detaylar, izleyen paragraflarda anlatılmıştır:

Artan Karmaşıklık Yasası

Lehman'ın yazılım karmaşıklığı ile ilgili ikinci yasası "Bir program geliştikçe, karmaşıklığı, onu korumak veya azaltmak için proaktif bir çalışma yapılmadıkça artar" şeklindedir. Yazılımlar gereksinimlerin artmasına bağlı olarak hızlı biçimde çok büyük boyutlara ulaşmaktadır**.** Yazılımın hızlıca büyümesiyle beraber karmaşıklığının da artması şaşırtıcı değildir. Karmaşıklık analizi için, Lehman ilk olarak modül sayısının yüzdesini kullanmıştır [14]. Lehman, yazılımın karmaşıklığını ve bağımlılığını ölçmede sınıfın ağırlıklı metot sayısı (İng. Weighted Method per Class-WMC), nesne sınıfları arasındaki bağımlılık (İng. Coupling between Object Classes-CBO) ve sınıfın tetiklediği metot sayısı (İng. Response for a Class-RFC) metriklerinin etkinliğini kanıtlarken Li ve arkadaşları [17], WMC metriğini karmaşıklığı ölçmede önermiştir. Kaur ve arkadaşları [61] karmaşıklığı ölçmede Chidamber ve Kemerer (C&K) metrik setini kullanırken Li ve arkadaşları [17] ve Xie ve arkadaşları çalışmalarında [18] karmaşıklığı ölçmek için ortalama McCabe döngüsel karmaşıklığının (İng. McCabe Complexity-MCC) kullanılmasını önermiştir. Ayrıca, Newhook ve arkadaşları [62] yazılımların evriminde karmaşıklığı ölçmek için, kod satır sayısını (İng. Lines of Code-LOC) önermiştir.

Sürekli Büyüme Yasası

Lehman'ın altıncı kuralına göre "Bir yazılımın boyutu, yaşam döngüsü boyunca kullanıcı memnuniyetini devam ettirmek için artmaya devam etmelidir". Performans iyileştirme ve yeni platformlara uyum sağlama gerekliliği nedeniyle yazılım gereksinimleri artmaktadır, buna bağlı olarak da yazılımın işlevi ve içeriği sürekli olarak artmakta ve sonuçta sistemin boyutu artmaktadır. Daha önceki çalışmalarda [14, 18, 63, 64] araştırmacılar, sistem boyutu için çeşitli metrikler kullanmışlarsa da çoğunlukla boyut ve fonksiyonellik metriklerini tercih etmişlerdir. Lehman [14], yazılım boyutunu ölçmek için sistemin modüllerinin sayısını kullanırken, Godfrey ve Gonzalez [65], boyut metrikleri olarak LOC'u, Israeli [66] ise Linux çekirdeğinin büyümesini tanımlamak için sistem çağrılarının sayısını, fonksiyon sayısını ve LOC'u kullanmıştır. Kaur ve arkadaşları [61], sürekli büyüme yasasını boyut ve fonksiyonellik perspektifinden analiz etmiştir: Boyut metriği olarak LOC'u, fonksiyon metriği olarak da fonksiyon sayısı ve sınıf sayısı metriklerini kullanmışlardır. Nesneye yönelik programlamada yazılım geliştikçe,

uygulamanın gereksinimlerinin artmasına bağlı olarak programın boyutunun ve sınıf sayısının artması beklenir [17].

Azalan Kalite Yasası

Lehman'ın yedinci yasasına göre, "Bir yazılımın iyileştirilmesi için sıkı bakım önlemleri alınmadığı ve uygun yöntemler standartlaştırılmadığı sürece, sistemler geliştikçe yazılımın kalitesi düşmektedir". Önceki araştırmacılar [14, 18, 64] Lehman'ın azalan kalite yasasını çeşitli perspektiflerden açıklarken Lehman, çalışmasında [49] belirli metrikler vermeyerek bu yasayı sadece "gömülü sistem, zamanın geçişi ile giderek daha uygunsuz hale gelmektedir" hipotezine dayanarak yorumlamıştır. Xie ve arkadaşları [18], yazılımın kalitesini değerlendirmek için hata yoğunluğunu (toplam hata/toplam satır sayısı) kullanmayı önermişse de bu yöntemin, bazen hatalar rastgele keşfedildiğinden yazılımın kalitesini ölçmede çok doğru bir araç olmayacağı düşünülmüştür. Xie ve arkadaşları [18] ayrıca, yazılım sisteminin kalitesinin iç ve dış kalite açılarından değerlendirilmesi gerektiğini belirtmiştir. Bu tez çalışmasında da iç kalite ve dış kalite değerlendirmesi yer almaktadır. Mobil uygulamalar genelde yaygın olarak kullanıldığından dış kalite, kullanıcının yazılımı kabul etmesi ve memnuniyeti anlamına gelir.

Gyimothy ve arkadaşlarının [63] yazılım kalitesini ölçmek için önerdiği metotlardaki uyum eksikliği (İng. Lack of Cohesion in Methods-LCOM) ve kararsızlık (İng. Instability) kalite özellikleri bu tez çalışmasında kod tabanlı değerlendirmede kullanılmıştır. Tez çalışması kapsamında ayrıca, kullanım kolaylığı (İng. Usability) ve taşınabilirlik (İng. Portability) özellikleri nedeniyle popüler bir hal alan mobil uygulamaların, taşınabilirlik ve kullanım kalite sınıflarının alt özelliği olan anlaşılabilirlik (İng. Understandability) kalite sınıfı da incelenmiştir. Bir bileşenin taşınabilirliği bağımsızlığına, yani bileşenin dış destek olmadan işlevselliğini gerçekleştirme kabiliyetine bağlıdır. Anlaşılabilirlik ise programın bir kişi tarafından ne kadar anlaşılabildiği ile alakalıdır ve özellikle yeniden kullanılabilirliğin önemli olduğu uygulamalar için kritik bir özelliktir. Bu bağlamda, kaliteli bir yazılım kolay anlaşılabilir ve kolay taşınabilir olmalıdır.

2.5. Yazılım Kalite Modelleri

Yazılımın kalitesi, bir ürünün özelliklerinin toplamı veya arzulanan ihtiyaçları karşılama becerisi ile alakalıdır. Yazılım kalitesinin artırılmasına yönelik

araştırmalar, kullanıcıların artan kalitede yazılım ürünleri talep etmesinden doğmaktadır. Kalitenin arttırılmasına yönelik faaliyetler bir mühendislik disiplini çerçevesinde ele alınmalıdır [67]. IEEE Yazılım Mühendisliği Terminoloji Sözlüğü'ne göre [68] yazılım ürünlerinin kalitesi 1) bir sistemin, bileşenin veya işlemin, belirtilen gereksinimleri karşılaması ve 2) bir sistemin, kullanıcıların gereksinimlerini veya beklentilerini karşılama derecesi olarak tanımlanabilir.

Yazılım kalite modeli, yazılımın kalitesini tanımlayan özellikleri temsil eden bir araçtır. Kalite modelleri, hem kullanımlarının daha kolay olması ve hem de kalite yönetiminde daha çok kalite fikrini yakalamaya imkân tanıması dolayısıyla faydalıdır. Yazılım kalitesini tanımlamak, değerlendirmek ve korumak için farklı kalite modelleri oluşmuştur ve bu bağlamda yüksek kaliteli bir yazılım, bir dizi kalite faktörüyle ilişkilidir. Zaman çizelgesindeki sıralarına göre temel yazılım kalite modelleri şunlardır: McCall Kalite Modeli (1977) , Boehm Kalite Modeli (1978), FURPS Kalite Modeli (1992), Dromey Kalite Modeli (1995), ISO/IEC 9126 Kalite Modeli (2001) ve ISO 25010 Kalite Modeli (2008).

Bu tez çalışması için en güncel model olan ISO 25010 kullanılmıştır. Bu sebeple ISO 25010'a temel olduğu için ISO 9126 ile ISO 25010 kalite modelleri aşağıdaki paragraflarda detaylı olarak anlatılmıştır.

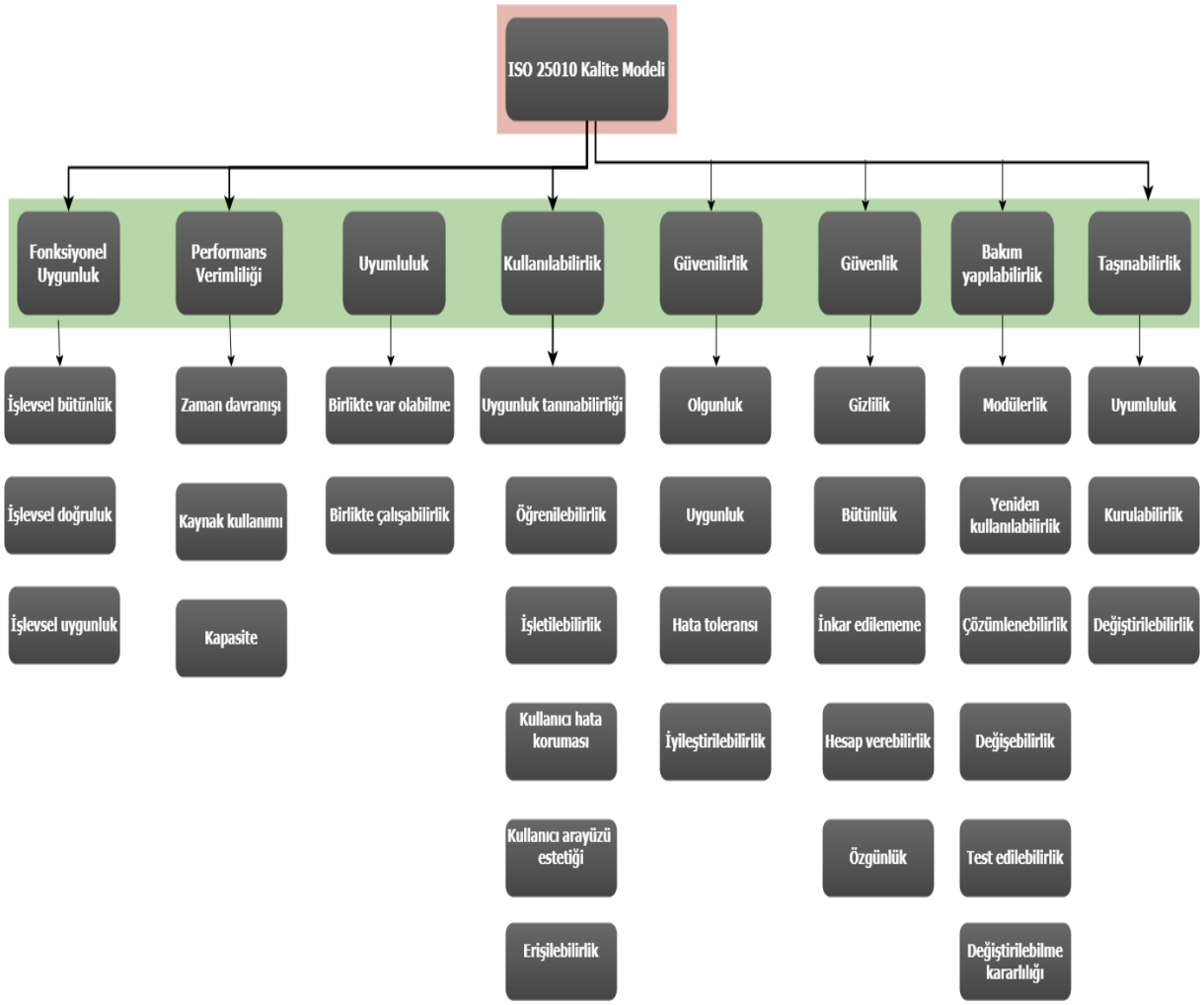
ISO/IEC 9126 Kalite Modeli (2001), ISO tarafından yazılım ürünü değerlendirmesi için, kalite özellikleriyle ilgili terminoloji konusunda, ilk uluslararası görüş birliği olarak yayınlanmıştır [69]. Bu model aslında McCall ve Boehm kalite modellerine dayanmaktadır. Modelin iki ana bölümü vardır: 1) İç ve dış kalite özellikleri, 2) Kullanımdaki kalite özellikleri.

İç kalite özellikleri, yazılım çalıştırılmadan değerlendirilebilecek özelliklere yönelikken dış kalite özellikleri, yazılımı yürütme sırasında gözlemlenerek değerlendirilebilecek özelliklere atıfta bulunur. Bu dış kalite özellikleri, yazılım çalışırken ve ayrıca bakım sırasında, kullanıcılar tarafından deneyimlenir. Kullanımdaki kalite özellikleri ise ürünün etkinliği, üretkenlik, sunulan güvenlik ve kullanıcıların memnuniyeti ile ilgilidir. Altı temel kalite özelliği ve bunlarla ilişkili 27 alt özellikten oluşan ISO 9126 kalite modeli, Şekil 1'de verilmiştir.

Şekil 1. ISO 9126 Kalite Modeli [69]

ISO 25010 Kalite Modeli (2008), ISO 9126 modelinin güncellenmesiyle ortaya çıkmıştır [70]. Yazılım kalitesi, müşteri gözünden kalite ve yazılım üreticisi gözünden kalite olmak üzere iki temel bakış açısı ile ele alınabilir. Müşteriler genel olarak satın aldıkları yazılımların kolay kullanılabilir, hatasız, tüm isteklerini karşılayan ve yeterli performansa sahip olmasını ister. Buna karşılık yazılım üreticileri ise geliştirme ve bakım maliyetlerinin düşük ve üretilen yazılımın parçalarının bir sonraki projelerinde de kullanılabilir olmasını ister. ISO 25010, yazılım ürünlerinin kalitesini anlatan ve sınıflandıran uluslararası bir standarttır [70] ve bu standartta geliştirici gözünden yazılımın iç kalitesi, “belirlenen ve hedeflenen gereksinimleri karşılama kabiliyetine sahip bir yazılımın özelliklerinin bütünü” olarak tanımlanmıştır. Yazılımın iç kalitesi, kötü tasarım ve uygulama seçeneklerinin yazılım içine istemsizce eklenmesi nedeniyle yazılımın evrimi boyunca gerileyebilir. "Memnuniyet" terimini kullanan ISO 25010’a göre dış kalite ise "yazılımın belirli bir kullanım bağlamında kullanıcıları tatmin edebilme kabiliyeti"

anlamına gelir. Bu model, 8 temel kalite özelliği ve bunlarla ilişkili alt özelliklerden oluşmaktadır. ISO 9126'nın anlaşılabilirlik ve kararsızlık özellikleri ISO 25010'da uygunluk tanınabilirliği ve değiştirilebilirlik olarak isimlendirilmektedir. Temel amaçlarından biri, ISO 9126'ya dayanan bir dizi standart oluşturmak ve kalite şartlarının belirlenmesi ve değerlendirilmesi ile yazılım ürünlerinin geliştirilmesine rehberlik etmek olan ISO 25010 kalite modeli, Şekil 2'de verilmiştir.



Şekil 2. ISO 25010 Kalite Modeli [70]

Bu tez çalışması kapsamında en güncel kalite modeli olan ISO 25010 kalite modeli, mobil uygulamaların evriminde iç kalite gelişiminin analizi için temel alınmıştır. İç kalite gelişimi aşamasında 'Uyumluluk', 'Kararsızlık', 'Taşınabilirlik' ve 'Anlaşılabilirlik' kalite özellikleri analiz edilmek üzere seçilmiştir.

2.6. Nesneye Yönelik Yazılım Metrikleri ve İlişkili Kalite Özellikleri

Metrikler, yazılım süreç ve ürünlerini geliştirmek için anlamlı ve zamanında yönetim bilgisi sağlamak amacıyla yazılım geliştirme sürecine ve ürünlerine ölçüm

bazlı tekniklerin uygulanmasında kullanılırlar [71]. Yazılımın kalitesi, metrikler kullanılarak değerlendirilebilir ve metriklerin kullanımı, yazılım ürünleri, süreçleri ve hizmetleri hakkında daha fazla bilgi edinmeye yardımcı olabilir. Yazılım ürünlerini, süreçlerini ve hizmetlerini belirlenen standartlara ve hedeflere göre değerlendirmek için metrikler kullanılabilir. Metrikler, yazılımı üretmek için kullanılan kaynakları ve süreçleri kontrol etmek için ihtiyaç duyduğumuz bilgileri sağlayabilir. Yazılım ürünlerinin gelecekteki özelliklerini tahmin etmek için de metrikler kullanılabilir [72].

Mobil uygulamalar çoğunlukla Java ya da Objective-C gibi nesneye yönelik (İng. Object oriented-OO) diller kullanılarak geliştirilmektedir. Yazılım kalitesini değerlendirmek ve temin etmek zor olabildiğinden literatürde, özellikle de nesneye yönelik kod ve tasarımın yapısal kalitesini yakalamak ve ölçebilmek için birçok metrik önerilmiştir [73-76]. Kaliteli yazılımın uyumluluğunun (İng. Cohesion) yüksek, karmaşıklık (İng. Complexity) ve bağımlılığının (İng. Coupling) düşük olduğu yaygın olarak kabul edilmektedir [73]. Bu tez çalışmasında, en çok başvurulan metrik seti olan C&K metrik seti iç kalitenin değerlendirilmesi aşamasında kullanılacaktır.

2.6.1. Chidamber & Kemerer Metrik Seti

Chidamber ve Kemerer'in tanımlamasından bu yana [73], yazılım kalitesini değerlendirmek için nesneye yönelik metrikleri popülerlik kazanmış olup kısaca C&K metrik seti olarak isimlendirilmektedir. C&K metrik seti, nesneye yönelik tasarımın farklı özelliklerini değerlendiren altı metrikten oluşmaktadır.

Metrik 1. Alt Sınıf Sayısı (İng. Number of Children - NOC)

Tanım	NOC, hiyerarşideki bir sınıfa bağlı olan alt sınıfların sayısıdır. NOC, bir sınıftan doğrudan türetilmiş alt sınıfların sayısını ölçer.
Amaç	NOC yeniden kullanım düzeyini, uygunsuz soyutlamanın olasılığını ve gerekli test seviyesinin olası belirtimini gösterir. Bir sınıf çok fazla alt sınıfa sahipse bu durum, kalıtımın yanlış kullanıldığının bir göstergesi olabilir. Çok alt sınıfı olan sınıfların metotları daha çok test etmeyi gerektirdiğinden bu metrik, sınıfı test etmek için harcanacak bütçe hakkında bilgi verir.

Kılavuz ve Yorum

Daha fazla sayıda çocuk daha büyük bir yeniden kullanım olasılığını göstermekle birlikte, aşağıdaki dezavantajlara sahiptir:

Yanlış soyutlama olasılığı artar (alt sınıfın yanlış kullanımı).

Tüm çocukların temel sınıfa bağımlılıkları çok fazla olduğunda yapılacak herhangi bir değişikliğin gerçekleştirilmesi tüm çocukları etkileyeceğinden daha zordur.

Daha çok test etmeyi gerektirir.

Bir sınıfın, sistemin tasarımı üzerinde sahip olabileceği potansiyel etkinin bir göstergesidir.

Kalıtım yoluyla yeniden kullanımda tasarım, yüksek bir bağımlılığa sahipse, fonksiyonelliği birkaç sınıfa ayırmak daha iyi olabilir.

Metrik 2. Kalıtım Ağacının Derinliği (İng. Depth of Inheritance Tree - DIT)

Tanım

DIT, bir sınıfın kalıtım hiyerarşisinin maksimum seviyesini ölçer. Hiçbir sınıftan türetilmemiş sınıflar için DIT 0'dır. Eğer çoklu kalıtım varsa DIT, en uzak köke olan uzaklık olarak alınır.

Amaç

DIT sınıf karmaşıklığını, tasarım karmaşıklığını ve yeniden kullanım olasılığını ölçmeyi amaçlar. Bir sınıf ne kadar derin olursa, daha fazla sayıda metodu miras alması muhtemeldir.

Kılavuz ve Yorum

Derin kalıtım ağaçları tasarım karmaşıklığı oluşturarak test edilebilirliği ve yeniden kullanılabilirliği zorlaştırır.

Düşük kalıtım derinlikleri, kalıtım mekanizmasının avantajlarından yararlanamayan fonksiyonel kodu gösterir.

DIT metriği öncelikle, verimliliği ve yeniden kullanılabilirliği değerlendirir; ancak aynı zamanda, anlaşılabilirlik ve test edilebilirlik ile de ilgilidir [73, 75-77].

Metrik 3. Sınıfın Tetiklediği Metot Sayısı (İng. Response for a Class - RFC)

Tanım

RFC, verilen sınıftan bir nesnenin metotları çağrıldığında, bu nesnenin tetikleyebileceği tüm metotların sayısıdır. Bu metrik

özellikle sınıf dışından çağrılan metotları içerdiğinden, aynı zamanda sınıf ve diğer sınıflar arasındaki potansiyel iletişimin bir ölçüsüdür. Bir sınıftaki metot sayısı ile sınıftaki metotlar tarafından doğrudan çağrılan metot sayısının toplamına eşittir.

Amaç — RFC, bir sınıfın karmaşıklığını ve diğer sınıflarla olan iletişim miktarını ölçmeyi amaçlar. Bir sınıf için RFC büyükse, yüksek bir karmaşıklık olduğu anlamına gelir. Örneğin, bir sınıfta bir metot çağrısı, hedef ve diğer sınıflarda çok sayıda farklı metot çağrısı ile sonuçlanabilir. Sınıfın nesnelerinin sistemin geri kalanıyla sahip olabileceği potansiyel etkileşimlerin derin bir anlayışını gerektirdiğinden, problemleri ayıklamak ve sınıf davranışını test etmek zor olabilir.

Kılavuz ve Yorum — RFC, test etme ve hata ayıklama çabalarının bir göstergesidir.

Bir mesajın çok sayıda metodun çağrılmasını tetiklemesi, sınıfın testinin ve hata ayıklamasının zorlaşması demektir.

Bir sınıftan fazla sayıda metodun çağrılması, sınıfın karmaşıklığının yüksek olduğuna işaret eder; çünkü bir mesaja cevap olarak çok sayıda metot çağrılabilir. Bu nedenle RFC, daha fazla anlaşılabilirlik gerektirir ve test süresine daha fazla zaman ayrılması gerekir.

RFC anlaşılabilirlik, bakım yapılabilirlik ve test edilebilirliği değerlendirmede kullanılabilir [73, 76, 78].

Metrik 4. Sınıfın Ağırlıklı Metot Sayısı (İng. Weighted Methods Per Class - WMC)

Tanım — WMC, bir sınıfın tüm metotlarının karmaşıklığının toplamıdır. Tüm metotların karmaşıklığı 1 kabul edilirse, WMC sınıfın metot sayısına eşit olur.

Bir C sınıfı M_1, M_2, … M_n metoda sahipse ve bu metotların karmaşıklığı c_1, c_2, …c_n ise sınıfın ağırlıklı metot sayısı aşağıdaki formülle hesaplanır:

$$WMC= \sum_{i=1}^{n} c_i \qquad (1)$$

Amaç — WMC metriği ile bir sınıfın karmaşıklığını ölçmek amaçlanmaktadır. Bu nedenle WMC, bir sınıfın geliştirilmesi ve bakımı için gereken çabanın bir göstergesi olarak düşünülebilir.

Kılavuz ve Yorum — Çok sayıda metot içeren sınıflar;

Geliştirilmesi ve bakımı için daha fazla çaba gerektirir. Tanımlanan tüm metotlar türetilen alt sınıflarda da yer alacağı için, çocuk düğümlerde daha çok etki bırakırlar.

Sınıf sayısı çok olan sınıfların uygulamaya özgü olma ihtimali yüksektir. Bu nedenle tekrar kullanılabilirliği düşürürler.

Ortalama WMC'deki bir artış hata yoğunluğunu arttırır ve kaliteyi düşürür [79].

Metrik 5. Nesne Sınıfları Arasındaki Bağımlılık (İng. Coupling Between Object Classes - CBO)

Tanım — CBO, sınıfın bağımlı olduğu diğer sınıfların sayısıdır. Bir sınıf içindeki nitelikler (İng. attribute) ya da metotlar başka bir sınıfta kullanılıyor ve bu sınıflar arasında kalıtım yoksa iki sınıf arasında bağımlılık olduğu kabul edilir.

Amaç — Bir sınıfın geliştirilmesi ve bakımı için gereken çabanın bir göstergesi olarak tanımlanmıştır.

Kılavuz ve Yorum — Sınıflar arasındaki aşırı bağımlılık modüler tasarıma zarar verir ve tekrar kullanılabilirliği azaltır. Bağımlılıktaki artış ile değişime duyarlılık da artacağından yazılımın bakımı daha zor ve testlerin daha özenli yapılması gerektiğinden de test maliyetleri daha yüksek hale gelir.

Yüksek bağımlılık bir sistemi karmaşıklaştırır çünkü bir sınıf, diğer sınıflarla bağımlıysa onu anlamak, değiştirmek veya düzeltmek daha zordur.

Metrik 6. Metotlardaki Uyum Eksikliği (İng. Lack of Cohesion in Methods - LCOM)

Tanım	C1 sınıfının M1, M2, … , Mn metotları olduğunu ve kümesinin de Mi metodunda kullanılan nitelik değişkenleri kümesi olduğunu kabul edelim. Bu durumda metotların uyumluluğu (LCOM - Lack of cohesion in methods), bu n kümenin kesişiminden oluşan ayrık kümelerin sayısıdır.
Amaç	LCOM metriği, farklı metotların kullandığı ortak özelliklerin sayısını ölçerek bir sınıfın uyumluluğu için bir kalite ölçüsü olarak amaçlanmıştır.
Kılavuz ve Yorum	Sınıfın uyumluluğunun düşük olması, sınıfın iki veya daha fazla alt parçaya bölünmesi gerektiğini gösterir. Düşük uyumluluk karmaşıklığı artırır ve bu nedenle, geliştirme aşamasında hata yapılma ihtimali yükselir. Metotlar arasındaki ilişkisizliklerin ölçüsü, sınıfların tasarımındaki kusurların belirlenmesinde de yardımcı olabilir.

Birçok çalışmada [80-83], özellikle metriklerin kalite göstergesi olarak doğrulanmasında, C&K metrik seti kullanılmıştır. Bu çalışmada C&K metrik seti üç sebeple tercih edilmiştir:

● C&K metrik setindeki yazılım metrikleri, önerilmelerinden günümüze kadar diğer metriklerden daha fazla test edilmiş ve onaylanmıştır [80].

● C&K metrik setindeki yazılım metrikleri, önemli kalite özniteliklerinin büyük kısmını (bakım yapılabilirlik, test edilebilirlik vb.) temsil etmektedir [80-82]. Örnek olarak DIT kalıtımı, CBO nesnelerin bağımlılığını, WMC sınıfın karmaşıklığını ve LCOM sınıflar arasındaki uyum eksikliğini ölçer. Özetle tüm metrikler, nesneye yönelik yazılımlarda kalitenin incelenmesine olanak sağlayan önemli özellikleri belirlemektedir.

● C&K metrik setindeki altı metriğin hepsi, literatürde kabul görmüş eşik değerlerine sahiptir. Bu eşik değerleri teoriler, deneyler ve istatistikler aracılığıyla doğrulanmıştır. Çizelge 2, altı C&K metriğinin ilgili referanslarıyla mevcut eşik değerlerini göstermektedir.

Çizelge 2. C&K metriklerine ait eşik değerleri

İlişkili Çalışmalar	WMC	RFC	DIT	LCOM	CBO	NOC
[25]	6-30	6-36	1-6	1-3	3-9	1-3
[26]	0-15	0-35	0-6	0-1	0-8	0-6
[27]	Düşük,<11	Düşük,<12	Düşük,<4	Düşük,0	Düşük,<3	Düşük,<3
[28]	Düşük	Düşük	Düşük	Düşük	Düşük	Düşük

Ayrıca, C&K metrik seti dışında aşağıdaki nesneye yönelik tasarım metrikleri de çalışmaya dâhil edilmiştir.

Metrik 7. Kod satır sayısı (LOC): Yazılımın büyüklüğü hakkında bilgi verir. Yazılım büyüklüğü, yazılım geliştirme maliyeti ve süresinin kestiriminde, başarım ve kalite ölçümünde vb. farklı amaçlarla kullanılmaktadır [84].

Metrik 8. Yorum satır sayısı (Comment Lines of code (CLOC)): Program için yazılmış yorum satırlarının sayısıdır. Yorum satır sayısının fazla olması, anlaşılabilirliği ve bakımı kolaylaştırmaktadır [85].

Metrik 9. İçsel bağımlılık (Afferent couplings (Ca)): Bir sınıfın dışında yer aldığı halde söz konusu sınıfa bağımlı sınıfların sayısıdır. Sınıfın değişmesi halinde etkilenecek sınıf sayısını gösterir. Bir sınıfa olan iç bağımlılık olarak da ifade edilebilir [86].

Metrik 10. Dışa bağımlılık (Efferent couplings (Ce)): İncelenen sınıfın kendi dışında kaç tane sınıfa bağımlı olduğunu gösterir. Bir sınıfın dışa bağımlılığını ifade eder [86].

2.7. Kalite Özellikleri ve Kaynak Kodu Öz nitelikleri

Yazılım kalite değerlendirmesi, bir dizi önceden tanımlanmış kurallara dayalı olarak, kod değerlendirmede sistematik bir yaklaşım sağlamak için belirlenmiş metriklerin toplanmasını gerektirir. Bu metrikler potansiyel olarak problemli alanları belirlemek için çeşitli göstergeler sunmaları açısından da yararlı olabilir. Bu

çalışmada uygulanan iç kalite değerlendirme metodunun aşamalarından biri, kaynak kod öz niteliklerini değerlendirmek için uygun metrik kümesini tanımlamaktır.

Çizelge 3. Kaynak kodu öz nitelikleri

Öz nitelik	Tanımı
Karmaşıklık	Nesnelerin birbiriyle olan ilişkisi inceleyerek yapısal özelliklere göre kavramadaki zorluğun derecesi olarak tanımlanmıştır [87].
Soyutlama	Nesne tanımlanırken verinin kullanılması veya veriye erişilmesi için gerekli detayların azaltılması işlemidir [87].
Bağımlılık	Nesneler arasındaki bağımlılık (nesnelerin ne kadar sıkı ilişkili olduğu) olarak tanımlanmıştır [87].
Uyumluluk	Bir sınıftaki metot ve niteliklerin tek ve iyi tanımlanmış bir amaca ulaşmak için birlikte çalışan tüm metotlara ne derece sahip olduğu ile ilişkilidir [87].
Mesajlaşma	Mesaj alışverişi yoluyla nesneler arasındaki işbirliği olarak tanımlanmıştır [87].
Kalıtım	Bir sınıfta tanımlanmış değişkenlerin ve/veya metotların yeniden tanımlanmasına gerek olmaksızın yeni bir sınıfa taşınabilmesidir [87].

Çizelge 3, C&K metrik seti ile ilişkili ve ISO 25010 kalite özelliklerini etkileyen kaynak kodu özniteliklerini, tanımlarıyla birlikte sunmaktadır. Çizelge 3'te verilen kaynak kodu özniteliklerinin her biri, bir veya daha fazla iyi tanımlanmış metrik kullanarak nesnel olarak değerlendirilebilir.

ISO 25010'a göre bir kaynak kodu özniteliği, bir veya daha fazla alt özniteliğe katkıda bulunabilir [70]. Bu kapsamda, analiz edilen metrikler ve her bir kalite özelliği arasındaki ilişki temel alınarak çeşitli yorumlar yapılabilir. Bu çalışmada, Lehman'ın artan karmaşıklık yasasına (Yasa 1) ilişkin oluşturulan hipotezler karmaşıklık ve bağımlılık özniteliklerini, azalan kalite yasasına (Yasa 3) ilişkin oluşturulan hipotez ise C&K metrik setinin uyumluluk özniteliklerini yansıttığından, dördüncü bölümde buna ilişkin sınıflandırma üç başlık altında gerçekleştirilmiştir.

2.8. DeLone ve McLean Başarı Modeli

AKY hakkında literatürde birçok kaynak mevcuttur ve bazı çalışmalar AKY'nın bir dereceye kadar başarısını da değerlendirmektedir. Bununla birlikte, AKY başarısı ile ilgili literatürdeki kaynaklar büyük oranda niteliksel (İng. qualitative) [88, 89] veya araştırmacı (İng. Exploratory) [90, 91] özelliktedir. AKY başarısını ölçmek için, çok az sayıda ampirik bir model oluşturulan çalışma mevcuttur. Bunun yerine, AKY başarısını ölçmek için, temel olarak iyi bilinen bir bilgi sistemi başarı modelini kullanan girişimler görülmektedir.

DeLone ve McLean başarı modeli, bilişim sistemlerinin yaygın olarak değerlendirildiği en kritik altı başarı boyutu arasındaki ilişkiyi tanımlayan ve

açıklayan bir teori olarak, açık kaynak yazılımların başarısının kapsamlı bir şekilde anlaşılmasını amaçlar. Teori ilk kez, 1992'de, William H. DeLone ve Ephraim R. McLean tarafından geliştirilmiş [92] ve daha sonra bu alanda çalışan diğer yazarlardan alınan geri bildirimlere yanıt olarak, ilk yazarlar tarafından revize edilmiştir [19, 93].

Crowston ve arkadaşları [94], AKY başarısının ölçülmesi için DeLone ve McLean [92] modeline dayalı bir model geliştirerek ve AKY projelerinin başarısını değerlendirmek için kullanılabilecek bir dizi boyut tanımlayarak çalışmalarıyla bu alana katkıda bulunmuştur. Ancak yazarlar, deneysel çalışma ile çalışmalarını doğrulamamışlardır. Seddon [95] modelinde algılanan yararlılık, bireysel net faydalar ve toplumsal net faydalar metriklerini dâhil ederek DeLone ve McLean [92] modelinin genişletilmiş ve yeniden tasarlanmış bir sürümünü önermiştir. DeLone ve McLean [19] daha sonra hizmet kalitesini içerecek şekilde özgün modellerini [92] güncellemişlerdir. DeLone ve McLean bir başka çalışma ile [19], güncellemenin bilişim sistemleri uygulamasındaki önemli değişiklikler, özellikle de e-ticaretin ortaya çıkması ve patlaması sonucunda gerekli olduğunu savunmuştur. Güncellenmiş DeLone ve McLean modeli [19], AKY'lara uygulanabilecek olası boyutları önermektedir. Güncellenmiş model sistem kalitesinin, bilgi kalitesinin ve hizmet kalitesinin hem bireysel hem de ortak olarak kullanıcı memnuniyetini ve kullanımını etkilediğini varsaymaktadır. Model aynı zamanda, kullanıcı memnuniyeti (İng. User satisfaction) ve kullanımın (İng. Use) karşılıklı olarak birbirine bağımlı olmasını önerirken bunların, bireysel etkinin doğrudan öncüllerini oluşturduğunu varsaymaktadır. DeLone ve McLean modelinde [19], sistem ve bilgi kalitesi (İng. System and Information Quality) boyutunda kod kalitesi, yazılım mühendisliğinde kapsamlı olarak incelenmiştir. Bu boyutta, yazılımın kod kalitesi üzerinde anlaşılabilirlik, etkililik, verimlilik, bakım yapılabilirlik gibi birçok metriğin ölçümü gerçekleştirilmektedir [96, 97].

AKY başarısını ölçmek için DeLone ve McLean [19] bilgi sistemleri başarı modelinin, iki nedenden ötürü Seddon ve arkadaşlarının çalışmalarına [98] kıyasla daha uygun olacağı düşünülmektedir. Birincisi, Seddon ve arkadaşları [98] organizasyonel faydaları ve toplumsal faydaları, modelin temel bileşenlerinden ikisi olarak sınıflandırmıştır ve bu sebeple kullanımının, firma ya da toplum düzeyinde daha uygun olduğu düşünülebilir. Ne var ki AKY kullananlar çoğunlukla bireysel

kullanıcılardır [94]. Hem bireysel etkiyi hem de organizasyonel etkileri karakterize eden modeli [19], bireysel kullanıcılar açısından deneysel testler için daha uygundur [99]. İkincisi, AKY başarısı üzerine yapılmış mevcut çalışmalardan çıkartılan boyutların DeLone ve McLean modeline [19] kolaylıkla dâhil edilebilmesi, bu modelin mevcut çalışma için en uygun referans model olduğuna işaret etmektedir.

AKY projeleri için başarı boyutları geliştirmek en az iki nedenden dolayı önemlidir. Birincisi, bu boyutlar, AKY proje yöneticileri için projelerini değerlendirmede işe yarayabilir. Bazı durumlarda, AKY projeleri üçüncü partilerin sponsorluğunda gerçekleştirilir; bu nedenle, sponsorların yatırım getirilerini anlamaları için bu boyutlar yararlı olacaktır. İkincisi, geliştiricilerin proje aktivitesi aşamasında projeye ne derece katkıda bulunduklarının tespit edilmesi önemli bir faktördür. Farklı topluluktan milyonlarca kullanıcı bu yazılımların geliştirilmesi aşamasında sonradan dâhil olabilmektedir, Scacchi'ye [100] göre bu kullanıcıların deneyim düzeyi, çalışma koşulları, projeye nasıl katkıda bulunduğu bilgileri çok az bilinmektedir. Bu sebeple, ölçülecek metriklerin belirlenmesi önemli bir aşamadır.
Açık kaynak mobil yazılım kullanımındaki hızlı artış ile hangi faktörlerin AKY başarısına yol açabileceğini bilmek önemlidir. AKY projeleri zamanla gelişir ve değişir; bu nedenle başarı belirli bir süre boyunca devamlı olarak incelenmelidir. Bir yazılımın uygulama marketindeki başarısının ne olduğu sorusunun cevabı sübjektiftir. Bunun için, AKY'nin market başarısını ölçebilmek adına birçok yöntem geliştirilmiştir. Bazı araştırmacılar AKY projelerinin başarısını kullanıcı tabanlı tanımlarken, bazıları geliştirici tabanlı olup geliştiricilerin katkıları ile tanımlamışlardır [94, 101, 102]. Raymond [103], AKY projelerinin başarısının, hataları düzeltme, özellik ekleme ve "sık ve erken" yazılım sürümlerinin ortaya konması gibi geliştiricilerin süregelen bir süreci ile karakterize olduğunu belirtmektedir. Feller ve Fitzgerald [101], Linux ve Apache gibi tanınmış açık kaynaklı projelerin pazar penetrasyonunun başarı boyutu olarak kullanılabileceğini gözlemlemişlerdir. Bununla birlikte, bu ölçüm az bilinen projeler için geçerli değildir.

Daha az bilinen projeler için başarı, projenin mevcut ve potansiyel kullanıcıları arasındaki popülerliği ile gösterilir [102]. Crowston ve arkadaşları [104], başarının, çok perspektiften değerlendirilmesi gereken çok boyutlu bir yapı olduğunu öne

sürmüştür. Grewal ve arkadaşları [105], sadece teknik başarılar veya sadece popülerlik açısından AKY projelerinin başarı oranının ölçülmesinin yetersiz olacağını savunmuşlardır. Ayrıca, AKY başarılarının kapsamlı bir şekilde hem geliştiricilerin teknik başarılarını hem de ticari başarısını (popülerlik) içermesi gerektiğini belirtmişlerdir. Hindle ve arkadaşları [106], bazı projelerin dış kaynaklardan önemli miktarda katkı aldığını ve bunların genellikle tek bir büyük değişiklik (İng. Commit) şeklinde olduğunu belirtmişlerdir. Bu çalışmada, commitlerin türleri (önemi ya da boyutu) arasında bir ayrım yapmamışlar ve gelecekteki çalışmalarda bu düzeyde analiz yapmayı planlamışlardır. Stewart ve Ammeter [91], "Başarılı olmayan açık kaynaklı yazılım projesinden başarılı olanı ayıran nedir?" sorusuna cevap bulabilmek için 240 açık kaynak kodlu proje üzerinde kullanıcı ilgisi (popülerlik) ve geliştirici aktivesini (projenin aktifliği-canlılığı) ne ölçüde etkilediklerini ölçebilmek adına; organizasyonel sponsorluk, lisans seçimi, geliştirici sayısı, sürüm sayıları arasındaki ilişkiyi incelemiştir. Ayrıca, daha aktif projelerin zamanla daha popüler hale gelebileceği varsayımıyla canlılık ve popülerlik arasındaki ilişki de incelenmiştir. Benzer şekilde, popülerlik, geliştiriciler arasında daha fazla etkinliği teşvik edebilir ve böylelikle canlılığı arttırabilir. Projedeki aktifliğin (geliştirici etkinliği) popülerlik üzerinde bir etkisi olduğu ve bir proje, yeni sürümlerinin ortaya çıkması açısından ne kadar canlı olursa, topluluğun aldığı ilginin o kadar artacağı sonucu ortaya çıkmıştır. Fakat projedeki canlılık üzerinde sponsorluğun, lisans durumunun ve proje gelişim sürecinin herhangi bir etkisine rastlanmamıştır.

Midha ve Palvia [107], proje başarısının iki boyutunu incelemişlerdir: zaman içindeki değişiklikleri gözlemlemek için üç yıllık bir süre boyunca 283 AKY projesi üzerinde proje popülerliği ve geliştirici etkinliği metrikleri incelenmiştir. Sonuç olarak, yüksek teknik başarıya sahip AKY projelerinin daha popüler olduğu hipotezi çürütülmüştür. Bu çalışmanın tersi yönünde Stewart ve arkadaşları [91] projenin popülerliğinin geliştirici aktivitesi üzerinde pozitif bir etkiye sahip olduğu sonucuna varmışlardır. Ayrıca daha fazla geliştiricinin proje başarısına olan etkisi hakkında net bir yorum yapılamayacağı sonucuna varmışlardır. Hataların düzeltilebilmesi, geri bildirimlere hızlı dönüş yapılabilmesi açılarından olumlu olabileceği gibi koordinasyon eksikliği, daha önce aynı projede yer almayan kişilerin projede kritik hatalar yapabilme olasılığının olması açılarından olumsuz

etki gösterebilmektedir. Buna rağmen Singh ve Tan [108] yapmış oldukları çalışmalarında başarılı bir AKY projesinin farklı tipten geliştiricilerin katılımıyla gerçekleşebileceğini savunmuşlardır. Kod üzerinde çekirdek bir geliştiriciye sahip olup geri bildirimlerin sağlanması, yaratıcı fikirlerin ortaya konması gibi sebepler açısından da diğer geliştiricilerin projeye dâhil olmalarının başarılı bir AKY projesine olumlu etkide bulunacağını savunmuşlardır. Colazo [109] ise yapmış olduğu çalışma sonrasında geliştirici sayısının artmasının proje aktivitesi üzerinde negatif bir etkiye sahip olduğunu gözlemlemiştir.

Subramaniam ve arkadaşları [110] yapmış oldukları çalışmada AKY lisanslarının AKY başarısı üzerindeki etkisi incelenmiştir. Sınırlı AKY lisanslarının AKY başarısı üzerinde olumsuz bir etkisi olduğu bulunmuştur. Daha fazla analiz için, sınırlı AKY lisansının geliştirici ilgisi ile olumsuz olarak ilişkili olduğu tespit edilmekle birlikte, geliştirici olmayan kullanıcıların ve proje yöneticilerinin ilgisi ile pozitif ilişkili olduğu görülmüştür. Ayrıca, geliştirici ve geliştirici olmayan kullanıcıların ilgisinin ve proje faaliyet seviyelerinin (proje etkinliği) AKY başarısını önemli ölçüde etkilediği görüşünü savunmuşlardır. Bir başka çalışmada Amrollahi ve arkadaşları [111], açık kaynak kod yazılımların başarısını değerlendirirken ilişkili çalışmaları dikkate alarak çeşitli metrikler belirleyip ölçmüşlerdir. Sonuç olarak, geliştirici sayısı, proje etkinliği, kullanıcı ilgisi (indirme sayısı) vb. metriklerinin proje başarısını etkiledikleri görülmüştür. Dolayısıyla proje başarısının tek bir perspektiften değil de her yönüyle (geliştirici, kullanıcı, teknik, sosyal, ekonomik) değerlendirilmesi gerektiği sonucuna varmışlardır. Crowston ve arkadaşları [94], AKY projelerinde başarıyı etkileyen boyutları baz alarak deneysel araştırmalara katkıda bulunmuşlardır. İncelenen metriklerin farklı perspektifler üzerinde değerlendirilmesi gerektiğini düşündükleri için farklı perspektiflere dayalı modeller geliştirerek deneysel çalışma gerçekleştirmişlerdir. Farklı metriklerin ölçülerek projenin başarısını değerlendirmenin ilginç olabileceğini belirtmişler. Yaptıkları çalışma sonucunda tanınabilirlik (ziyaret sayısı), kullanıcıların katılımı (kullanıcılardan gelen hatalar ve kullanıcı önerileri) ve popülerlik metriklerinin, AKY başarısı üzerinde etkisi olduğu görülmüştür. Bu yöntemler için kullanılan metrikler, açık kaynak kodlu yazılımların başarısını ölçerek ve takip ederek yazılımların gelişmesine yardımcı olmaya yönelik bilinçli kararlar vermeye olanak sağlamaktadır.

2.8.1. Toplum Tabanlı Metrikler

Çalışmada analiz edilmek üzere seçilen toplum tabanlı metriklerin detaylı açıklaması izleyen paragraflarda verilmiştir:

Metrik 1. Eleştirici sayısı (İng. Number of Reviewers - NOR)

Kullanıcı yorumları uygulama geliştiren kurumu veya geliştiricileri önemli ölçüde etkiler. Kullanıcılar bir ürünü ya çok beğendiklerinde ya da beğenmediklerinde yorum yapmaya yönelirler [112]. Kötü eleştiriler, satışları iyi eleştiriden daha çok etkilemektedir (olumsuz anlamda); çünkü alıcının düşük oylama ve şikâyetlere tepki gösterme olasılığı daha yüksektir [113]. Bu yüzden uygulamayı kullanıp yorum yapan eleştiricilerin sayısı, bir ürünün kalitesi yani başarısıyla ilgili tahmin yürütülmesine olanak sağlayabilmektedir.

Metrik 2. Kullanıcı Oyları (İng. User Ratings - UR)

Kullanıcıların verdiği tüm oylamalardan hesaplanan ortalama derecedir. Kullanıcılar bir ürünü 1-5 aralığında (1-en az beğenilen, 5-en fazla beğenilen) oylayabilmektedirler. Uygulama marketlerinde uygulama derecelendirmesi, tüm değerlendirmelerin toplamının eleştirici sayısına bölünmesiyle elde edilir ve kullanıcının uygulamadan ne kadar memnun kaldığı konusunda bir gösterge teşkil eder. Düşük oylanan uygulamalar, uygulamanın kalitesini negatif olarak etkiler, böylece uygulamanın popülerliği ve hatta geliri olumsuz etkilenmektedir [114].

Metrik 3. Geri bildirim sayısı (İng. Number of Feedback - NOF)

Kullanıcıların uygulamaya ne kadar geri dönüş yaptığı da önemli bir faktör olarak değerlendirilebilir. Bu geri dönüşler olumlu olabileceği gibi olumsuz da olabilmektedir. Bu sebeple, kullanım devamlılığını sağlamak için geliştiricilerin uygulama kalitesini sürekli yüksek tutması gerekmektedir.

Metrik 4. Katkıda bulunan geliştirici sayısı (İng. Number of Developer - NOD)

Birçok AKY projesi gönüllü geliştiricilere bağımlı olduğundan geliştiricileri sürekli olarak bir projeye çekebilmek AKY başarısı için önemlidir. Dolayısıyla bir projeye katılan geliştiricilerin sayısı, başarının bir göstergesi olabilir. Bir yazılıma katkıda bulunan geliştiricilerin sayısı (İng. contributor), proje yaşam döngüsü sürecine ve bir projenin zaman içinde nasıl büyüdüğüne, değiştiğine ve hız kazandığına birlikte etki etmektedir; yani yazılımın kalitesi hakkında bizlere bilgi verebilmektedir. Daha

fazla geliştiriciye sahip bir uygulama açılan hataların kısa süre içerisinde kapatılmasına olanak sağlayabildiği gibi, öncesinde hiç ortak projede çalışmamış kişilerin geliştirmeye dâhil olması daha fazla hatanın ortaya çıkmasına da sebebiyet verebilir.

Metrik 5. Değişiklik sayısı (İng. Commit Number - CN)

Kod satırı ekleme, değiştirme veya kaldırma, dosyaları ekleme veya kaldırma, dokümantasyon dosyalarındaki değişiklikler gibi bir kaynak kodu değişikliği, değişikliklere (İng. commit) örnek olarak verilebilir. AKY'larda kullanıcılar aynı zamanda geliştirici olabilecekleri için, değişiklik sayısı genelde projenin canlılığı açısından önemli bir metrik olarak değerlendirilmektedir.

Metrik 6. Sürümler arasındaki zaman (İng. Time between Releases - TBR)

Geliştirme topluluğu hizmet kalitesi, yani grup etkinliği ile ilgili bir diğer metrik, sürümler arasındaki süredir. Sürümlerin sık ve kısa sürelerle hazırlanması, AKY'nin sağlıklı bir proje ve gelişim sürecine sahip olduğunun kanıtıdır [103].

3. İLİŞKİLİ ÇALIŞMALAR

Lehman ilk olarak 1976 yılında yazılım evrimi üzerine deneysel bir çalışmaya öncülük etmiştir. Bu çalışmada Lehman [14], son araştırmalarında [16, 48-50] yazılım geliştirme evrimi yasaları olarak formüle edilen, sistemin büyüklüğü ve karmaşıklığı ile ilgili bir dizi gözlem yapmıştır ve bu yasalar, yazılımın karakter ve özelliklerini kendi evrim sürecinde tanımlamaktadır. Bir çalışmada Lehman [115], bir finansal işlem sistemi (İng. Financial Transactions System) üzerine yaptığı araştırmasının sonuçlarını, daha önceki OS/360 çalışmasında elde ettiği bulgularla karşılaştırmış; yeni analizin, yazılım evriminin yasalarını ne tam olarak desteklemekte ne de reddetmekte olduğunu gözlemlemiştir. Örneğin, beşinci ve yedinci yasaları için, teyit ya da aleyhine kanıt sağlayan yeterli veri bulamamıştır. Bu yasaları daha popüler ve yaygın bir şekilde sınamak için Lehman, iki çalışmada daha ayrıntılı kurallar, araçlar ve uygulamalar öne sürmüştür [115, 116].

Kemerer ve arkadaşları, yazılım karmaşıklığı ve yazılım bakımı arasındaki ilişkiyi araştırmışlardır [117]. Bu araştırmada yazarlar, McCabe döngüsel karmaşıklığını kaynak kod satır sayısıyla normalleştirmenin, yazılım bakım verimliliğinin önemli bir açıklayıcısı olduğunu fark etmişlerdir. Ayrıca, başka bir çalışmada Paulson ve arkadaşları [118], yazılım karmaşıklığını ölçmek için McCabe döngüsel karmaşıklığını kullanmayı önermişlerdir. İsrailli ve Feitelson [66], sistem evrimini karakterize etmek için, 14 yıllık bir süre boyunca yayımlanan Linux çekirdeğinin 810 sürümünün karmaşıklığını, McCabe karmaşıklık metriği ile izlemiştir. Bir başka çalışmada Kemerer ve arkadaşları [119], yazılım evrimi üzerine yapılan deneysel çalışma ile ilgili bir dizi yöntem ve teknik tanımlamışlardır. Yazarların deneysel çalışmaları, 23 ticari yazılım sisteminin 20 yılı aşkın süre boyunca karşılaştığı 25.000'den fazla değişim olayını toplama, kodlama ve analiz etmeyi içermektedir.

Yukarıda verilen yazılım evrimi üzerine çalışmaların çoğu [16, 49, 115-117] ticari sistemler üzerinde yoğunlaşırken AKY ile ilgili ilk çalışmalardan biri, Godfrey ve Tu [120] tarafından gerçekleştirilmiştir. Yazarlar ilk olarak, ticari bir sistemin açık kaynak bir yazılım ile olan gelişim sürecindeki farkını belirlemeye yönelik araştırmalar yapmışlar, daha sonra Linux çekirdeğinin 1994 ve 1999 yılları arasındaki gelişimini incelemiş ve Linux'un boyutunun lineer bir hızda artış gösterdiğini gözlemlemişlerdir. Aynı sonuçları, Vim metin editörü üzerinde

yaptıkları çalışmada da doğrulamışlardır. Linux'un 1996 sürümü üzerinde yapılan başka bir araştırmaya göre Schach ve arkadaşları [100], Linux modülleri arasındaki bağımlılığın ve karmaşıklığın üssel olarak arttığını ve bakım yapılabilirliğin zamanla azaldığını gözlemlemişlerdir. Wang ve arkadaşları [121], AKY gelişim sürecini değerlendirmek için bir dizi metrik önermiştir. Çalışmada, açık kaynak topluluğunun özelliklerine ve üyelerin açık kaynak yazılımların evriminde oynadığı role odaklanılmış ve Ubuntu projesi üzerinde basit bir vaka çalışması yapılmıştır. Araştırmaları sonucunda yazarlar, modüllerin sayısını, geliştiricilerin sayısını, belirli bir andaki hata sayısını, sabit hataların sayısını ve sabit hata sayısının mevcut hata sayısı metriğine oranını, açık kaynak kodlu yazılımların gelişimini değerlendirmede kullanışlı bulmuşlardır. Mockus ve arkadaşları [122], açık kaynaklı projeler için çekirdek geliştirici büyüklüğü, üretkenlik, hata yoğunluğu vb. unsurları ölçmek için, Apache Web sunucusunun kaynak kodu değişikliği geçmişine ve sorun raporlarına ait e-posta arşivlerini kullanmışlardır. Paulson ve arkadaşları [118], açık kaynaklı ve kapalı kaynaklı yazılım projeleri hakkındaki ortak algıları araştırmak ve çeşitli hipotezleri test ederek bu algıları doğrulamak için deneysel bir çalışma gerçekleştirmiş; altı yazılım projesinin (üç açık kaynaklı ve üç kapalı kaynaklı) büyümesini, üretkenliğini, hatalarını ve modülerliğini araştırmışlardır.

Bir çalışmada [56], çeşitli özniteliklere (boyut, modül sayısı ve geliştiriciler) dayanan 12 açık kaynaklı proje analiz edilmiş ve projelerin büyümesi bazı noktalarda en üst seviyede olmakla birlikte, büyümenin zamanla daha istikrarlı bir hale geldiği gözlenmiştir.

Antoniol ve arkadaşları [123], Mozilla, Alice ve Eclipse projelerinin kararlılıklarını incelemek üzere analizler gerçekleştirmişlerdir. Onların gözlemleri, bu sistemlerin daha yüksek bir kararlılığa doğru evirildiğini, ancak zaman zaman değişimlere uyum sağlamak için kararsızlıkların ortaya çıktığını ve eğilimin tersi bir seyir gösterdiğini ortaya koymuştur.

Izurieta ve Bieman [124], FreeBSD'nin evrimini (34 kararlı sürüm) ve Linux'un evrimini (127 kararlı sürüm) yeniden incelemiş; her sürüm için toplam büyüklüklerini sürüm sayılarıyla ölçüp her iki sistemin de büyümesinin Godfrey ve Tu [120] ile tutarlı olmadığına dair kanıtlar bulmuşlardır. Gonzalez ve arkadaşları

[125], Debian GNU / Linux hakkında keşifsel bir çalışma yapmış ve Debian'ın her iki yılda bir boyutunun ikiye katlandığını görmüşlerdir. Peng ve arkadaşları [126], işletim sistemi özelliklerinin ve işletim sistemi çevresel parametrelerini kullanarak işletim sistemlerinin gelişimini modellemeye çalışmış, bu da yazılım teknolojisi eğilimlerinin evrimini daha iyi anlamamızı sağlamıştır.

Herraiz ve arkadaşları [127], SourceForge'daki 3821 Libre projesinin evrimini analiz etmiş; projelerin evrimini sınamak için ana metrik olarak LOC, değişiklik sayısı ve dosya sayısını kullanmışlardır. Li ve arkadaşları'nın yaptıkları çalışmada [17] ise mobil uygulamaların evrimini analiz etmek için LOC, sürümler arasındaki değişiklik sayısı, metot ve sınıf sayısındaki değişiklikler göz önüne alınmıştır. Ayrıca Herraiz'in çalışmasında [128], bir yazılım sistemini karakterize etmek için temel boyut metriklerinin küçük alt kümelerinin de yeterli olduğu fark edilmiştir.

Ramil ve arkadaşları [129], 11 büyük yazılımın (Blender, FPC, Eclipse, GCC, GCL, GIMP, GDB, GNUBinUtils, XEmacs, WireShark ve NCBITools) evrimini gözlemleyerek deneysel bir çalışma gerçekleştirmiş; LOC, ay bazında değişiklik yapan geliştirici sayısı, dosya sayısı ve katkıda bulunan geliştirici sayısı metriklerini kullanmışlardır. Kaynak kodunun boyutunun, LOC veya dosya sayısı fark etmeden ölçüldüğünde, zamanla artma eğiliminde olduğunu gözlemlemişlerdir. Ayrıca Li ve arkadaşları [17], mobil uygulamaların boyutunu bu metriklerle ölçmüşler ve son sürümlere doğru özellikle bazı sürümlerde hızlı bir büyüme eğiliminin olduğunu gözlemlemişlerdir.

Neamtiu ve arkadaşları [18], yazılımların gelişimindeki evrimsel süreçleri ve benzerlikleri karakterize etmek için, altı açık kaynaklı projenin (Bash, BIND9, OpenSSH, Samba, SQLite ve Vsftpd) sürümleri boyunca deneysel bir çalışma yürütmüşlerdir. İlk olarak, Lehman'ın yazılım geliştirme yasalarını bu projelerde istatistiksel hipotez testi kullanarak onaylamaya çalışmışlar; daha sonra, değişimlerin dağılımında "paralel evrim" ve "sıcak noktalar" gibi yeni bulgular ortaya koymuşlardır. Çalışmaları geleneksel masaüstü uygulamalarına odaklanmıştır.

Godfrey ve Tu'ya göre [120], açık kaynak kodlu yazılımların büyümesi ve karmaşıklıklarının artması kaçınılmazdır. Yukarıdaki çalışmalardan yola çıkarak

büyümenin devam etmesi, karmaşıklığın artması ve kalitenin düşmesi, AKY gelişiminde sık görülen olgular olarak değerlendirilmektedir.

Evrimsel açıdan mobil uygulamalardaki ilk çalışmalardan biri, Zhang ve arkadaşları [11] tarafından yapılmıştır. Bu çalışmada Lehman yasalarından üçünün, yani sürekli büyüme, artan karmaşıklık ve azalan kalitenin, mobil uygulamalara uygulanabilirliği incelenmiştir. Yazarlar bu kapsamda belli metrikler elde etmişler, VideoLAN ve ownCloud uygulamaları için bir vaka çalışması yapmışlardır. Bulguları, sürekli büyüme ve azalan kalite yasalarının mobil uygulamalar için geçerli olduğunu gösterirken artan karmaşıklık yasası için farklı sonuçlar ortaya çıkarmıştır. Minelli ve Lanza çalışmalarında [10], mobil uygulamaların yapısal ve tarihsel bir perspektiften anlaşılmasına yönelik yeni bir yaklaşım sunmuştur. Örneğin, hazır metot çağrılarının eklenmesiyle uygulama boyutlarını ve karmaşıklığını ilişkilendirmişlerdir ve incelenen mobil uygulamaların, geleneksel yazılımlardan önemli ölçüde farklılık taşıdığını savunmuşlardır.

Harman ve arkadaşları [130], veri madenciliğini, Blackberry uygulama mağazasındaki mobil uygulamaların özellik bilgilerini almak için kullanmışlar ve kullanıcı oyları ile indirme sayısı arasında bir korelasyon olduğunu gözlemlemişlerdir. Pagano ve Maalej [131] ise uygulama mağazalarında kullanıcıların geri bildirimine ilişkin keşifsel bir çalışma düzenlemişler, geri bildirimlerin çoğunun yeni sürümlerden sonra kısa sürede gerçekleştiğini ve geri bildirimlerin içeriklerinin indirme sayıları üzerinde etkisi olduğunu belirtmişlerdir.

Hecht ve arkadaşları [132], Android uygulamalarının yazılım kalitesini evrimleri boyunca izlemek için, deneysel bir çalışma yapmışlardır. Yazılımdaki negatif örüntüleri (İng. Antipattern) analiz ederek uygulamaların zaman içindeki gelişimini, ‘Paprika’ isimli bir araçla gözlemlemeye çalışmışlardır. Gyimothy ve arkadaşları [63] yazılım kalitesini ölçmek için, LCOM ve kararsızlık kalite özelliklerinin kullanılmasını önermişlerdir. Neamtiu ve arkadaşları [18], yazılım sisteminin kalitesinin, iç ve dış kalite açısından değerlendirilmesi gerektiğini öne sürmüştür. Dış kalite analizi için hata yoğunluğu metriğini analiz ederken iç kalite için karmaşıklık analizi gerçekleştirmiştir. Dış kalite, kullanıcının yazılımı kabul ve memnuniyetini ifade etmektedir. Mobil uygulamalar yaygın olarak kullanıldığından

ve kullanıcılar tarafından oylanabilir olduğundan, dış kalite kullanıcı perspektifinden kabul edilebilirdir.

Bir başka çalışmada ise Li ve arkadaşları [17], mobil uygulamaların evriminde Lehman'ın sekiz yasasının geçerliliğini, sekiz açık kaynak kodlu mobil uygulamanın 348 sürümü üzerinde çeşitli hipotez testleri gerçekleştirerek sınamışlardır. 'Artan karmaşıklık' yasası üzerine yapmış oldukları hipotez testine göre, sekiz uygulamadan beşinde bu yasa geçerlenmemişken 'sürekli büyüme' yasasına göre yapmış oldukları test sonuçları, tüm uygulamalar için geçerlenmiştir. 'Azalan kalite' yasasına göre oluşturulan hipotezler ise hiçbir uygulama için geçerli bulunmamıştır.

Yapılan çalışmaların çoğu iç kalite üzerine yoğunlaşırken literatürde hem açık kaynak hem mobil yazılıma odaklanan ve hem de iç ve dış kalite arasındaki ilişkiyi inceleyen çalışma yok denecek kadar azdır. Hâlbuki bu ilişki ile ilgili deneysel çalışmalar, genel yazılım kalitesini iyileştirmek ve yazılım geliştirme sürecinde değişiklikler önermek için, geliştiricilere ya da uygulama mağazalarına bir yol sunabilir. Stamelos ve arkadaşları [133], Linux üzerine yaptıkları çalışmada, iç kalite özellikleri (test edilebilirlik ve açıklık (İng. Simplicity)) ile dış kalite (kullanıcı memnuniyeti) arasındaki ilişkiyi incelemiş ve istatistiksel testler sonucunda aralarında bir ilişki olmadığını gözlemişlerdir. Wang ve arkadaşları [121], Ubuntu üzerine deneysel bir çalışma yapmışlar ve hata sayısı, çözülen hata sayısı, modül sayısı metriklerini, iç kalite analizinde kullanırken katkıda bulunan geliştirici sayısı ve aktif kullanıcı sayısı metriklerini, dış kalite analizinde kullanmışlardır. Çalışmada ayrıca iç ve dış kalite metrikleri arasındaki ilişki, Pearson korelasyon analizi ile tespit edilmiş ve bu metriklerin aralarında anlamlı bir ilişki olduğu tespit edilmiştir. Capiluppi ve arkadaşları [57] açık kaynak bir yazılımın 64 sürümü üzerinde büyüme ile ilişkili metrikleri analiz etmiş ve çalışma bulguları kod satır sayısı, fonksiyon sayısı, dosya sayısı metriklerinde zamanla bir artış olduğunu göstermiştir. Ayrıca yazarlar, projeye katkıda bulunan geliştirici sayısının da zamanla artış gösterdiğini gözlemişlerdir. Ancak iç ve dış kalite özellikleri arasında herhangi bir ilişki tespit edilmemiştir. Goeminne ve Mens' in [134] Netbeans, Eclipse ve ArgoUML açık kaynak kodlu projelerini kullanarak yaptıkları çalışma, yazılımın iç kalitesi ve kullanıcı memnuniyeti, popülerlik, geliştirici aktivitesi gibi dış kalite boyutları arasındaki ilişkiyi analiz etmeyi amaçlamıştır. Abreu ve

arkadaşlarının yaptıkları çalışmada [135], geliştiriciler arasındaki iletişim ve koordinasyon ile yazılımdaki hataların sayısı arasında ilişki tespit edilmiştir. Kawin ve arkadaşlarına göre ise [136], evrimsel örüntüler üzerinde yapılan araştırmalar, bir geliştirici ekibinin yapısal organizasyonu ile yazılım kalitesinin gelişimi arasında ilişki olduğunu göstermektedir.

Bu tez çalışmasında, sadece açık kaynak, sadece mobil, sadece iç kalite ya da sadece dış kaliteye odaklanan çalışmalardan farklı olarak, açık kaynaklı mobil yazılımların hem iç hem de dış kalitesi üzerine yoğunlaşmış, ayrıca çalışmanın sonunda bunlar arasındaki ilişki tespit edilmeye çalışılmıştır. Li ve arkadaşları [17], Zhang ve arkadaşları [11], Minelli ve Lanza'nın [10] yapmış oldukları çalışmalarda olduğu gibi mobil uygulamaların evrimi boyunca iç kalite değerlendirilmiş, Harman ve arkadaşları [130] ile Pagano ve Maalej'in [131] çalışmalarına benzer olarak da mobil uygulamaların evriminde dış kalite değerlendirilmiştir. Bu çalışmalardan farklı olarak mevcut çalışmada ayrıca farklı metrik setleri ile çalışılmış ve iç kalite ile dış kalite arasındaki ilişkinin tespiti korelasyon analizi ile gerçekleştirilmiştir.

4. YÖNTEME GENEL BAKIŞ

Bu çalışmada iç ve dış olmak üzere iki boyutlu kalite değerlendirme amaçlanmış ve her boyutun değerlendirmesi, farklı yollar izlenerek gerçekleştirilmiştir. İç kalite kod tabanlı değerlendirilirken dış kalite toplum tabanlı olarak değerlendirilmiştir. Çalışmanın bundan sonraki bölümlerinde kod tabanlı değerlendirme için iç kalite, toplum değerlendirme için dış kalite ifadeleri kullanılacaktır.

4.1. İç Kalite Değerlendirmeye İlişkin Esaslar

Kod tabanlı değerlendirme sürecinde, ISO 25010 kalite standardı [70] ile kalite özellikleri ve nesneye yönelik tasarım metrikleri temel alınarak Lehman'ın sekiz yazılım geliştirme yasasından kalite ile ilişkili olan 'artan karmaşıklık', 'sürekli büyüme' ve 'azalan kalite' yasaları ışığında hipotezler oluşturulmuştur. Çalışmaya özel olarak belirlenen araştırma sorularının yanıtlanması için oluşturulan bu hipotezler, üç açık kaynak kodlu mobil uygulamanın sürümleri boyunca toplanan yazılım tasarım metrikleri üzerinde istatistiksel analizler yapılarak test edilmiştir.

Her bir kaynak kodu öz niteliği, C&K metrik setine ait metriklerle ölçülebilir ve bu ölçüm sonuçları, ISO 25010'a ait kalite özellikleri hakkında öngörüde bulunmayı sağlayabilir. Bu sebeple Çizelge 4, yazılımın her bir kaynak kodu özniteliğiyle ilişkili nesneye yönelik metrikler ile ISO 25010'un yazılım kalite özelliği/alt-özelliği arasındaki ilişkiyi göstermektedir.

Çizelge 4. Kaynak kodu özniteliği, C&K metrik seti ve yazılım kalite özellikleri arasındaki ilişki

Kaynak Kodu Özniteliği	**C&K Metriği**	**İlişkili Çalışmalarda Ele Alınan ISO 25010 Kalite Özelliği/Alt-özelliği**	**İlişkili Çalışmalara Referanslar**
Karmaşıklık	-WMC -RFC -DIT -NOC	Bakım yapılabilirlik, Anlaşılabilirlik, Kullanılabilirlik, Yeniden kullanılabilirlik, Test edilebilirlik, Verimlilik, Taşınabilirlik	[73, 76-78]
Uyumsuzluk	-LCOM	Yeniden Kullanılabilirlik, Verimlilik, İşlevsellik, Bakım yapılabilirlik, Taşınabilirlik	[73, 76-78]
Bağımlılık	-CBO	Yeniden Kullanılabilirlik Verimlilik, Bakım yapılabilirlik, Taşınabilirlik	[73, 74, 76-78]

Kalite özelliği, sistemin paydaşlarını tatmin etme kabiliyeti için kullanılan, ölçülebilir bir özelliktir [69]. Bir yazılımın çeşitli bakış açılarından kaliteli olup olmadığını değerlendirmek için bu özelliklerden faydalanılmaktadır. Bu tez çalışmasında kullanılan iç kalite özelliklerine ait detaylar Çizelge 5'te verilmiştir:

Çizelge 5. Çalışmada incelenen her bir kalite özelliği, tanımı ve formülü

Kalite Özelliği	**Tanımı**	**Formülü**
Kararsızlık	Yeni gereksinimlerle karşı karşıya kalırken veya platform değiştirirken yazılımın değişime karşı gösterdiği direncin ölçüsü olarak tanımlanır [69]. Bakım yapılabilirlik kalite özelliğinin alt-özelliğidir. Yazılım sisteminin kararlığı, yazılımın sürümler boyunca karşı karşıya kaldığı geliştirme aktiviteleri için gereken ek çaba ve maliyeti düşürmesi açısından önem taşır.	Ce / (Ce+Ca)
Taşınabilirlik	Bir sistem, ürün veya bileşenin bir donanım, yazılım, başka bir işletim veya kullanım ortamından diğerine aktarılabilme, farklı çalışma ortamlarına uyum sağlayabilme yeteneğidir [69]. Taşınabilirlik, özellikle sürekli büyüyen mobil pazarda, bir yazılım ürününün sahip olması arzu edilen bir özelliktir.	1 - DIT
Anlaşılabilirlik	Farklı altyapıya sahip kullanıcıların mimariyi ne ölçüde anlayabileceklerini ifade eder [69]. Yazılım mimarisinin anlaşılmasındaki zorluk, yazılımın yeniden kullanımını ve bakımını engellemektedir. Bu sebeple anlaşılabilirlik kalite özelliği, yeniden kullanılabilirlik ve bakım yapılabilirlik kalite özellikleriyle ilişkisi bakımından bu çalışmaya dâhil edilmiştir.	1 - [(0.25 * CBO)+ (0.25 * LCOM) + (0.25 * Cloc) + (0.25 * (0.5*LOC) + (0.5 * metot sayısı))]

Bu tez çalışmasında, Lehman'ın yasaları ışığında karmaşıklık, uyumsuzluk ve bağımlılık öznitelikleri kullanılacağından Çizelge 4 bu üç başlık altında incelenmiştir. Ayrıca her bir öznitelik kalite değerlendirmesinde tahminde bulunmayı sağladığından Çizelge 5 ile bu kaynak kodu özniteliklerini ölçmeye yarayan metrikler ve çalışmada değerlendirilmesi hedeflenen kalite özelliklerine ilişkin detaylar verilmiştir.

4.2. Dış Kalite Değerlendirmeye İlişkin Esaslar

Toplum tabanlı değerlendirme sürecinde, belirlenen ihtiyaçlar doğrultusunda doğrudan uygulama mağazalarından erişimi sağlanan bilgilerin elde edilmesine kadar olan aşamada, DeLone ve McLean modeli [19] takip edilmiştir.

Çizelge 6. Önceki çalışmalar ve çalışmamıza temel olarak uyarladığımız başarı modeli

Modeller	**Boyutları**
(DeLone and McLean 1992)	Sistem Kalitesi (İng. System Quality) Bilgi Kalitesi (İng. Information Quality) Kullanım (İng. Use) Kullanıcı Memnuniyeti (İng. User Satisfaction) Bireysel Etkiler (İng. İndividual Impacts) Kurumsal Etkiler (İng. Organisational Impacts)
(Seddon 1997)	Sistem Kalitesi Bilgi Kalitesi Algılanan Yararlılık (İng. Perceived usefulness) Kullanıcı Memnuniyeti Bireysel Faydalar (İng. Individual Benefits) Kurumsal Faydalar (İng. Organisational Benefits) Sosyal Faydalar (İng. Social Benefits)
(DeLone and McLean 2003)	Sistem Kalitesi (a) Bilgi Kalitesi (b) Kullanım-Kullanım Amacı (İng. Use-Intention to Use) (c) Kullanıcı Memnuniyeti (d) Net Faydalar (İng. Net Benefits) (e) Hizmet Kalitesi (İng. Service Quality) (f)
Mevcut Çalışmamıza temel olarak uyarlanan AKY Başarı Modeli	Yazılım Kalitesi (İng. Software Quality) (1) Kullanım (2) Kullanıcı Memnuniyeti (3) Geliştirme Topluluğu Hizmet Kalitesi (İng. Development Community Service Quality) (4)

Çizelge 6, önceki modellere ve bu çalışma için uyarlanan modele ilişkin boyutları göstermektedir. Uyarlanan modelin boyutları ile bu boyutların DeLone ve McLean modelinin [19] boyutlarına olan izlenirliği, takip eden paragraflarda açıklanmıştır.

Sistem Kalitesi ve Bilgi Kalitesi (Çizelge 6 (a,b; 4)): Bu bölümde Çizelge 6'daki DeLone ve McLean modeline ait (a, b) ve çalışmamıza temel olarak uyarladığımız AKY Başarı Modeli' ne ait (4) 'Sistem Kalitesi ve Bilgi Kalitesi' boyutuna dair detaylara yer verilmiştir.

AKY kendisi bir uygulama sistemi değildir ve çıktı olarak bilgi üretmez veya işlemez. DeLone ve McLean'in [19] bilgi kalitesi boyutu, hedef alınan bir sistem tarafından üretilen bilgilerin doğruluğunu, anlamlılığını ve zamanlamasını karakterize etmektedir. Bilgi kalitesi, AKY tabanlı uygulama sistemlerinin önemli bir unsuru olabilirken, hedeflenen yazılımlar (örneğin, Keepassdroid) herhangi bir bilgi üretmek için tasarlanmamıştır. Bu nedenle, AKY'nın başarısını ölçmek için 'Bilgi Kalitesi' yapısı DeLone ve McLean [19] başarı modelinden çıkarılmıştır. Benzer şekilde, AKY yazılım yönü ile karakterize edilmiştir. Bu nedenle DeLone ve McLean'in 'Sistem Kalitesi' boyutu 'Yazılım Kalitesi' olarak değiştirilmiştir. Bu

aşamada, yazılımın kod kalitesi üzerinde anlaşılabilirlik, etkililik, verimlilik, bakım yapılabilirlik gibi birçok metriğin ölçümü gerçekleştirilebilmektedir [96, 97]. Bu tez çalışmasında, yazılımın kod kalitesi üzerinde uyumsuzluk, kararsızlık, taşınabilirlik ve anlaşılabilirlik kalite özelliklerinin analizi gerçekleştirilmiştir.

Kullanım (Çizelge 6 (c; 2)): Bu bölümde Çizelge 6'daki DeLone ve McLean modeline ait (c) ve mevcut çalışmaya temel olarak uyarlanan AKY Başarı Model'ine ait (2) 'Kullanım' boyutuna dair detaylara yer verilmiştir.

Uygunluğu hakkında bazı tartışmalar [19, 95] olsa da, pek çok çalışma proje kullanımını AKY başarısının bir göstergesi olarak ele almaktadır. Çoğu AKY'de olduğu gibi, kullanımı isteğe bağlı olan yazılımlar için kullanım, proje başarısının potansiyel olarak önemli bir göstergesi gibidir. Gerçek kullanımı ölçmek adına, popülerlik olarak adlandırılan, yazılımın gerçek veya potansiyel kullanıcı sayısını saymak yeterli olabilir [91]. Ayrıca, projelerin e-posta listelerinde ya da tartışma gruplarında yer alan geri bildirim yoğunluğu da önemli bilgiler verebilmektedir. Ek olarak, başka bir popülerlik ölçüsü olarak yazılımın indirilme sayısı da bu boyutta değerlendirilebilir.

Kullanıcı Memnuniyeti (Çizelge 6 (d; 3)): Bu bölümde Çizelge 6'daki DeLone ve McLean modeline ait (d) ve çalışmamıza temel olarak uyarladığımız AKY Başarı Modeli' ne ait (3) 'Kullanıcı Memnuniyeti' boyutuna dair detaylara yer verilmiştir.

Kullanıcı memnuniyeti, AKY başarısında sıklıkla kullanılan bir boyuttur. Kullanıcı oyları uygulama geliştiren organizasyon ya da geliştiricileri önemli ölçüde etkiler. Paydaşlara, bir projenin başarısı konusunda fikirlerini sormak da yaygın bir uygulamadır [137]. AKY projelerindeki kullanıcı memnuniyeti ile ilgili bazı veriler mevcuttur. Örneğin, kullanıcı oyları bir yazılımın ne kadar başarılı olduğuna dair tahminler yürütülmesine olanak sağlar.

Net Faydalar (Çizelge 6 (e)): DeLone ve McLean'in modelindeki bu boyut, kullanıcılar için net faydalardır. Bu boyut, AKY projeleri için pek geçerli değildir. AKY'ların kaynak kodlarının herkes tarafından görülebilir olması sebebiyle bu yazılımlar için ancak çok düşük bir fiyat talep edilebilir. Çünkü kodu elde eden herhangi bir geliştirici bu yazılımı özgürce yeniden dağıtabilir; dolayısıyla fiyatlar, açık kaynak kodlu yazılımların ortalama dağıtım maliyetleriyle doğru orantılıdır. Hatta İnternet kullanımının her geçen gün daha da yaygınlaşıyor olması, bu

yazılımların İnternet üzerinden indirilmesini mümkün kılmakta, bununla birlikte dağıtım maliyetleri sıfıra yaklaşmaktadır. Dolayısıyla bu boyutun, AKY projelerinin çoğunda yapılan çalışmalar için kullanılamayacağı düşünülmektedir.

Hizmet Kalitesi (Çizelge 6 (f; 4)): Bu bölümde Çizelge 6'daki DeLone ve McLean modeline ait (f) ve mevcut çalışmaya temel olarak uyarladığımız AKY Başarı Model'ine ait (4) 'Hizmet Kalitesi' boyutuna dair detaylara yer verilmiştir.

DeLone ve McLean modelindeki 'Hizmet Kalitesi' terimi de bu çalışma için önemlidir. Bilgi sistemleri başarı modelinde hizmet kalitesi, kurumsal departmanlara bilgi teknolojileri departmanı tarafından sağlanan hizmetlerin etkinliğini ölçer. AKY bağlamında hizmet kalitesi ise AKY geliştirme topluluğundan gelen teknik destek şeklinde yorumlanabilir [138]. AKY geliştirme topluluğu, bilgi teknolojileri departmanı rolüne benzer bir rol oynayabilir. Çünkü AKY geliştirme topluluğu AKY'yı geliştirir ve değiştirir, ayrıca AKY kullanıcılarına yazılım ile alakalı bilgi sağlamak da dâhil olmak üzere ek hizmetler sunar. Dolayısıyla AKY bağlamında hizmet kalitesi, geliştirme topluluğunun hizmet kalitesi olarak düşünülebilir. Kullanıcıların hem geliştirme hem de geliştirme sonrası aşamalarda destek, işbirliği ve yardıma ihtiyacı vardır. İlk kurulumdan sonraki yeni yazılım sürümlerinin yükseltilmesi, değişikliklerin gerçekleştirilmesi (yeni özellik ekleme ya da var olan özelliklerin çıkartılması) ve yeni sürümlerin hazırlanması da bir destek biçimi olarak düşünülebilir.

Sonuç olarak, mevcut bilgi sistemleri modelleri, AKY projeleri için bir dizi potansiyel başarı boyutu önermektedir. Bu çalışmada ise, dört farklı başarı boyutu ve altı farklı metriği kapsayan model ile hem kullanıcı hem de geliştirici perspektifinden değerlendirmeler gerçekleştirerek çeşitli analizlerin yapılması amaçlanmaktadır.

Literatür çalışmalarından elde edilen bilgi çıkarımlarını dikkate alarak, tez çalışması kapsamında market başarısını ölçebilmek adına yazılım kalitesi, kullanım, kullanıcı memnuniyeti ve geliştirme topluluğu hizmet kalitesi ölçümlerine odaklanmaktadır.

Kullanım boyutunu değerlendirebilmek adına uygulamanın popülerliği, market başarısı hakkında bir öngörü sunabilmektedir. Öyle ki kullanıcıların uygulamayı indirip kurması, kullanması ve de oylaması, ürünü ne kadar beğendikleri ile

alakalıdır. Uygulamalara yapılan yorum sayısının ve buna bağlı olarak eleştirici sayısının fazla olması, uygulamanın daha fazla kullanıcıya ulaşması anlamı taşıyabilmektedir. Ayrıca, genelde kullanıcılar ya bir hata ile karşılaştıklarında ya da bir öneri sunmak üzere tartışma gruplarında, çeşitli gönderiler paylaşmaktadırlar. Bu sebeple, tartışma gruplarında geri bildirimlerin fazla olduğu uygulamalar, başarısı yani kullanımı hakkında tahminler yürütülmesine olanak sağlayabilmektedir. Bunun için ‘eleştirici sayısı’ ve ‘geri bildirim sayısı’ metrikleri ölçülmüştür.

Kullanıcı memnuniyetini ölçebilmek için kullanıcıların uygulamaya vermiş oldukları oylar değerlendirmeye alınabilir. Bunun için ‘kullanıcı oyları’ metriği ölçülmüştür.

Geliştirme topluluğu hizmet kalitesini ölçebilmek için ise katkıda bulunan geliştirici sayısı ve bu geliştiricilerin uygulamaya yapmış oldukları değişiklikler incelenmiştir. Ayrıca "sık ve erken" yazılım sürümlerinin ortaya konması AKY başarısının bir boyutu olarak kabul edildiği için, geliştirici topluluğu hizmet kalitesini ölçebilmek amacıyla ‘geliştirici sayısı’, ‘değişiklik sayısı’ ve ‘sürümler arasındaki zaman’ metrikleri kullanılmıştır.

Çizelge 7, bu çalışmada incelenen başarı boyutları ile ilişkili metrikleri ve ilişkili çalışmaları içermektedir.

Çizelge 7. İncelenen başarı boyutları ve ilişkili metrikler

Başarı Boyutu	**Metrikler**	**İlişkili Çalışmalar**
Kullanım	● Eleştirici sayısı ● Geribildirim sayısı	[19, 91, 94, 111]
Kullanıcı Memnuniyeti	● Kullanıcı oyları	[19, 91, 94, 101, 102]
Geliştirme Topluluğu Hizmet Kalitesi	● Geliştirici sayısı ● Değişiklik sayısı ● Sürümler arasındaki zaman	[19, 91, 94, 103, 108, 111]

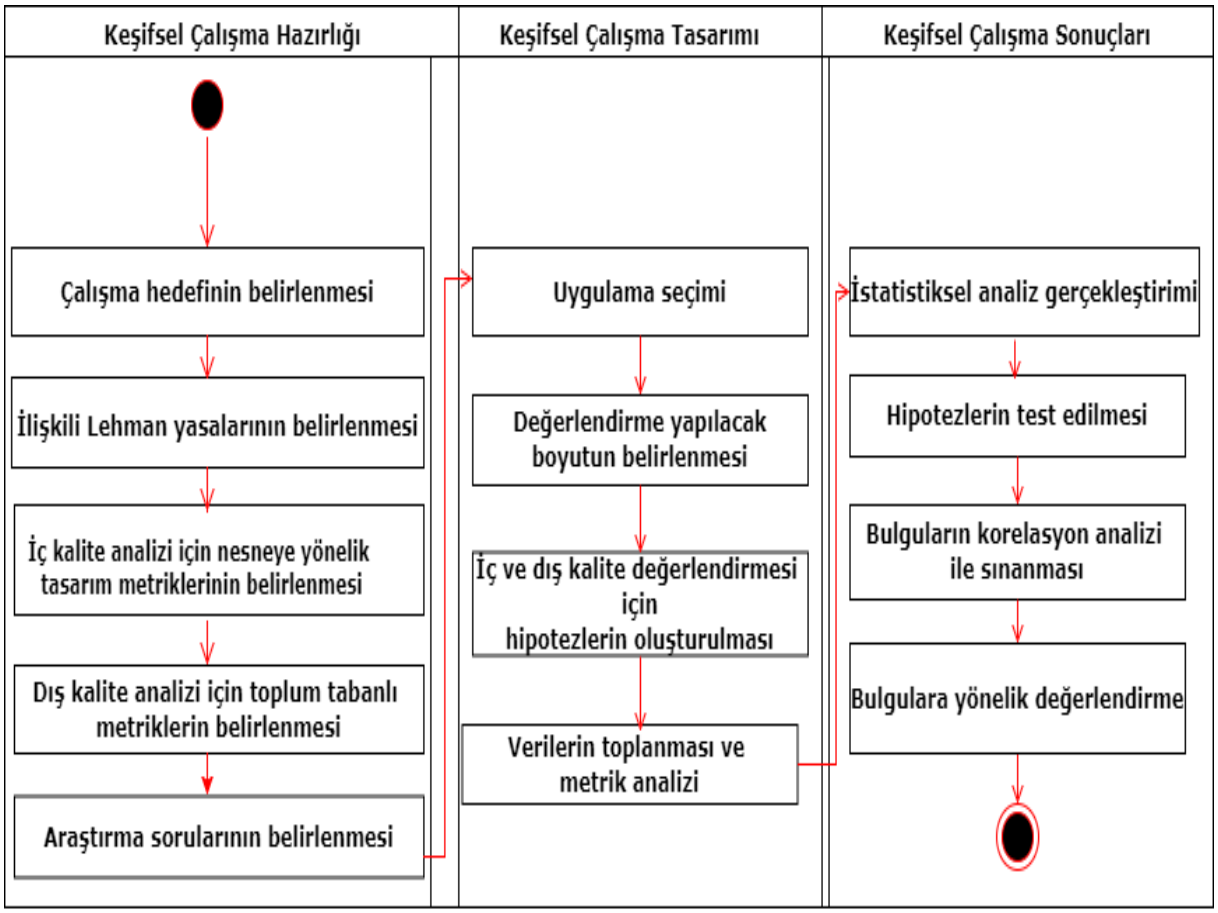
5. MOBİL UYGULAMALARIN EVRİMİ İÇİN KEŞİFSEL ÇALIŞMA TASARIMI

Bu tez çalışmasının tasarımı; keşifsel çalışma hazırlığı, tasarım aşaması ve keşifsel çalışma sonuçları olmak üzere, üç adımdan oluşmaktadır. Çalışmada uygulanan yönteme ilişkin adımlar Şekil 3'te özetlenmiştir.

Keşifsel çalışma hazırlığına ait adımlardan önceki bölümlerde bahsedilmiştir: Öncelikle, çalışma hedefi, "açık kaynak mobil yazılımların evrimleri boyunca kalitelerinin değişimini anlamak" olarak belirlenmiştir. Sonrasında yazılım evrimi konusunda literatürde sıklıkla karşılaşılan Lehman yasaları incelenmiş ve bu yasalardan çalışma hedefi ile ilgili olanları (artan karmaşıklık, sürekli büyüme, azalan kalite) seçilmiştir. Ardından, iç ve dış kalite analizinde ölçülmek üzere metrikler seçilmiş ve çalışma hedefine hizmet eden araştırma soruları aşağıdaki gibi belirlenmiştir:

- **AS 1.** Mobil uygulamaların evriminde iç kalite nasıl bir gelişim göstermektedir?
  - **AS 1.1.** Mobil uygulamalar evrimleri süresince karmaşıklık, boyut ve iç kalite açılarından nasıl bir gelişim göstermektedir?
  - **AS 1.2.** Lehman'ın artan karmaşıklık, sürekli büyüme ve azalan kalite yasaları mobil uygulamalar için geçerli midir?
- **AS 2.** Mobil uygulamaların evriminde dış kalite nasıl bir gelişim göstermektedir?
- **AS 3.** İç kalite ve dış kalite arasında anlamlı bir ilişki kurulabilir mi?

Keşifsel çalışma tasarımına ilişkin adımlar bu bölümde ve keşifsel çalışma sonuçları ise altıncı bölümde verilmiştir.



Şekil 3. Çalışmada izlenen adımlar

5.1. İncelenecek Açık Kaynak Kodlu Mobil Yazılımların Seçilmesi

Keşifsel çalışmayı gerçekleştirmek üzere mobil uygulama mağazalarından üç mobil uygulama aşağıdaki kriterlere göre seçilmiştir.

- *Açık kaynak kodlu uygulamalar.* Seçilecek uygulamaların kaynak kodlarına Github vb. veri depolarından erişim sağlanabilmelidir.
- *Uygulama platform benzerliği.* Çalışmada mobil uygulamaların sürümlerinin iç ve dış kalite özellikleri inceleneceğinden, platform farklılığından doğabilecek olası etkileri azaltmak adına, seçilecek uygulamalar benzer mobil platformlar için geliştirilmiş olmalıdır.
- *Nesneye yönelik dillerde geliştirilmiş uygulamalar.* Çalışmada incelenecek olan 'Yazılım Kalitesinin' analizi aşamasında C&K nesneye yönelik metrik seti kullanılacağı için uygulamalar nesneye yönelik bir programlama dilinde yazılmış olmalıdır.

- *Uygulama fonksiyonellik benzerliği.* Uygulamaların iç ve dış kalite gelişimi inceleneceğinden, farklı işlevselliğin çalışma üzerindeki olası etkilerini azaltmak adına, benzer fonksiyonelliğe sahip uygulamalar seçilmelidir.
- *Sürüm çeşitliliği.* Uygulamaların sürümleri boyunca iç ve dış kalite özelliklerinin gelişimi inceleneceğinden seçilen uygulamaların sürüm çeşitliliği olmalıdır.

Bu kriterleri sağlayan, şifre üretimi ve şifre saklama için kullanılabilen üç mobil uygulama olan Keepassdroid, UPM ve PasswdSafe uygulamaları, çalışmada analiz edilmek üzere seçilmiştir. Uygulamalar hakkında genel bilgiler Çizelge 8'de verilmiştir.

Çizelge 8. Seçilen mobil uygulamalara ait demografik bilgiler

				İlk sürüm			Son sürüm		
Uygulama	**Prog. dili**	**Kaynak kod adresi**	**Toplam sürüm sayısı**	**Sürüm No**	**Yayım tarihi**	**LOC**	**Sürüm No**	**Yayım tarihi**	**LOC**
Keepass-droid	Java	https://github.com/bpellin/keepassdroid	127	0.0.1	26.1.2009	4226	2.0.6.4	04.10.2016	33090
UPM	Java	https://github.com/adrian/upm-android	16	1.0	01.2.2010	1345	1.15	13.6.2016	4340
Passwd-Safe	Java	https://sourceforge.net/projects/passwdsafe/	92	1.0.0	03.1.2010	5728	6.10.0	29.7.2017	45096

5.2. Değerlendirme Yapılacak Boyutun Belirlenmesi

Seçilen açık kaynak kodlu mobil yazılımların evrimleri boyunca kalitelerinin hangi boyutlardan değerlendirileceği konusu önemlidir. Literatürde incelenen çalışmalar ışığında mobil ürünlerin, kod tabanlı (iç kalite) ve toplum tabanlı (dış kalite) olmak üzere iki boyutta değerlendirilmesine karar verilmiştir.

İç kalite evriminin incelenmesi için Keepassdroid ve PasswdSafe uygulamalarının son 5 yıldaki tüm sürümleri, UPM uygulamasının ise tüm (16) sürümleri; dış kalite evriminin incelenmesi için Keepassdroid'in 9 sürümü, UPM'in 5 sürümü ve PasswdSafe'in 33 sürümü kullanılmıştır (Çizelge 8).

Kod tabanlı ölçümde, açık kaynak yazılımlara erişim sağlanabilen veri depolarından (Github, Sourceforge vb.), uygulamaların kaynak kodlarına ulaşılmıştır. Toplum tabanlı ölçümde ise ihtiyaçlar doğrultusunda mobil uygulama mağazalarından analiz edilmek üzere metrikler toplanmıştır.

5.2.1. Kod Tabanlı (İç Kalite) Değerlendirme Tasarımı

5.2.1.1. Test Edilecek Hipotezler

Araştırma sorularına bağlı olarak ve Lehman'ın üç yasasını test etmek üzere, Çizelge 9'daki hipotezler oluşturulmuştur. Hedeflenen araştırma sorularına cevap bulabilmek ve hipotezlerin doğruluğunu araştırmak için C&K metrik seti [73] ve bazı geleneksel metrikler (örn. kod satır sayısı) kullanılmıştır.

Çizelge 9. Kod Tabanlı Boyutta Test Edilecek Hipotezler

Yasa 1. Artan Karmaşıklık (Lehman 2)	**Hipotez 1.a**: NOC zamanla artar.
	Hipotez 1.b: DIT zamanla artar.
	Hipotez 1.c: RFC zamanla artar.
	Hipotez 1.d: WMC zamanla artar.
	Hipotez 1.e: CBO zamanla artar.
Yasa 2. Sürekli Büyüme (Lehman 6)	**Hipotez 2.a:** LOC zamanla artar.
	Hipotez 2.b: Uygulamaların sınıf sayısı zamanla artar.
Yasa 3. Azalan Kalite (Lehman 7)	**Hipotez 3.a**: Uygulamaların uyumsuzluğu zamanla artar.
	Hipotez 3.b: Uygulamaların kararsızlığı zamanla artar.
	Hipotez 3.c: Uygulamaların taşınabilirliği zamanla azalır.
	Hipotez 3.d: Uygulamaların anlaşılabilirliği zamanla azalır.

5.2.1.2. Metrik Verilerinin Toplanması ve Analizi

Sourceforge ve Github veri depolarından üç mobil uygulamanın kaynak kodlarına erişim sağlanmıştır. Tüm kaynak kodlar, analiz edilmek üzere Understand aracına girdi verilmiş ve her bir uygulama için metrik değerleri çıkarılmıştır. Sonrasında her bir metrik değerine göre boyut, karmaşıklık ve iç kalitenin nasıl geliştiği analiz edilmiştir.

C&K metrikleri için farklı aralıklarda gerçek metrik değerleri elde edildiğinden ve bu değerleri aynı aralıkta tutmak amacıyla metrik değerleri, Min-Max Normalleştirme yöntemi (Eş. 2) kullanılarak normalize edilmiş ve tüm değerler 0 ile 1 arasındaki bir değere dönüştürülmüştür.

$$X_{inorm} = \frac{X_i - X_{min}}{X_{max} - X_{min}} \quad (2)$$

Keepassdroid, UPM ve PasswdSafe uygulamalarının seçilen sürümleri için, C&K metriklerine ilişkin normalize edilmiş değerler, sırasıyla Çizelge 10, Çizelge 11 ve Çizelge 12'de verilmiştir.

Çizelge 10. Keepassdroid uygulaması için normalize edilmiş C&K metrik değerleri

Metrik / Sürüm	NOC	DIT	RFC	WMC	CBO	LCOM
1.99.10	0,018565	0,337167	0,114479	0,049536	0,090476	0,31057
2.0	0,018522	0,336833	0,114609	0,049609	0,090611	0,311247
2.0.1	0,018565	0,339667	0,114457	0,048600	0,090032	0,30263
2.0.2	0,018565	0,340000	0,114457	0,049395	0,090000	0,309300
2.0.3	0,018541	0,340000	0,114348	0,049386	0,090135	0,308700
2.0.4	0,018391	0,338333	0,114239	0,049256	0,090000	0,307800
2.0.5	0,018304	0,336667	0,114130	0,049209	0,090270	0,307530
2.0.6	0,018261	0,336667	0,113696	0,049302	0,090811	0,308300
2.0.6.1	0,018217	0,336667	0,113696	0,049476	0,090827	0,308344
2.0.6.2	0,018174	0,336667	0,113478	0,048651	0,090811	0,303200
2.0.6.3	0,018174	0,336667	0,113478	0,049270	0,090750	0,309707
2.0.6.4	0,018174	0,333333	0,113043	0,049259	0,090811	0,309700

Çizelge 11. UPM uygulaması için normalize edilmiş C&K metrik değerleri

Metrik / Sürüm	NOC	DIT	RFC	WMC	CBO	LCOM
1.0	0,042500	0,250000	0,215714	0,211111	0,212500	0,267949
1.1	0,030000	0,245000	0,173571	0,078584	0,187692	0,259565
1.2	0,027500	0,240000	0,185357	0,081062	0,174286	0,260543
1.3	0,027500	0,240000	0,179655	0,081681	0,174286	0,259457
1.4	0,027500	0,240000	0,179655	0,081770	0,174286	0,259457
1.5	0,026750	0,230500	0,179679	0,078823	0,171429	0,249163
1.6	0,026750	0,231000	0,180786	0,079920	0,172500	0,252848
1.7	0,027000	0,231000	0,180786	0,080053	0,172500	0,252837
1.8	0,027000	0,231000	0,180786	0,080195	0,172500	0,252848
1.9	0,027000	0,231000	0,180786	0,080195	0,172571	0,252848
1.10	0,027000	0,231000	0,180786	0,080195	0,172500	0,252848
1.11	0,027000	0,231000	0,184607	0,082097	0,175857	0,252700
1.12	0,026923	0,230769	0,188462	0,083594	0,175824	0,252400
1.13	0,026923	0,230769	0,188462	0,084547	0,176923	0,242500
1.14	0,029605	0,256579	0,184211	0,082324	0,214575	0,233000
1.15	0,029221	0,266234	0,187848	0,082749	0,219780	0,223000

Çizelge 12. PasswdSafe uygulaması için normalize edilmiş C&K metrik değerleri

Metrik / Sürüm	NOC	DIT	RFC	WMC	CBO	LCOM
4.0.3	0,022164	0,148760	0,109120	0,053405	0,088048	0,331157
4.4.0	0,021725	0,156863	0,105805	0,052463	0,091158	0,334890
4.5.0	0,023604	0,161404	0,105222	0,052231	0,092218	0,336877
4.6.0	0,023522	0,160839	0,104854	0,052102	0,092254	0,335874
4.6.1	0,023522	0,160839	0,104854	0,052115	0,092254	0,335874
4.7.0	0,023197	0,163218	0,104010	0,051593	0,090981	0,331241
4.7.1	0,023197	0,163218	0,104010	0,051593	0,090981	0,331241
4.7.2	0,023118	0,371134	0,103812	0,051546	0,090669	0,331168
4.7.3	0,023118	0,371134	0,103812	0,051546	0,090669	0,331168
5.0.0	0,022727	0,369088	0,103158	0,051575	0,091043	0,330203
5.1.0	0,022651	0,368687	0,103007	0,051516	0,090736	0,332458
5.2.0	0,016332	0,380653	0,104680	0,050777	0,095026	0,365553
5.3.0	0,016041	0,381961	0,105073	0,051556	0,095611	0,368450
5.3.1	0,016041	0,381961	0,105073	0,051556	0,095611	0,368450
5.3.2	0,016746	0,386962	0,105847	0,051413	0,095755	0,368086
5.4.0	0,015907	0,386273	0,097148	0,046115	0,092544	0,339479
6.0.0	0,014898	0,311732	0,089171	0,042719	0,092823	0,330782
6.0.1	0,014898	0,311732	0,089214	0,042747	0,092871	0,330857
6.0.2	0,014898	0,311732	0,089257	0,042810	0,092967	0,331546
6.1.0	0,014870	0,311524	0,089476	0,043043	0,093509	0,331078
6.1.1	0,008970	0,307198	0,140649	0,039369	0,091886	0,306710
6.1.2	0,009075	0,308409	0,141787	0,039119	0,091828	0,307490
6.2.0	0,009134	0,308464	0,142429	0,039316	0,092731	0,309869
6.3.0	0,009123	0,308715	0,142439	0,039490	0,092777	0,310458
6.4.0	0,009041	0,308087	0,141339	0,039506	0,092887	0,306919
6.4.1	0,009044	0,308786	0,142163	0,039747	0,093222	0,308798
6.4.2	0,009044	0,308786	0,142253	0,039781	0,093189	0,308837
6.4.3	0,008977	0,308602	0,141965	0,039870	0,093433	0,308761
6.5.0	0,008767	0,305067	0,137642	0,039053	0,093835	0,300101
6.6.0	0,008797	0,304691	0,137044	0,039003	0,093751	0,298767
6.7.1	0,008384	0,294048	0,131924	0,037813	0,092979	0,286940
6.8.0	0,013533	0,294545	0,082151	0,040428	0,094851	0,298661
6.8.1	0,013533	0,294545	0,082379	0,040552	0,094893	0,299587

Çalışma kapsamında incelenen hipotezlerin geçerlilik sorgulaması, normalize edilmiş metrik değerleri kullanılarak yapılmıştır. Metriklerin nasıl yorumlanacağı konusunda birçok kural varken bunları doğrulayacak deneysel çalışmalar yetersizdir. Örneğin; NASA'nın Yazılım Güvence Teknoloji Merkezi, bir kodun diğer modüllerinden neden farklı olduğunu anlamak için, çıktılara bakarak elde edilen değerlerin karşılaştırılmasına dayanan yorumlama kurallarını önerir [139]. Bu çalışmada elde edilen metrik değerleri, literatürde yaygın bir şekilde kabul gören bu öneri kapsamında değerlendirilmiştir. Buna göre WMC, RFC, LCOM, CBO, DIT ve NOC metriklerinin değerlerinin düşük olması istenen durumdur.

5.2.2. Toplum Tabanlı (Dış Kalite) Değerlendirme Tasarımı

5.2.2.1. Test Edilecek Hipotezler

Mobil uygulamaların toplum tabanlı boyutta (dış kalite) evriminde gelişimini görmek amacıyla, DeLone ve McLean modeli kapsamında ele alınan başarı boyutları, Lehman'ın 'Azalan Kalite' yasası ışığında analiz edilmiştir. Bu analizler, Çizelge 13'te verilen hipotezler oluşturularak gerçekleştirilmiş ve hipotezlerin doğruluğunu araştırmak için toplum tabanlı metrikler kullanılmıştır.

Çizelge 13. Dış kalite değerlendirme için test edilecek hipotezler

Başarı Boyutu 1. Kullanıcı Memnuniyeti	**Hipotez 4.a**: Kullanıcı oyları (UR) zamanla azalır.
Başarı Boyutu 2. Kullanım	**Hipotez 5.a:** Eleştirici sayısı (NOR) zamanla azalır.
	Hipotez 5.b: Geribildirim sayısı (NOF) zamanla azalır.
Başarı Boyutu 3. Geliştirici Topluluğu Hizmet Kalitesi	**Hipotez 6.a**: Geliştirici sayısı (NOD) zaman içinde azalır.
	Hipotez 6.b: Değişiklik sayısı (CN) zamanla azalır.
	Hipotez 6.c: Sürümler arasındaki zaman (TBR) zamanla artar.
Dış Kalite	**Hipotez 7.** Uygulamaların dış kalitesi zamanla azalır.

Daha özel olarak öncelikle, toplum tabanlı metriklerin aralarındaki ilişkiler tespit edilerek ilişkilerin anlamlılığı ve yönü belirlenmiştir. Sonrasında ise anlamlı ilişkiler temel alınarak bir başarı metriği tanımı yapılmış ve uygulamaların evrimi boyunca bu metriğin gelişimi izlenmiştir.

5.2.2.2. Metrik Verilerinin Toplanması ve Analizi

Seçilen üç mobil uygulamanın sürümleri boyunca toplum tabanlı (dış kalite) özelliklerinin gelişimini değerlendirmek için, toplum tabanlı metriklere ilişkin veriler toplanmış ve analiz edilmiştir.

Uygulama marketlerinden elde edilen verilere göre Keepassdroid uygulaması için toplamda 33.971, UPM uygulaması için 4.053 ve PasswdSafe uygulaması için 3.083 oy kullanılmış ve çalışma kapsamında yorum içeren kullanıcı oyları analiz edilmiştir. Keepassdroid için 595, UPM için 202, PasswdSafe için ise 368 kullanıcı yorumu mevcut olup bu veriler temel alınarak her bir uygulama için sürüm seçimi gerçekleştirilmiştir.

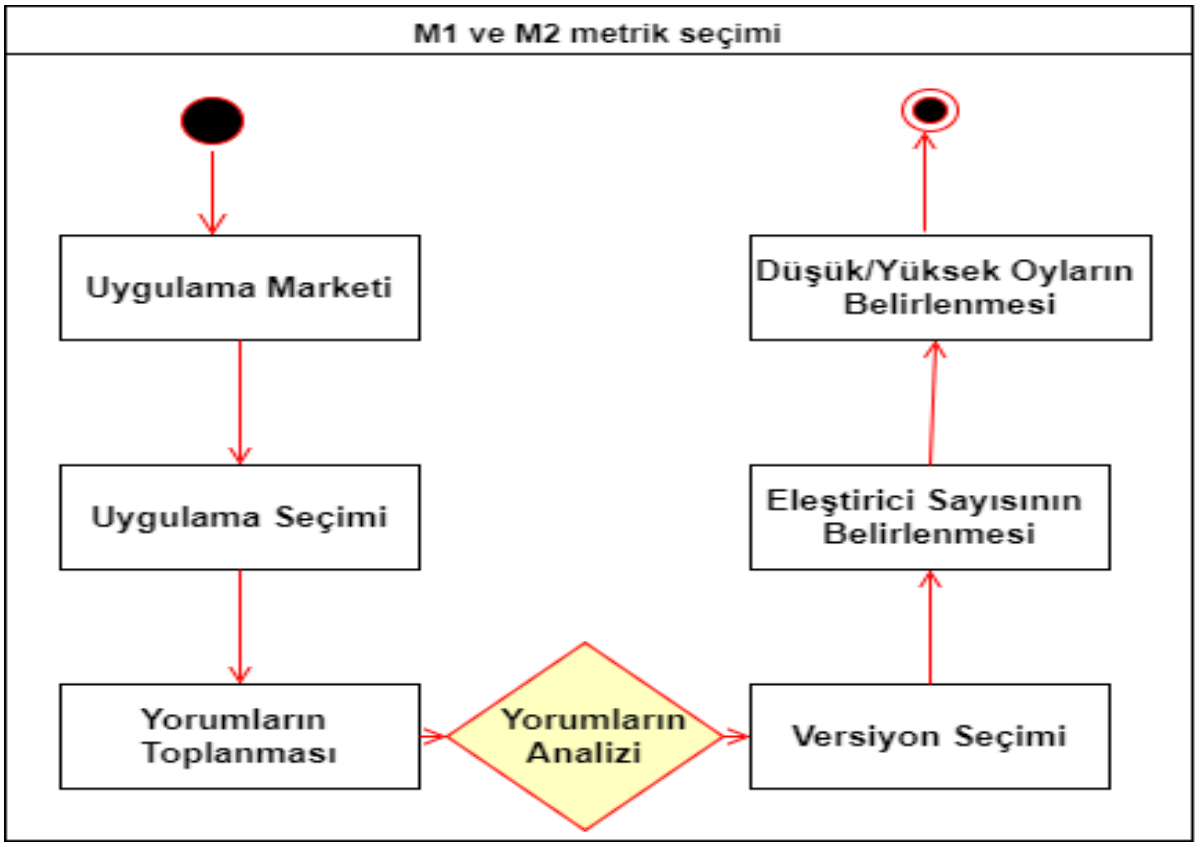
Kullanıcı oyları ve Eleştirici sayısı metriklerine ilişkin sürümlerin nasıl seçildiğini gösteren akış diyagramı Şekil 4'te verilmiştir.

M1- Metrik 1. Kullanıcı Oyları (İng. User Ratings - UR)

Bu metrik ile kullanıcıların uygulamadan ne kadar memnun kaldığı ve her bir sürüm için kaç kullanıcının kaç puan verdiği (1-5) incelenmiştir. Kullanıcı oyları metriği, 1'den 5'e kadar oylanan her bir sürüme ait medyan değerler ölçülerek hesaplanmıştır.

M2- Metrik 2. Eleştirici sayısı (İng. Number of Reviewers - NOR)

Keepassdroid için 595, UPM için 202 ve PasswdSafe için 368 kullanıcı yorumu incelenmiş ve her bir sürüm için eleştirici sayıları Çizelge 14, Çizelge 15 ve Çizelge 16'da verildiği gibi elde edilmiştir. Sürümler için yapılan yorumlar manuel olarak tespit edilmiş olup tablolardaki NOR sütunu her bir sürüme ait eleştirici sayısının, o sürümün piyasada kalma süresine oranı ile elde edilmiş normalize değerleri içermektedir.



Şekil 4. M1 ve M2 metrik seçimi

M3- Metrik 3. Geri bildirim sayısı (İng. Number of Feedbacks - NOF)

Kullanıcıların uygulamaya ne kadar geri dönüş yaptığı, uygulamanın kalitesi açısından önemli öngörüler sunabilmektedir. Bu sebeple her bir uygulama için

tartışma gruplarında yer alan, geliştirici-kullanıcı ya da kullanıcı-kullanıcı arasında gönderilen e-posta sayıları dikkate alınmıştır. Bu metrik, her bir sürüme ait geri bildirim sayısının, sürümün piyasada kalma süresine oranı ile hesaplanmıştır.

M4- Metrik 4. Katkıda bulunan geliştirici sayısı (İng. Number of Developers - NOD)

Bir yazılıma katkıda bulunan geliştiricilerin sayısına bakılarak o yazılımın kalitesi hakkında olumlu/olumsuz yorumlar yapılabilmektedir. Her bir sürüme katkıda bulunan geliştiricilerin sayısı hesaplanarak bu metrik değerlendirilmiştir.

M5- Metrik 5. Değişiklik sayısı (İng. Number of commits - CN)

Geliştirici sayısı metriğinin belirleyici bir metrik olabilmesi adına, her bir sürüm için yapılan değişiklikler (commit) gözlenmiştir.

M6- Metrik 6. Sürümler arasındaki zaman (İng. Time between Releases - TBR)

Bir sürümün ortaya çıkışından bir sonraki sürümün sunulmasına kadar geçen süredir.

Keepassdroid, UPM ve PasswdSafe uygulamalarının sürümleri için elde edilen toplum tabanlı metrik değerleri, sırasıyla Çizelge 14, Çizelge 15 ve Çizelge 16'da verilmiştir.

Çizelge 14. Keepassdroid için analiz edilen toplum-tabanlı metrik değerleri

KEEPASS-DROID	NOR	UR	NOF	NOD	CN	TBR
v2.0.2	0,5500	2,0000	0,0001	1,0000	2,0000	9,0000
v2.0.3	0,7980	19,0000	0,2100	1,0000	5,0000	233,0000
v2.0.4	0,6850	6,0000	0,3820	1,0000	9,0000	89,0000
v2.0.5	2,2220	3,5000	2,0000	3,0000	24,0000	9,0000
v2.0.6	3,0000	3,0000	1,0000	1,0000	23,0000	2,0000
v2.0.6.1	0,5855	11,0000	0,3600	1,0000	2,0000	222,0000
v2.0.6.2	0,6000	1,5000	0,0001	5,0000	30,0000	5,0000
v2.0.6.3	1,3333	2,0000	0,0010	1,0000	5,0000	3,0000
v2.0.6.4	0,4403	12,0000	0,2050	1,0000	2,0000	327,0000

Çizelge 15. UPM için analiz edilen toplum-tabanlı metrik değerleri

UPM	NOR	UR	NOF	NOD	CN	TBR
v1.11	0,4000	3,0000	2,5333	1,0000	7,0000	15,0000
v1.12	0,0198	2,0000	0,0148	1,0000	2,0000	202,0000
v1.13	0,2435	5,0000	0,6923	2,0000	10,0000	78,0000
v1.14	0,1058	13,0000	0,0309	1,0000	10,0000	1323,0000
v1.15	0,0701	5,0000	0,1337	1,0000	4,0000	471,0000

Çizelge 16. PasswdSafe için analiz edilen toplum-tabanlı metrik değerleri

PasswdSafe	NOR	UR	NOF	NOD	CN	TBR
v4.0.3	0,1666	1,0000	0,6666	1,0000	24,0000	6,0000
v4.4.0	0,1914	4,5000	0,0851	1,0000	6,0000	47,0000
v4.5.0	0,1923	1,0000	0,5064	1,0000	78,0000	156,0000
v4.6.0	4,0000	4,0000	0,0010	1,0000	13,0000	1,0000
v4.6.1	0,3200	4,0000	0,0800	1,0000	4,0000	25,0000
v4.7.0	3,0000	3,0000	0,0010	1,0000	10,0000	1,0000
v4.7.1	0,5714	4,0000	1,0714	1,0000	5,0000	14,0000
v4.7.2	0,7692	1,0000	0,8461	1,0000	4,0000	13,0000
v4.7.3	0,4482	5,0000	1,1724	1,0000	4,0000	29,0000
v5.0.0	0,6363	14,0000	0,2727	1,0000	24,0000	22,0000
v5.1.0	0,2500	20,0000	0,4375	1,0000	15,0000	160,0000
v5.2.0	0,1041	5,0000	0,2083	1,0000	147,0000	48,0000
v5.3.0	1,6666	2,5000	0,0003	1,0000	127,0000	3,0000
v5.3.1	0,7142	3,0000	0,2142	1,0000	5,0000	14,0000
v5.3.2	0,1833	5,5000	0,2666	1,0000	32,0000	60,0000
v5.4.0	0,1161	2,0000	0,1741	1,0000	45,0000	155,0000
v6.0.0	4,0000	2,0000	0,0010	2,0000	14,0000	1,0000
v6.0.1	3,5000	10,5000	0,6666	2,0000	15,0000	6,0000
v6.0.2	1,4615	3,0000	1,3846	2,0000	16,0000	13,0000
v6.1.0	1,6666	2,0000	0,2666	2,0000	16,0000	15,0000
v6.1.1	0,2926	1,0000	0,5365	2,0000	16,0000	41,0000
v6.1.2	1,0000	1,0000	0,0010	2,0000	16,0000	1,0000
v6.2.0	0,4545	2,0000	0,5909	2,0000	18,0000	22,0000
v6.3.0	0,2340	1,0000	0,0851	2,0000	17,0000	47,0000
v6.4.0	0,5294	1,0000	0,3529	2,0000	15,0000	17,0000
v6.4.1	0,1176	2,0000	0,2352	1,0000	19,0000	17,0000
v6.4.2	0,1875	2,0000	0,3281	1,0000	20,0000	64,0000
v6.4.3	0,3095	2,0000	0,4047	1,0000	15,0000	42,0000
v6.5.0	0,1578	2,0000	0,1929	1,0000	19,0000	57,0000
v6.6.0	0,1428	1,0000	0,2500	1,0000	19,0000	56,0000
v6.7.1	0,3703	1,0000	0,3703	1,0000	18,0000	27,0000
v6.8.0	4,0000	4,0000	0,0010	1,0000	15,0000	1,0000
v6.8.1	0,0857	1,0000	0,0571	1,0000	14,0000	105,0000

Toplamda üç farklı uygulama için elde edilen kırk yedi sürümün, altı farklı toplum tabanlı metrik değerleri hesaplanmış olup Çizelge 17'de bu metriklere ait tanımlayıcı istatistiksel değerlerinin bir özeti verilmiştir.

Çizelge 17. Her bir toplum-tabanlı metriğe ait tanımlayıcı istatistiksel değerler

	NOR	UR	NOF	NOD	CN	TBR
Minimum	0,0198	1,0000	0,0001	1,0000	2,0000	1,0000
Ortanca (Median)	0,4482	3,0000	0,2500	1,0000	15,0000	25,0000
Çeyrekler arası aralık	0,6410	3,5000	0,2170	1,0000	47,0000	185,0000
Maksimum	4,5500	20,0000	2,5333	5,0000	147,0000	1323,0000
Ortalama (Mean)	0,9977	4,3829	0,4097	1,3404	20,4255	90,9361
Standart Sapma	1,2608	4,5887	0,5218	0,7305	28,0510	206,5904
Çarpıklık	1,6700	2,0790	2,3357	3,2064	3,4735	4,9735
Basıklık	1,5710	3,9254	6,3763	13,2521	12,7235	28,4814

6. MOBİL UYGULAMALARIN EVRİMİ İÇİN KEŞİFSEL ÇALIŞMA SONUÇLARI

6.1. Kod Tabanlı (İç Kalite) Değerlendirme Sonuçları

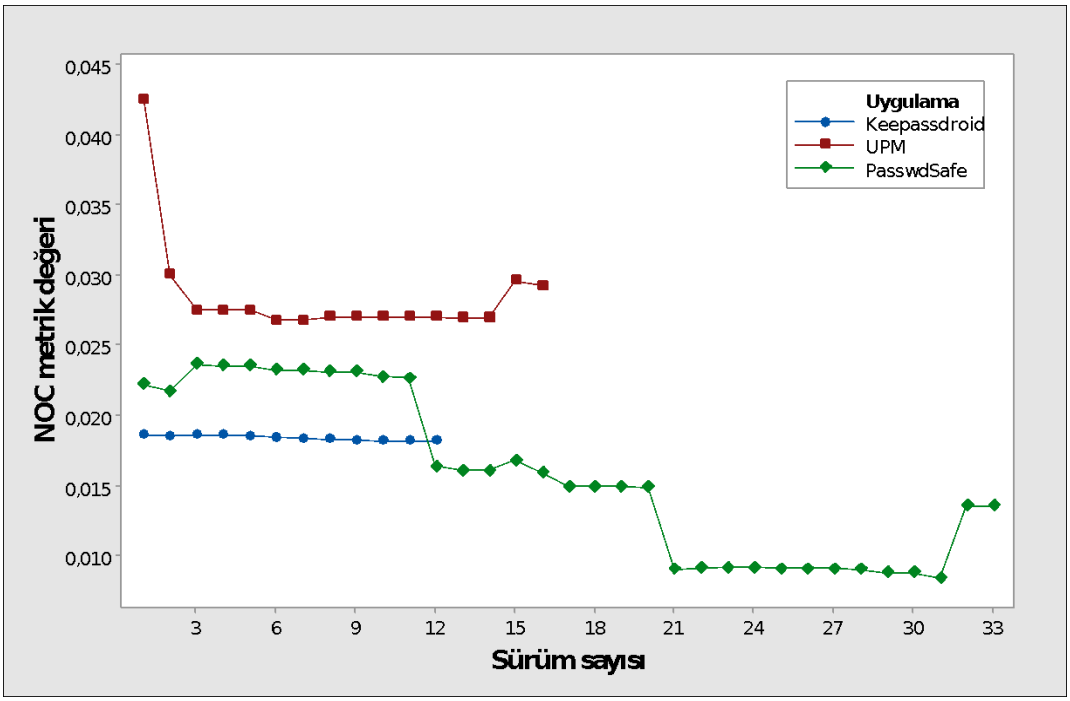
Birinci araştırma sorusuna (AS1) cevap bulabilmek için üç yasaya (artan karmaşıklık, sürekli büyüme, azalan kalite) ilişkin metrikler analiz edilerek uygulamaların sürümleri boyunca kod tabanlı gelişimleri incelenmiştir. Yapılan analizlerin detayları alt bölümlerde açıklanmıştır.

6.1.1. AS 1.1 için Elde Edilen Bulgular

AS 1.1 ile “mobil uygulamalar evrimleri süresince karmaşıklık, boyut ve iç kalite açılarından nasıl bir gelişme göstermektedir” sorusuna cevap aranmaktadır. Bunun için, bu bölümde mobil uygulamaların evrimleri süresince karmaşıklık, boyut ve iç kalite açılarından gelişimleri gözlenmiştir.

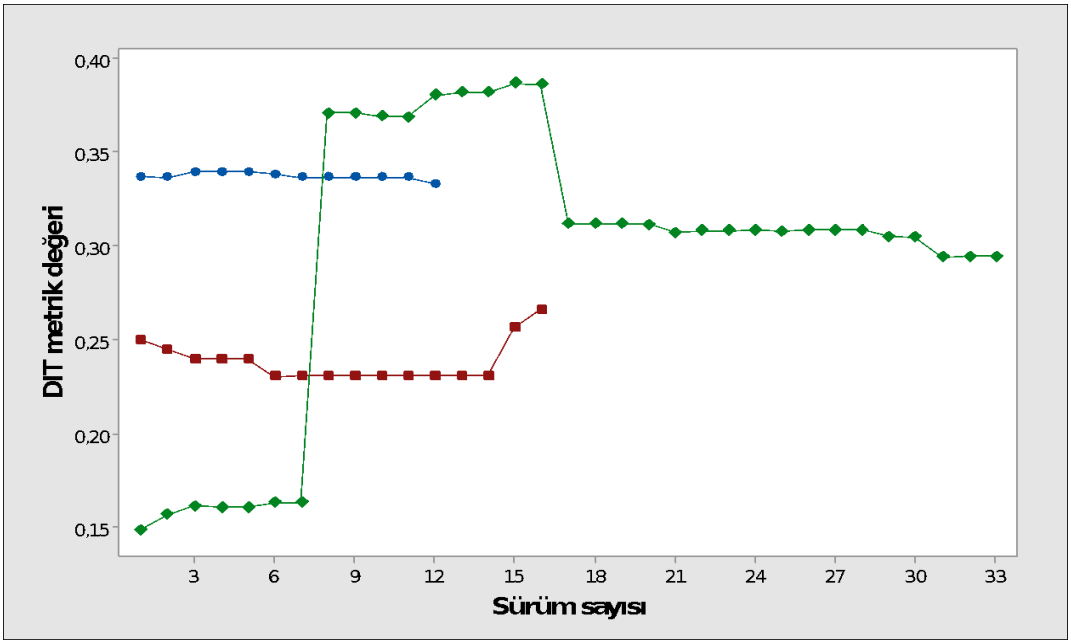
Karmaşıklık Değişimi

Yazılımlar gereksinimlerin artmasına bağlı olarak hızlı biçimde çok büyük boyutlara ulaşmaktadır**.** Yazılımın hızlı büyümesiyle beraber, önleyici bir davranış alınmadığı durumda, karmaşıklığının da artması şaşırtıcı değildir. Karmaşıklık analizi için Lehman, ilk olarak modül sayısının yüzdesini kullanmış, Godfrey ve Tu [120] yazılımın karmaşıklığını ve bağımlılığını ölçmede WMC, CBO ve RFC metriklerinin etkinliğini kanıtlarken Li ve arkadaşları [17], WMC metriğini karmaşıklığı ölçmede önermiştir. Li ve arkadaşları [17] ile Xie ve arkadaşları [18] çalışmalarında, karmaşıklığı ölçmede ortalama MCC karmaşıklığının kullanılmasını önermiştir. Ayrıca Newhook [62], yazılımların evriminde karmaşıklığı ölçmek için kod satır sayısını önermiştir. Bu çalışmada ise karmaşıklık ve bağımlılığı analiz etmek için NOC, DIT, RFC, WMC ve CBO metriklerinin ölçümü gerçekleştirilmiştir. Bu metriklerin uygulamaların sürümleri boyunca grafiksel değişimi Şekil 5-9 ile verilmiştir.



Şekil 5. Uygulamaların sürümleri boyunca NOC metrik değeri değişimi

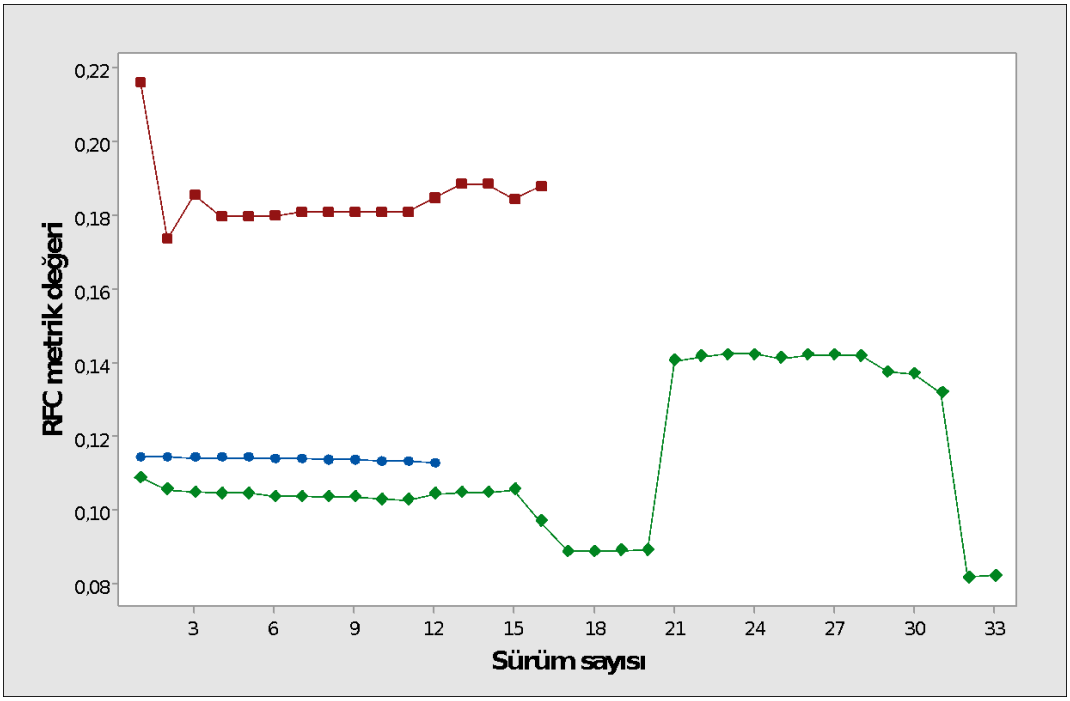
NOC metriğinin bulgularına bakıldığında, Keepassdroid ve PasswdSafe için değerlerin belirli bir dönem artıp daha sonra azaldığı, UPM için ise azalan bir grafik seyri gösterdiği görülmekte olup üç uygulama için de zamanla bu metrik değerinin son sürümlere doğru azaldığı gözlenmektedir (Şekil 5). Bu durum, yazılım karmaşıklığının olumlu yönde değiştiği şeklinde yorumlanmaktadır.



Şekil 6. Uygulamaların sürümleri boyunca DIT metrik değeri değişimi

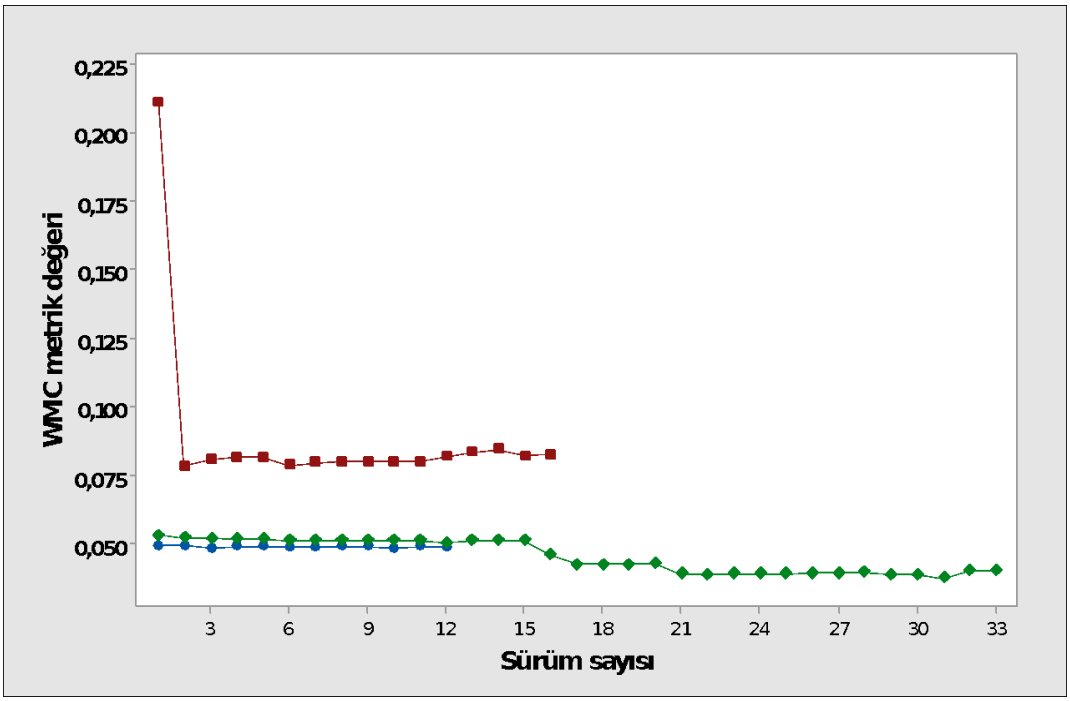
DIT metriğinin bulgularına göre, Keepassdroid ve PasswdSafe'de değerlerin belirli bir dönem artıp daha sonra azaldığı şeklinde benzer seyir gösterdiği, UPM için ise son sürümlere doğru artış gösterdiği gözlenmektedir (Şekil 6). Ağaçların derinliği, daha fazla metot ve sınıf içereceği için, tasarımı daha karmaşık hale getirmektedir; bu sebeple DIT değerinin düşük olması istenir.

Şekil 6'daki bulgular DIT metriğine göre incelendiğinde; Keepassdroid'in son sürümlerine doğru, PasswdSafe'in ise ilk sürümlerinde karmaşıklığın daha az olduğu gözlenmektedir.



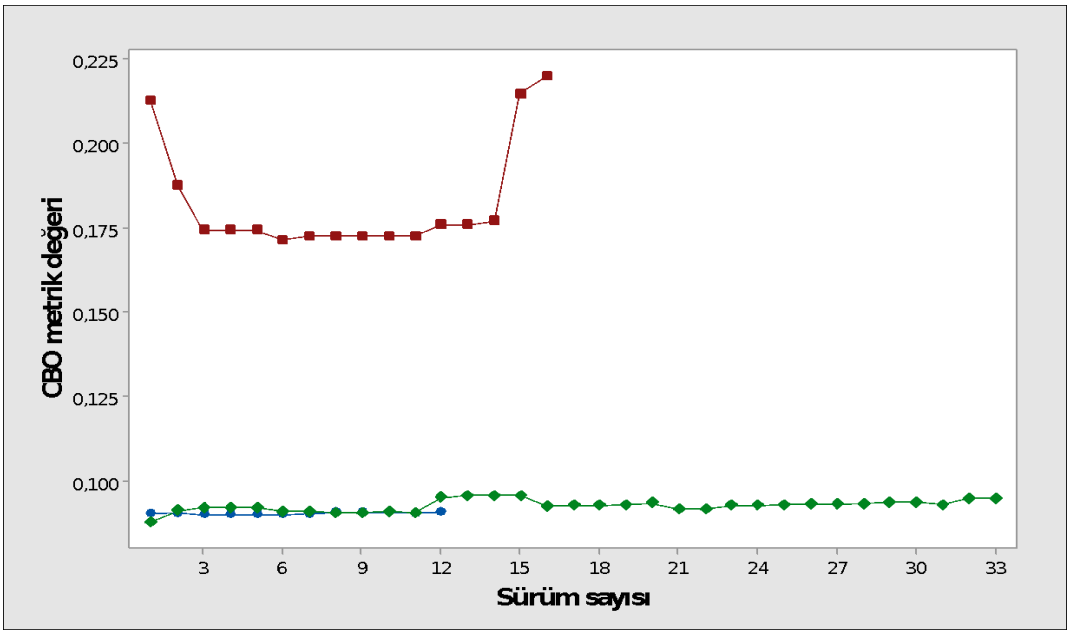
Şekil 7. Uygulamaların sürümleri boyunca RFC metrik değeri değişimi

Üç uygulama için de son sürümlere doğru RFC değeri azalan bir seyir gösterirken Şekil 7'deki RFC değeri bulgularına göre, sınıfın testi ve hata ayıklaması son sürümlere doğru daha az karmaşık hale gelmektedir. PasswdSafe uygulamasının 21. (6.1.1) sürümü ile birlikte sınıf sayısındaki çok hızlı bir büyüme, RFC değerinin büyük olmasına yol açmıştır.



Şekil 8. Uygulamaların sürümleri boyunca WMC metrik değeri değişimi

WMC değeri, üç uygulamanın son sürümlerine doğru azalmış olup bu durum yazılım karmaşıklığının olumlu yönde değiştiğini gösterir (Şekil 8). Keepassdroid ve PasswdSafe uygulamalarının bazı sürümleri için bu metriğin artış gösterip tekrar azaldığı görülmektedir. Bu durum ise o sürümler boyunca sınıf sayısının artmış olmasına rağmen yazılımdaki metot sayısında değişimin gözlenmemesi ile açıklanabilir.



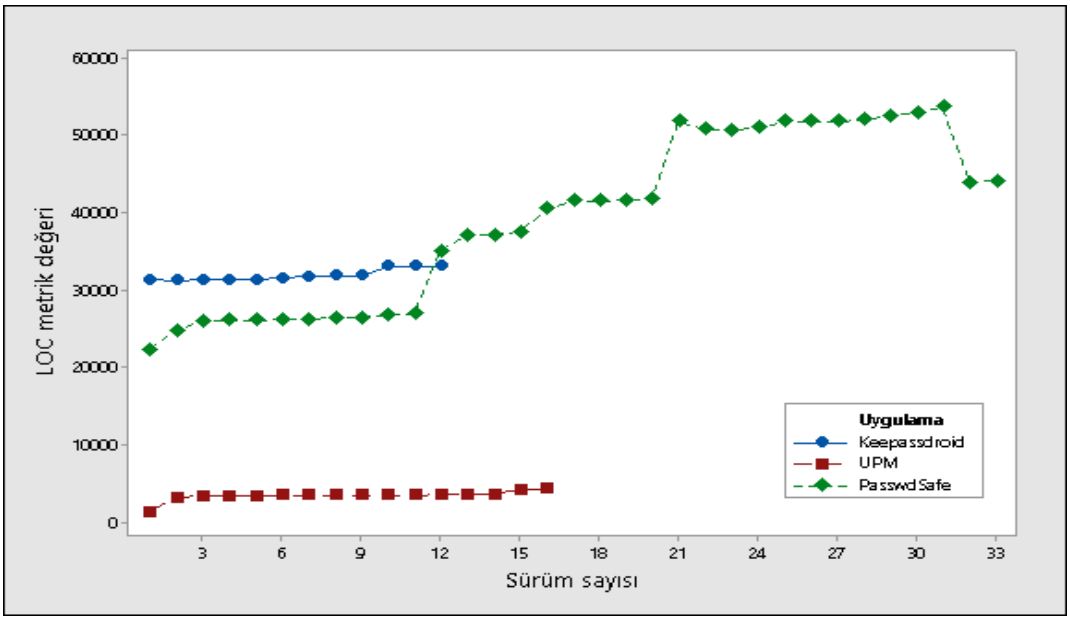
Şekil 9. Uygulamaların sürümleri boyunca CBO metrik değeri değişimi

CBO metriği bulgularına göre, incelenen üç yazılım için de değerlerin son sürümlere doğru arttığı görülmektedir. Bağımlılık metriğinin ilk sürümden sonra azalıp sınıf sayılarının ise artmasına bağlı olarak, CBO değerlerinin bazı sürümler boyunca arttığı gözlenmiştir (Şekil 9).

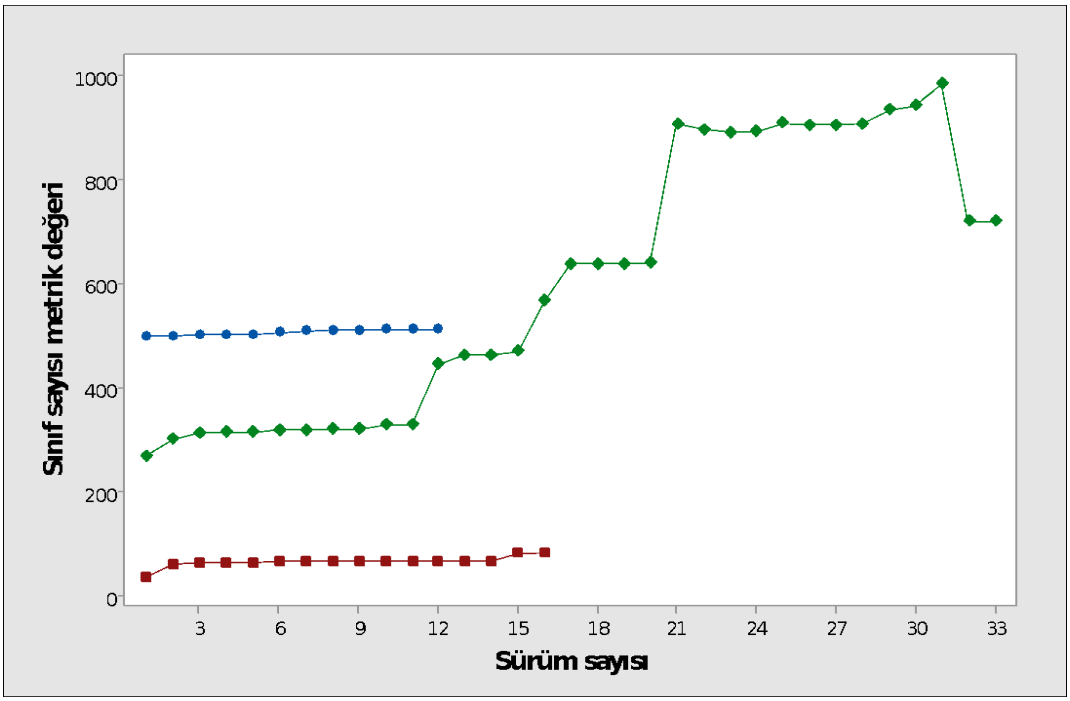
Sonuç olarak, karmaşıklığın genelde son sürümlere doğru azaldığı, bağımlılığın az da olsa bir artış gösterdiği gözlenmiştir.

Boyut Değişimi

Performans iyileştirmesi ve yeni platformlara uyum sağlaması gerekliliği nedeniyle gereksinimler artmaktadır, buna bağlı olarak yazılımın işlevi ve içeriği de sürekli olarak artmakta ve sonuçta sistemin boyutu artmaktadır. Daha önceki çalışmalarda [14, 18, 63, 64] araştırmacılar, sistem boyutu için çeşitli metrikler kullanmışlarsa da çoğunlukla uzunluk ve fonksiyonellik metriklerini tercih etmişlerdir. Lehman, yazılım boyutunu ölçmek için sistemin modül sayısını kullanırken Godfrey [120] ve Gonzalez [65], boyut metrikleri olarak kod satır sayısını, Israeli [66] ise Linux çekirdeğinin büyümesini tanımlamak için sistem çağrılarının sayısını, fonksiyon sayısını ve kod satır sayısını kullanmıştır. Nesneye yönelik programlamada yazılım geliştikçe, uygulamanın gereksinimlerinin artmasına bağlı olarak programın boyutunun ve sınıf sayısının artması beklenir. Şekil 10 ve Şekil 11'de sürekli büyüme yasasına ilişkin olarak analiz edilen metriklerin değişimi verilmiştir.



Şekil 10. Uygulamaların sürümleri boyunca kod satır sayısı değişimi



Şekil 11. Uygulamaların sürümleri boyunca sınıf sayısı değişimi

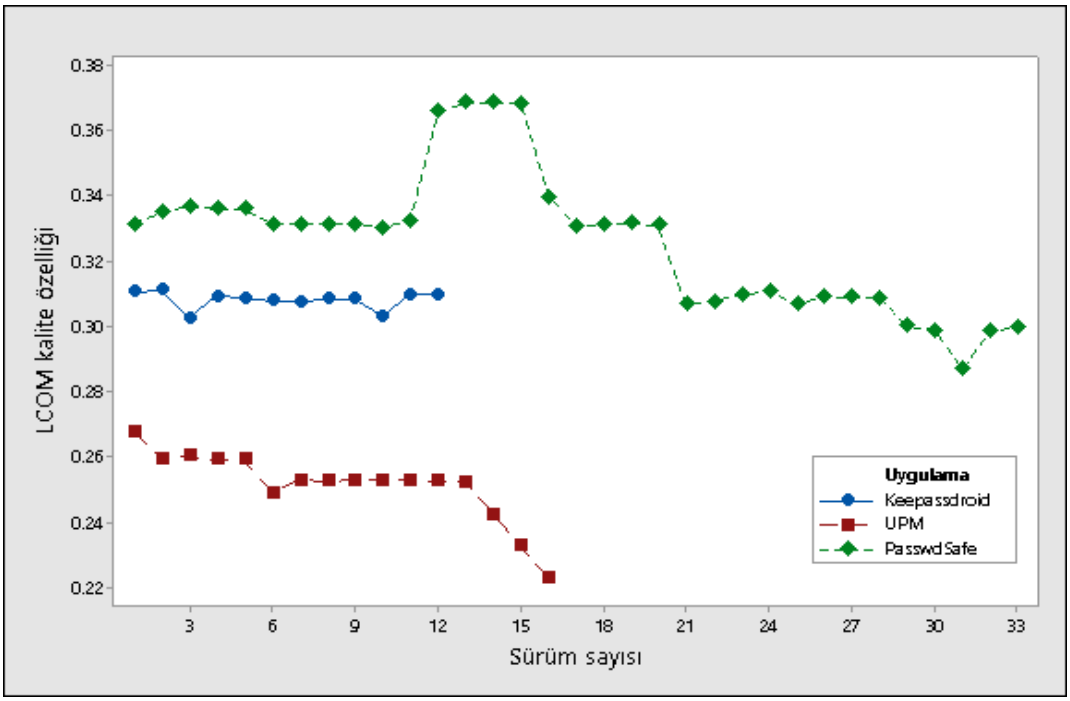
Yapılan analizlere göre tüm uygulamalar, evrimleri boyunca benzer bir artan eğilim göstermişlerdir. Şekil 10'da seçilen üç uygulamanın kod satır sayısına ve Şekil 11'de sınıf sayısına ait grafikler yer almaktadır. Grafik sonuçları, tüm uygulamaların evrim sürecinde büyümeye devam ettiklerini göstermektedir.

Kalite Değişimi

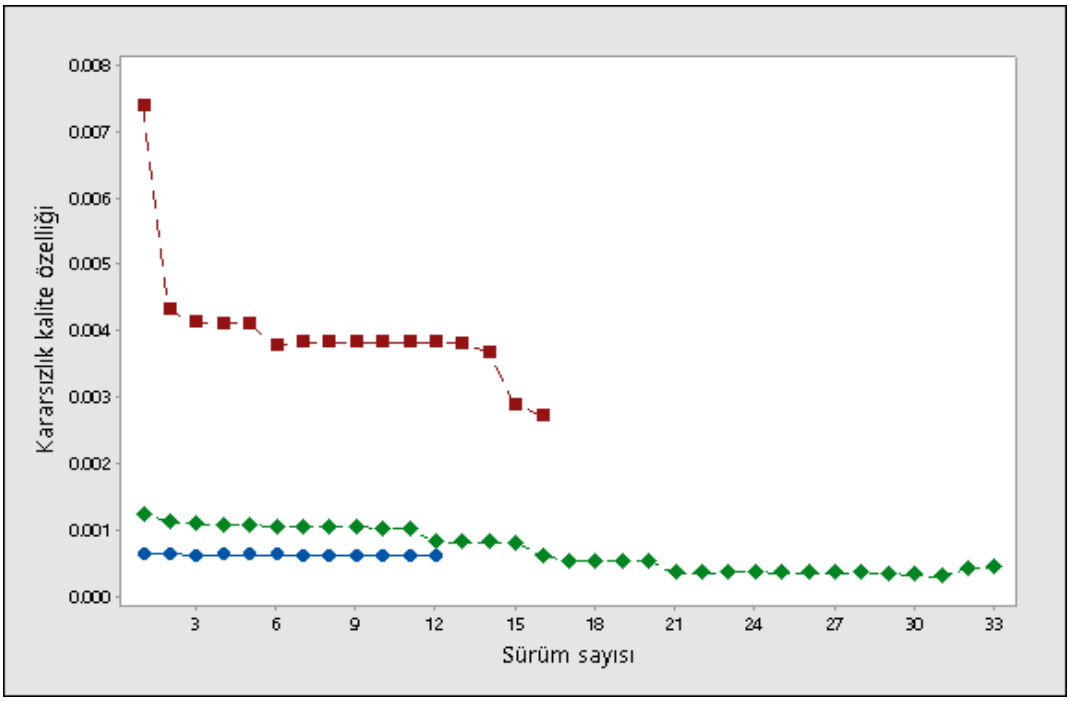
Önceki araştırmacılar [14, 17, 18] Lehman'ın azalan kalite yasasını çeşitli perspektiflerden açıklarken Lehman çalışmasında, belirli metrikler vermeyerek bu yasayı sadece "gömülü sistem, zamanın geçişi ile giderek daha uygunsuz hale gelmektedir" hipotezine dayanarak yorumlamıştır.

Bu bölümde kod tabanlı kalite incelenmiş ve bunun için, Gyimothy ve arkadaşlarının [63] yazılım kalitesini ölçmek için önerdiği, LCOM ve kararsızlık kalite özellikleri kullanılmıştır. Ayrıca, kullanım kolaylığı ve taşınabilirlik özellikleri nedeniyle popüler bir hal alan mobil uygulamaların, taşınabilirlik ve (kullanılabilirlik kalite özelliğinin alt-özelliği olan) anlaşılabilirlik kalite özellikleri de incelenmiştir. Metotlardaki uyum eksikliği bir sınıfın, iki veya daha fazla alt sınıfa ayrıldığını gösterir ve karmaşıklığı arttırır. Bir bileşenin taşınabilirliği bağımsızlığına, yani bileşenin dış destek olmadan işlevselliğini gerçekleştirme kabiliyetine bağlıdır. Anlaşılabilirlik ise programın bir kişi tarafından ne kadar anlaşılabildiği ile alakalıdır

ve özellikle yeniden kullanılabilirliğin önemli olduğu uygulamalar için kritik bir özelliktir. Bu bağlamda, kaliteli bir yazılım kolay anlaşılabilir ve kolay taşınabilir olmalıdır. Şekil 12-15'de sırasıyla uyumsuzluk, kararsızlık, anlaşılabilirlik ve taşınabilirlik kalite özelliklerinin sürümler boyunca değişimleri verilmiştir.



Şekil 12. Uygulamaların sürümleri boyunca uyumsuzluk değişimi

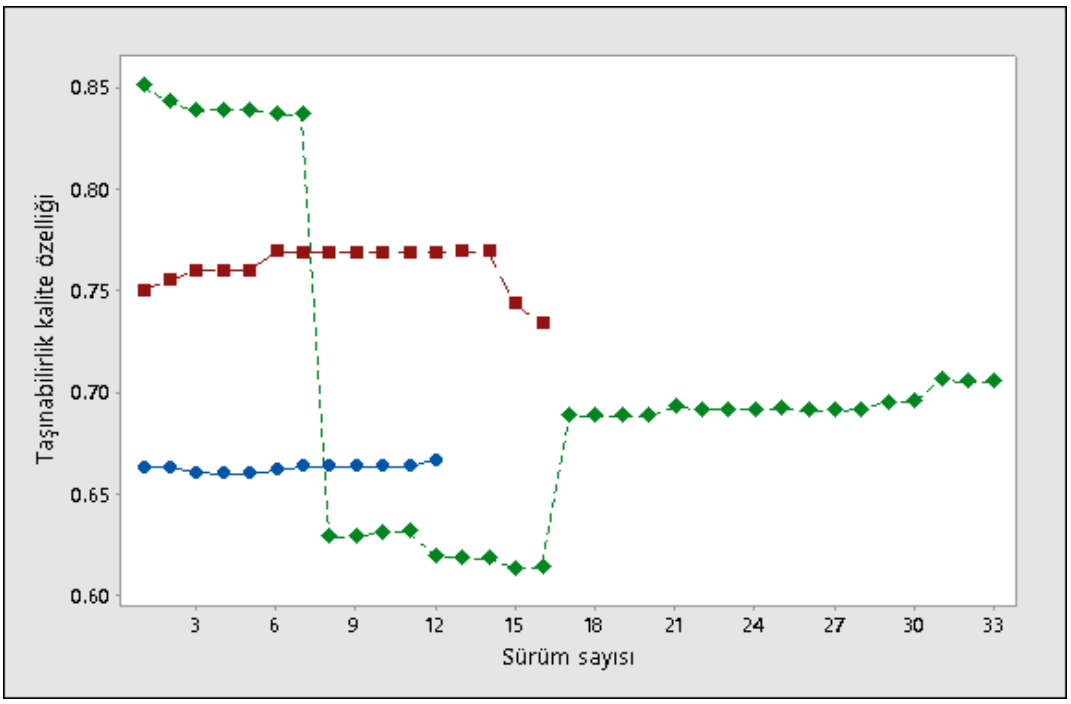


Şekil 13. Uygulamaların sürümleri boyunca kararsızlık değişimi

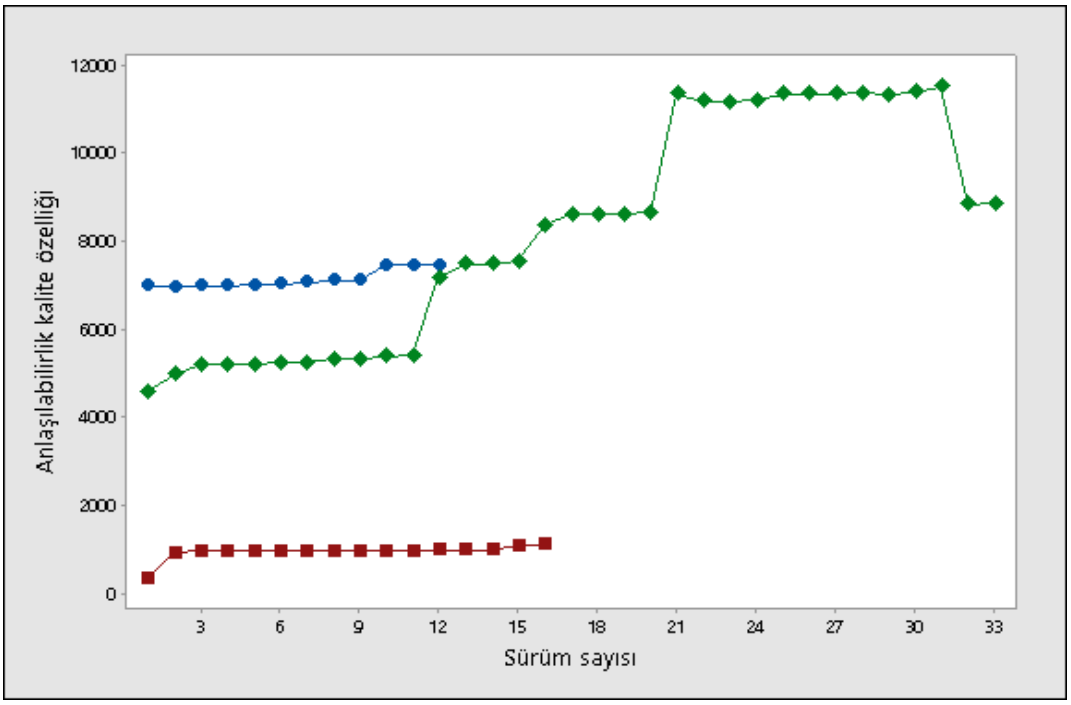
Grafik sonuçlarına göre, her üç uygulamanın belirli sürümlerinde, uyumsuzluk ve kararsızlık kalite özelliği değerlerinin azaldığı, daha sonra arttığı, sonra tekrar azaldığı gözlenmektedir (Şekil 12 ve Şekil 13). Genelde, ilk sürümler yayınlandıktan sonra, uygulamaların hataları sürekli olarak keşfedilmekte, yazılımdaki metot sayısı artmaya devam etmekte ve bu nedenle metotların uyumsuzluk yüzdesi azalmaktadır [12]. İlerleyen sürümlerde gereksinimlerin artmasına bağlı olarak uygulamanın boyutu büyümeye devam ederken uygulamanın yeni sürümlerine daha fazla özellik ve metot eklenmekte, uyumsuzluk tekrar yükselmekte ve zaman geçtikçe bu eğilim tekrarlanmaktadır.

Ayrıca, hazır kütüphanelerin ilavesi ile yazılımın kararsızlığı artmaktadır ve bu durum, çeşitli araştırmacılar tarafından birçok kez fark edilmiştir [140, 141]. Sürümlerin ilk ortaya çıkış sürecinde paketlerin daha durağan oldukları düşünülürse uygulamaların kararsızlığı başlangıçta azalırken sonrasında, özellikle hazır kütüphanelerin programa dâhil edilmeleriyle kararsızlığın arttığı gözlenmektedir.

Sonuç olarak, uyumsuzluk ve kararsızlık kalite özellikleri için uygulamaların kalitelerinin önce arttığı, sonra azalıp tekrar arttığı gözlenmiştir.



Şekil 14. Uygulamaların sürümleri boyunca taşınabilirlik değişimi



Şekil 15. Uygulamaların sürümleri boyunca anlaşılabilirlik değişimi

Şekil 14 ve 15'te sırasıyla taşınabilirlik ve anlaşılabilirlik kalite özelliklerine ilişkin grafikler verilmiştir. Anlaşılabilirlik, üç uygulama için de benzer bir artan eğilim gösterirken taşınabilirlik, Keepassdroid ve PasswdSafe uygulamaları için başlangıçta azalıp sonra artmış, UPM uygulaması için ise son sürümlere doğru azalan bir eğilim göstermiştir.

Yazılım mühendisliği pratiklerinin genel gelişimi düşünüldüğünde, yazılım kalitesinin eğilimine dair elde ettiğimiz bu gözlemler anlaşılabilir. Yazılım geliştirmenin ilk evrelerinde, sınırlı zaman ve zorlu piyasa rekabetinden dolayı ilk sürümlerin kalitesi genellikle düşük olurken uygulamaların hatalarının zamanla keşfedilmesi ve kodun yeniden yapılandırılması sonrasında, kalite giderek artmakta ve süreç bu şekilde devam etmektedir.

6.1.2. AS 1.2 için Elde Edilen Bulgular

AS 1.2 ile "Lehman'ın artan karmaşıklık, sürekli büyüme ve azalan kalite yasaları mobil uygulamalar için geçerli midir?" sorusuna cevap aranmaktadır. Bunun için, bu bölümde Lehman yasalarının mobil uygulamalar için geçerliliğini ve uygulamaların sürümleri boyunca metrik değerlerinin artışını (ya da azalışını) test etmek için, Mann-Kendall eğilim (trend) analizi [142] gerçekleştirilmiştir. Çizelge 18'de kod tabanlı değerlendirme ve Çizelge 19'da toplum tabanlı değerlendirme

için hipotezler, çalışma tasarımı aşamasında oluşturulmuş ve metrik verileri analiz edilerek hipotezlerin geçerlilikleri sınanmıştır.

Çizelge 18. Kod tabanlı değerlendirme için hipotezler ve test sonuçları

Yasa	Hipotez ID: Metrik		Keepass-droid	UPM	PasswdSafe
1	H_{1a}: NOC	Kendall's tau	-0,858	-0,258	-0,781
		S	-54,000	-29,000	-409,000
		p	1,000	0,909	1,000
	H_{1b}: DIT	Kendall's tau	-0,581	-0,245	-0,036
		S	-35,000	-27,000	-19,000
		p	0,994	0,896	0,616
	H_{1c}: RFC	Kendall's tau	-0,946	0,422	0,072
		S	-61,000	48,000	38,000
		p	1,000	0,014	0,278
	H_{1d}: WMC	Kendall's tau	-0,303	0,363	-0,743
		S	-20,000	43,000	-391,000
		p	0,924	0,026	1,000
	H_{1e}: CBO	Kendall's tau	0,406	0,182	0,460
		S	26,000	21,000	242,000
		p	0,036	0,0069	< 0,0001
2	H_{2a}: Kod satır sayısı	Kendall's tau	0,962	0,996	0,891
		S	63,000	119,000	469,000
		p	< 0,0001	< 0,0001	< 0,0001
	H_{2b}: Sınıf sayısı	Kendall's tau	0,930	0,822	0,868
		S	59,000	81,000	454,000
		p	< 0,0001	< 0,0001	< 0,0001
3	H_{3a}: Uyumsuzluk	Kendall's tau	-0,121	-0,781	-0,578
		S	-8,000	-91,000	-304,000
		p	0,727	1,000	1,000
	H_{3b}: Kararsızlık	Kendall's tau	-0,121	-0,373	-0,262
		S	-8,000	-44,000	-138,000
		p	0,727	0,977	0,984
	H_{3c}: Taşınabilirlik	Kendall's tau	0,581	0,245	0,036
		S	35,000	27,000	19,000
		p	0,006	0,104	0,384
	H_{3d}: Anlaşılabilirlik	Kendall's tau	0,939	0,767	0,861
		S	62,000	92,000	454,000
		p	< 0,0001	< 0,0001	< 0,0001

*Gri alanlar geçerlenen hipotezleri işaret etmektedir.

Çizelge 19. Toplum tabanlı değerlendirme için hipotezler ve test sonuçları

Yasa	Hipotez ID: Metrik		Keepass-droid	UPM	PasswdSafe
3	H_{4a}: UR	Kendall's tau	-0,085	0,527	-0,307
		S	-3,000	5,000	-149,000
		p	0,623	0,103	0,991
	H_{5a}NOR	Kendall's tau	-0,444	-0,400	-0,118
		S	-16,000	-4,000	-62,000
		p	0,962	0,883	0,832
	H_{5b}: NOF	Kendall's tau	-0145	-0,200	-0,082
		S	-5,000	-2,000	-43,000
		p	0,703	0,758	0,748
	H_{6a}: NOD	Kendall's tau	0,129	0,000	0,213
		S	3,000	0,000	72,000
		p	0,332	0,500	0,073
	H_{6b}:CN	Kendall's tau	0,059	0,105	0,140
		S	2,000	1,000	72,000
		p	0,415	0,400	0,131
	H_{6c}: TBR	Kendall's tau	-0,085	0,600	0,146
		S	-3,000	6,000	76,000
		p	0,623	0,117	0,119
	H_{7a}: Dış Kalite	Kendall's tau	-0,500	0,200	-0,237
		S	-18,000	2,000	-125,000
		p	0,978	0,408	0,974

Bu testle, değişkenlerin zaman içindeki gidişatı ölçülmektedir. Bu test, verilerin belirli bir dağılıma uyma şartını aramayan ve zamana bağlı eğilim analizlerini önemseyen çalışmalar için kullanışlıdır. Belirlenen p anlamlılık seviyesinde hesaplanan Mann-Kendall testinin istatistik değeri olan S değeri pozitif ise artan, negatif ise azalan bir eğilim mevcuttur. Kendall tau ise korelasyon katsayısıdır. Bu değer 1'e ne kadar yakınsa o derece yüksek oranda bir ilişkinin varlığından söz edilir.

Çizelge 18 verilerine göre, 33 veri grubundan 11'inin geçerliliği sağlanırken üç uygulama için de geçerliliği kanıtlanan (p<.05 ve S>0) hipotez sayısı 3 olarak karşımıza çıkmıştır (H_{1e}, H_{2a}, H_{2b}). Dış kalite ile ilişkili Çizelge 19 verilerine göre, 21

veri grubundan hiçbirinin geçerliliği sağlanmamıştır. Buna göre, Lehman'ın 'karmaşıklığın artması' yasası üzerine oluşturulan hipotezlerden sadece birinin (H_{1e}) ve 'sürekli büyüme' yasasına ilişkin oluşturulan hipotezlerin hepsinin (H_{2a}, H_{2b}) geçerliliği sağlanırken, 'azalan kalite' yasasına ilişkin kurulan kod tabanlı ve toplum tabanlı hipotezlerden hiçbiri geçerli bulunmamıştır.

6.1.3. Kod Tabanlı Bulgulara Yönelik Değerlendirme

Önerilen yöntem, ISO 25010 [70] kalite özelliklerini değerlendirmek için kaynak kodu özniteliklerini kullandığından bu kaynak kodu öznitelikleri, belirli bir kalite seviyesi sergileyen sistemlerin gerektirdiği eğilimleri belirtmelidir. Bu eğilimler yazılım evrim yasaları ile ilgilidir [15]. Öncelikle, yeni özellikler eklenmesine ve ek gereksinimlerin karşılanmasına bağlı olarak kalite özelliklerinin bir sürümden diğerine geçerken iyileşmesi beklenmektedir. Bu sayede bir yazılım sistemi sürekli büyür ve değişikliklerle uyumlu ve kullanımda tatmin edici bir hal alır. Bu durum, yazılım sistemlerinin sürekli değişimi ve büyümesi yasalarıyla uyumludur [15]. Önceki çalışmalarda [17, 18] bu yasaya ilişkin oluşturulan hipotezler doğrulanmış olup bu bağlamda, çalışmamızda oluşturulan hipotezler de kod satır sayısı ve sınıf sayısı metrikleri (H_{2a} ve H_{2b}) ile doğrulanmıştır ve üç uygulamanın da ilk sürümden son sürümlere doğru büyüdüğü gözlenmiştir.

Bununla birlikte, yeni özellikler eklendiğinde ve kaynak kodu yeni sınıflar ve yöntemler ekleyerek genişletildiğinde, yazılımın daha karmaşık hale gelmesi nedeniyle bakım yapılabilirliğin başlangıçta düşmesi beklenmektedir. Bu durum, 'karmaşıklığın artması' yasasına göre olağandır. Yapılan çalışmalarda [11, 17, 18] ve bu çalışmada, Lehman'ın artan karmaşıklık yasasının test edilen tüm uygulamalar için doğrulanmadığı gözlenmiştir (H_{1e} hariç). Bu bağlamda, çalışmada test edilen beş hipotez (H_{1a}, H_{1b}, H_{1c}, H_{1d}, H_{1e}) sonucuyla uyumlu olarak belli sürümler boyunca, karmaşıklığın artıp daha sonra azaldığı çizilen grafiklerde (Şekil 5-9) gözlenmiştir. Bu bulgu ise bazı sürümler boyunca, geliştiricilerin tasarım ve kod karmaşıklığını azaltmaya odaklandığı sonucunu doğurmaktadır.

Azalan kalite yasasına ilişkin bulgular, [17, 18] çalışmalarıyla paralellik gösterirken sonuçlar, hipotezlerin tüm uygulamalar için doğrulanmadığını göstermiştir. Oluşturulan hipotezlerin (H_{3a}ve H_{3b}) tersine, son sürümlere doğru metotların uyumsuzluğu ve kararsızlığın azaldığı gözlenmiş ve bu durum, kalitenin bu

açılardan arttığı şeklinde yorumlanmıştır. Genellikle anlaşılabilirlik, her bir sistem bileşeni için kullanıcının bileşenin amacının ne olduğunu, belirli görevler ve kullanım koşulları için nasıl kullanılabileceğini anlamasına olanak tanıyarak sağlanır. Son sürümlere doğru ürünlerin işlevselliği artmakta ve ürün sistemi daha karmaşık bir hale gelmektedir. Buna bağlı olarak Lehman'ın 'azalan kalite' yasasında da belirtildiği gibi, zaman geçtikçe yazılımın kalitesi azalmaktadır [15]. Anlaşılabilirliğin, üç uygulamanın 61 sürümünde incelenmesi sonucunda ve bu yasanın tersine, doğrusal olmasa da zamanla artış gösterdiği ve uygulamaların son sürümlerinin daha anlaşılabilir olduğu tespit edilmiştir. Buna göre sürümler boyunca kalite, anlaşılabilirlik açısından artış göstermiştir. Taşınabilirlik ise yazılımın farklı çalışma ortamlarına uyum sağlayabilme yeteneği olarak tanımlanır [69]. Yazılım, öncelikle geliştirildiği platform ve altyapıya kolaylıkla yüklenebilmelidir. Sonrasında yazılımın farklı platformlarda değişiklik yapılmadan çalışılabiliyor olması, taşınabilirliğinin yüksek olduğu anlamına gelir. Karmaşıklık metriklerinin analizi sonucunda taşınabilirlik kalite özelliğinin son sürümlere doğru artması beklenmektedir. Keepassdroid ve PasswdSafe için bu artış gözlenirken UPM uygulaması için artıp azalan bir eğilim görülmüştür. Ayrıca Çizelge 19'da yer alan toplum tabanlı boyutta oluşturulan hipotezlerin hiçbiri geçerliliğini sağlayamamıştır ve 'azalan kalite' yasası ne iç kalite ne de dış kalite açısından geçerlenmiştir.

6.1.4. Keşifsel Çalışma Sonuçlarının Korelasyon Analizi ile Sınanması

Yasa 1 (Artan karmaşıklık) ve Yasa 2 (Sürekli büyüme)'ye ilişkin metriklerin (NOC, DIT, RFC, WMC, CBO, LOC, sınıf sayısı), Yasa 3'e (Azalan kalite) ilişkin kalite özellikleri ile aralarındaki ilişkiler tespit edilmiş ve Çizelge 20'de gösterilmiştir. Her üç uygulama için anlamlılık değerine göre kategorize edilen Çizelge 20 verileri, keşifsel çalışmamızın sonuçlarını istatistiksel yöntemle doğrulamak amacıyla elde edilmiştir.

İlişkileri tespit etmek ve saptanan ilişkilerin anlamlılığını sınamak için, Spearman korelasyon analizi gerçekleştirilmiştir. Korelasyon analizinde p değeri (<,05) ilişkinin anlamlılığına ilişkin gözlemler yapılmasını sağlarken korelasyon katsayısının (r_s) 1'e yaklaşması iki değişken arasındaki ilişkinin gücünün artması anlamına gelir. Katsayı değeri pozitifse ilişkinin de pozitif, negatifse ilişkinin de negatif olduğundan söz edilebilir.

Çizelge 20. Yasa 1 ve Yasa 2'ye ilişkin metriklerin kalite özellikleri ile ilişkisi

Uygulama	Kalite özellikleri	Test sonuçları	NOC	DIT	RFC	WMC	CBO	LOC	Sınıf sayısı
Keepassdroid	Uyumsuzluk	r_s	0,064	-0,025	0,239	0,811	0,134	-0,917	-0,921
		p	0,849	0,945	0,455	0,002	0,681	0,05	0,041
	Kararsızlık	r_s	0,170	0,004	0,211	0,655	0,081	-0,179	-0,194
		p	0,598	1,000	0,511	0,014	0,807	0,580	0,546
	Taşınabilirlik	r_s	-0,869	-1,000	-0,769	-0,171	-0,830	0,750	0,751
		p	0,000	< 0,0001	0,005	0,597	0,002	0,007	0,007
	Anlaşılabilirlik	r_s	-0,915	-0,737	-0,984	-0,692	-0,600	0,998	0,989
		p	< 0,0001	0,009	< 0,0001	0,021	0,044	< 0,0001	< 0,0001
UPM	Uyumsuzluk	r_s	0,392	0,280	-0,294	0,623	-0,140	-0,884	-0,905
		p	0,134	0,293	0,268	0,005	0,604	< 0,0001	< 0,0001
	Kararsızlık	r_s	0,063	-0,014	-0,165	0, 930	-0,371	-0,510	-0,552
		p	0,818	0,964	0,540	0,025	0,157	0,046	0,029
	Taşınabilirlik	r_s	-0,928	-1,000	-0, 991	-0,227	-0,648	0,179	0,178
		p	< 0,0001	< 0,0001	0,005	0,398	0,008	0,506	0,509
	Anlaşılabilirlik	r_s	-0,049	0,049	0,302	-0,575	-0,725	0,849	0,651
		p	0,858	0,861	0,255	0,022	0,010	< 0,0001	0,008
PasswdSafe	Uyumsuzluk	r_s	0,759	0,340	-0,258	0,769	-0,090	-0,746	-0,756
		p	< 0,0001	0,053	0,147	< 0,0001	0,617	< 0,0001	< 0,0001
	Kararsızlık	r_s	0,145	-0,168	0,204	0,788	0,165	-0,231	-0,224
		p	0,419	0,349	0,254	0,010	0,359	0,195	0,210
	Taşınabilirlik	r_s	-0,884	-1,000	0,602	0,098	-0,610	-0,115	-0,108
		p	0,026	< 0,0001	0,015	0,587	0,030	0,523	0,549
	Anlaşılabilirlik	r_s	-0,957	0,118	0,422	-0,951	-0,509	0,993	0,993
		p	< 0,0001	0,512	0,015	< 0,0001	0,003	< 0,0001	< 0,0001

*Gri alanlar anlamlı bir ilişkiye sahip verileri işaret etmektedir.

Test sonuçları, Yasa 1 ve Yasa 2'ye ilişkin metriklerin (NOC, DIT, RFC, WMC, CBO, LOC, sınıf sayısı) genel olarak, kalite özellikleri ile anlamlı bir ilişkiye sahip oldukları sonucunu ortaya çıkarmıştır. Bu da korelasyon analizi sonuçlarının keşifsel çalışma sonuçlarını doğrular nitelikte olduğunu göstermektedir. Bu sebeple, bulgulara ilişkin değerlendirme yaparken Yasa 1 ve Yasa 2'ye ilişkin metriklerin artış/azalışına göre yazılımların kalitesi yorumlanmıştır (Çizelge 21).

Üç uygulamanın 61 sürümüne ait veriler üzerine yapılan korelasyon analizi sonucunda elde edilen ilişkiler Çizelge 21'de özetlenmiştir. Bu bulgulara göre;

- NOC metriği ile taşınabilirlik kalite özelliği arasında negatif ilişki,
- DIT metriği ile taşınabilirlik kalite özelliği arasında negatif ilişki,
- RFC metriği ile taşınabilirlik kalite özelliği arasında negatif ilişki,
- WMC metriği ile uyumsuzluk ve kararsızlık kalite özellikleri arasında pozitif, anlaşılabilirlik kalite özelliği arasında negatif ilişki,

- CBO metriği ile taşınabilirlik ve anlaşılabilirlik kalite özellikleri arasında negatif ilişki,
- LOC metriği ile uyumsuzluk kalite özelliği arasında negatif, anlaşılabilirlik kalite özelliği arasında pozitif ilişki,
- Sınıf sayısı ile uyumsuzluk kalite özelliği arasında negatif, anlaşılabilirlik kalite özelliği arasında pozitif ve anlamlı bir ilişki olduğu gözlenmiştir.

Çizelge 21. Test sonuçlarına göre kalite özellikleri ve tasarım metrikleri arasındaki ilişki

Kalite özellikleri	**Tasarım Metrikleri**						
	NOC	DIT	RFC	WMC	CBO	LOC	Sınıf sayısı
Uyumsuzluk				+		-	-
Kararsızlık				+			
Taşınabilirlik	-	-	-		-		
Anlaşılabilirlik				-	-	+	+

+ : Pozitif ilişki, **- :** Negatif ilişki anlamına gelir.

Çalışmada gerçekleştirilen korelasyon analizi sonuçlarına göre her bir hipoteze ilişkin kullanılan metriklerin, yazılım kalitesinin çalışma kapsamında incelenen özellikleriyle anlamlı bir ilişkiye sahip olduğu görülmüştür. Örneğin, çalışmanın ikinci bölümünde yer alan Çizelge 4'te karmaşıklık metriklerinin ISO 25010 kalite özellikleri ile ilişkili olduğundan ve bu metriklerin, yazılım kalitesi hakkında öngörüde bulunmayı sağladığından bahsedilmişti. Benzer şekilde korelasyon analizi, keşifsel çalışmanın sonuçlarını doğrular nitelikte olup karmaşıklık metriklerinin uyumsuzluk, kararsızlık, taşınabilirlik ve anlaşılabilirlik kalite özellikleri ile anlamlı bir ilişkiye sahip olduklarını göstermiştir.

Benzer yorumlar boyut metrikleri (LOC, sınıf sayısı) için de yapılabilir. Öyle ki kod satır sayısı ve sınıf sayısı metriklerinin çalışmada hedeflenen kalite özellikleri ile anlamlı bir ilişkiye sahip oldukları gözlenmiştir.

Elbette, bazı kalite özellikleri ile bunlarla doğrudan ilişkili metriklerin aralarında yüksek bir ilişki çıkması beklenen bir durumdur. Örnek olarak, taşınabilirlik kalite özelliği, DIT metriği kullanılarak hesaplanmıştır (1-DIT). Bu sebeple, taşınabilirlik kalite özelliği ile DIT metriği arasındaki ilişkinin anlamlı bulunacağını öngörmek kaçınılmazdır. Bununla birlikte çalışmada, taşınabilirlik kalite özelliğinin sadece

DIT metriği ile değil, 7 farklı metrik ile de aralarındaki ilişki analiz edilmiş ve CBO metriğinin taşınabilirlik kalite özelliği ile ilişkili olduğu gözlenmiştir.

6.2. Toplum Tabanlı (Dış Kalite) Değerlendirme Sonuçları

İkinci araştırma sorumuz, mobil uygulamaların sürümleri boyunca toplum-tabanlı açıdan dış kalite (market başarısı) gelişimini mevcut metriklere göre analiz edebilmeyi; diğer bir deyişle, bir ürünün potansiyel başarısını, toplum tabanlı metrikler üzerinden anlamayı amaçlamaktadır. Bunun için öncelikle kullanıcı memnuniyeti, kullanım ve geliştirici topluluğu hizmet kalite boyutları açısından her bir metrik için ayrı ayrı, dış kalite gelişimi izlenmiştir. Sonrasında ise üç başarı boyutunu adresleyen metrikleri beraber ele alarak hesaplanan başarı indeksi değerine göre, üç uygulama için toplum tabanlı (dış) kalite gelişiminin ne yönde değiştiği analiz edilmiştir.

6.2.1. Dış kaliteyi niteleyen Başarı İndeksinin İstatistiksel Yöntemler Kullanılarak Oluşturulması

Bu tez çalışmasında, toplum tabanlı boyutta dış kalitenin bir göstergesi olarak market başarısı, tek bir açıdan değil de birden çok açıdan değerlendirilmiştir. Bunun için Çizelge 7'de verilen 3 boyut ve ilişkili 6 metrikten oluşan modelimiz temel alınarak öncelikle, bir başarı indeksi oluşturulmuştur. Bu başarı indeksi oluşturulurken her bir metriğin diğer metriklerle anlamlı bir ilişkisinin olma durumuna göre incelemeler yapılmıştır. Bunun için her bir metriğin diğer metriklerle arasındaki ilişkinin yönü tespit edildikten sonra bu ilişkiler gözetilerek bir başarı indeksi (İng. Success index – SI) denklemi oluşturulmuştur. Sonrasında bu başarı indeksinin her bir metrik ile olan anlamlılık durumuna göre anlamsız ilişkiye sahip metrikler denklemden çıkartılmış ve çalışmanın devamında kullanılmak üzere yeni bir denklem elde edilmiştir.

Öncelikle, çalışma kapsamında verileri analiz edebilmek ve yorumlayabilmek için, iki veya daha fazla değişkenin istatistiksel bağımlılıklarının tanımlanması gerekmektedir. Öyle ki, iki veya daha fazla değişkenin aynı özelliği tanımladığı tespit edilirse ölçümlerde gereksiz değişkenlerin göz ardı edilmesi işi kolaylaştıracaktır. Bunu yapabilmek için, analiz edilen değişkenler arasındaki ilişki katsayısı (İng.Correlation index) hesaplanacak ve yüksek oranda ilişkili değişkenlerin aynı özelliği tanımladığı varsayılacaktır.

İki veya daha fazla değişken arasındaki ilişkiyi hesaplamak için çeşitli yöntemler mevcuttur [20, 142, 143]. Ancak bu çalışmada kullanılan veriler farklı ölçeklere ait oldukları için, kullanılacak yönteme karar verebilmek adına öncelikle verilerin normal dağılım gösterip göstermediklerini tespit edebilmek amacıyla tüm değişkenler üzerinde normallik testi gerçekleştirilmiştir.

Normallik Testi: Normallik testi verilerin normal dağılıma uyup uymadığını görmek amacıyla yapılan bir testtir. Parametrik testlerin yapılması için sağlanması gerekli koşullardan biridir. En bilinen testler Kolmogorov-Smirnov ve Shapiro-Wilk testidir [144, 145]. Shapiro-Wilk-W testi normallik varsayımını sınayan en güçlü testtir ve Shapiro-Wilk testinin örneklem sayısının 3 ile 5000 arasında olduğunda kullanılması önerilir [146]. Bu sebeple çalışmada Shapiro- Wilk testi kullanılmıştır. Normallik testi sonuçları Çizelge 22'de verilmiştir.

Çizelge 22. Normallik testi sonuçları

Metrik\Test	Shapiro-Wilk Test Sonuçları
Eleştirici sayısı	< 0,0001
Kullanıcı oyları	< 0,0001
Geribildirim sayısı	< 0,0001
Katkıda bulunan geliştirici sayısı	< 0,0001
Değişiklik sayısı	< 0,0001
Sürümler arasındaki zaman	<0,0001

Test sonucunda tüm metrik değerleri için $p<0.05$ olduğundan verilerin normal dağılım göstermediği tespit edilmiştir. Kısacası, sürüm bazında ve her bir metrik değeri için oluşturulan veri setleri normal dağılım göstermemektedir.

Analiz edilen verilerin normal dağılmadığını göz önünde bulundurarak, metrikler arasındaki ilişkinin anlamlılığını ve yönünü tespit edebilmek amacıyla iki parametrik olmayan değişken arasındaki korelasyonu hesaplayan bir indeks olan Spearman korelasyon katsayısı kullanılmıştır. Spearman korelasyon analizi [147], iki değişken arasındaki ilişkiyi veya bir değişkenin iki veya daha çok değişken ile olan ilişkisini test etmek, varsa bu ilişkinin derecesini ölçmek için kullanılan istatistiksel bir yöntemdir. Amaç, bir değişken değeri değiştiğinde, diğer değişkenin ne yönde değişeceğini görmektir. Bu analizde r_s, iki veya daha fazla değişken arasındaki korelasyon katsayısını ifade eder. r_s katsayısı hesaplandıktan sonra, veri şu şekilde yorumlanabilmektedir:

- 0 < | r_s | <= 0.25: zayıf ilişki
- 0.25 < | r_s | <= 0.75: orta derecede ilişki
- | r_s | > 0.75: güçlü ilişki

Keepassdroid, UPM ve PasswdSafe uygulamalarının toplamda kırk yedi sürümüne ait tüm toplum-tabanlı metrikler bu teste tabi tutulmuştur. Böylece her bir metriğin birbiri ile olan ilişkisi analiz edilip sonrasında bu metrikler ilişki derecelerine göre başarı indeksi oluşturmada kullanılmıştır. Toplum-tabanlı metriklere ilişkin tanımlayıcı istatistiksel değerler Çizelge 23'te verilmiştir.

Çizelge 23. Tanımlayıcı istatistiksel değerleri

Metrik	Gözlem sayısı	Minimum	Maximum	Mean	Std. deviation
NOR	47	0,020	4,550	0,998	1,261
UR	47	1,000	20,000	4,383	4,589
NOF	47	0,000	2,533	0,410	0,522
NOD	47	1,000	5,000	1,340	0,731
CN	47	2,000	147,000	20,426	28,051
TBR	47	1,000	1323,000	90,936	206,590

Çizelge 24. Metrikler arasındaki Spearman Korelasyon Katsayısı (r_s)

Metrik	NOR	UR	NOF	NOD	CN	TBR
NOR	**1**	0,113	-0,048	0,291	-0,149	-0,732
UR	0,113	**1**	0,094	-0,257	-0,303	0,281
NOF	-0,048	0,094	**1**	0,132	0,100	0,117
NOD	0,291	-0,257	0,132	**1**	0,227	-0,304
CN	-0,149	-0,303	0,100	0,227	**1**	-0,155
TBR	-0,732	0,281	0,117	-0,304	-0,155	**1**

Çizelge 25. Metrikler arasındaki ilişkinin anlamlılık derecesi

Metrik	NOR	UR	NOF	NOD	CN	TBR
NOR	**0**	0,449	0,747	0,048	0,316	< 0,0001
UR	0,449	**0**	0,529	0,082	0,039	0,056
NOF	0,747	0,529	**0**	0,375	0,501	0,431
NOD	0,048	0,082	0,375	**0**	0,125	0,038
CN	0,316	0,039	0,501	0,125	**0**	0,298
TBR	< 0,0001	0,056	0,431	0,038	0,298	**0**

*Gri renkli alanlar ilgili hücredeki metriklerin anlamlı bir ilişkiye sahip olduklarını göstermektedir (p<.05)

Toplum tabanlı metrikler için, Spearman korelasyon indeksleri Çizelge 24'te ve metrikler arasındaki ilişkinin anlamlılık derecesi ise Çizelge 25'te verilmiştir. Eleştirici sayısı metriğinin; kullanıcı oyları metriği ile arasında pozitif ve düşük derecede bir ilişki, sürümler arasındaki zaman metriği ile arasında ise negatif ve orta derecede bir ilişki gözlenmiştir. Buna göre sürümler arasındaki zaman arttıkça projenin canlılığının azalmasına bağlı olarak eleştirici sayısında bir azalma görülmektedir. Sürümler arasındaki zaman arttıkça projenin canlılığının azaldığı, katkıda bulunan geliştirici sayısı ve değişiklik sayısı metriklerinin bu metriğe bağlı olarak negatif ilişkili olduğu gözlenmiştir. Ayrıca, geliştirici sayısı ve e-posta sayısı metrikleri arasında düşük derecede ve pozitif bir ilişki, geliştirici sayısı ve değişiklik sayısı metrikleri arasında ise düşük derecede ve pozitif bir ilişki gözlenmiştir. Metrikler arasında anlamlı ilişkiler tespit edilmiş ve başarı indeksini oluştururken dikkate alınmıştır. Buna göre, başarı indeksi (SI) şöyle hesaplanabilir:

SI = (NOR*UR*NOF*NOD*) / (CN*TBR) (3)

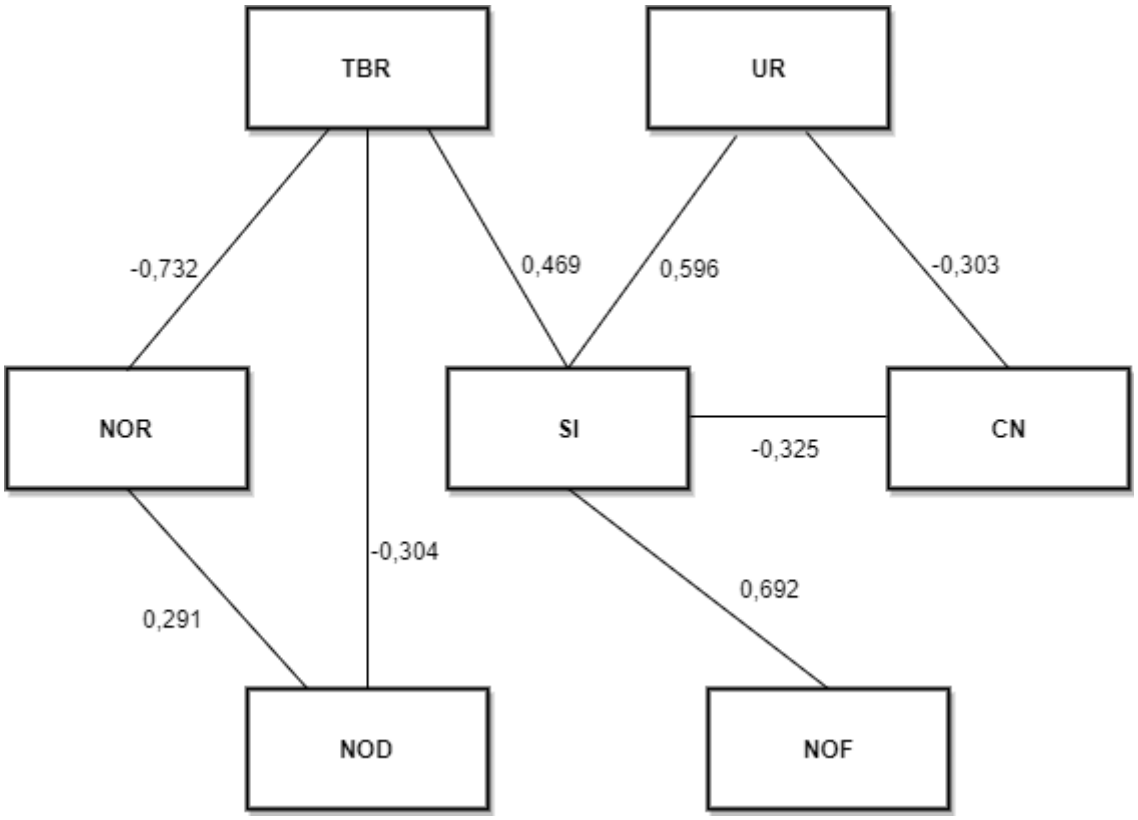
Başarı indeksi formülü oluşturulduktan sonra, hesaplanan indeks değeri ile her bir metrik değeri arasındaki ilişki, Spearman korelasyon analizi ile tespit edilmiştir. Analiz sonuçları Çizelge 26'da verilmiştir.

Çizelge 26. Başarı indeksi ile her bir metrik arasındaki ilişkinin Spearman korelasyon analizi ile tespiti

			NOR	UR	NOF	NOD	CN	TBR	SI
Spearman Korelasyon Analizi	NOR	r_s	**1**	0,113	-0,048	0,291	-0,149	-0,732	-0,025
		p		0,449	0,747	0,048	0,316	< 0,0001	0,867
		N	47	47	47	47	47	47	47
	UR	r_s	0,113	**1**	0,094	-0,257	-0,303	0,281	0,596
		p.	0,449		0,529	0,082	0,039	0,056	< 0,0001
		N	47	47	47	47	47	47	47
	NOF	r_s	-0,048	0,094	**1**	0,132	0,100	0,117	0,692
		p	0,747	0,529		0,375	0,501	0,431	< 0,0001
		N	47	47	47	47	47	47	47
	NOD	r_s	0,291	-0,257	0,132	**1**	0,227	-0,304	0,044
		p.	0,048	0,082	0,375		0,125	0,038	0,769
		N	47	47	47	47	47	47	47
	CN	r_s	-0,149	-0,303	0,100	0,227	**1**	-0,155	-0,325
		p.	0,316	0,039	0,501	0,125		0,298	0,026
		N	47	47	47	47	47	47	47
	TBR	r_s	-0,732	0,281	0,117	-0,304	-0,155	**1**	0,469
		p	< 0,0001	0,056	0,431	0,038	0,298		0,001
		N	47	47	47	47	47	47	47
	SI	r_s	-0,025	0,596	0,692	0,044	-0,325	0,469	**1**
		p.	0,867	< 0,0001	< 0,0001	0,769	0,026	0,001	
		N	47	47	47	47	47	47	47

*Gri renkli alanlar ilgili hücredeki metriklerin anlamlı bir ilişkiye sahip olduklarını göstermektedir ($p<.05$)

Başarı indeksi ile her bir metrik arasındaki korelasyon katsayısının -0,025 ile 0,692 arasında olduğu gözlenmiştir. Başarı indeksi ile kullanıcı oyları, geri bildirim sayısı ve sürümler arasındaki zaman metrikleri arasında anlamlı, pozitif ve orta derecede bir ilişki gözlenirken bu indeks ile değişiklik sayısı metriği arasında negatif ve orta derecede bir ilişki görülmüştür. İstatistiksel analizler sonucunda tespit edilen her bir metriğin market başarısı ile arasındaki ilişkinin grafiksel gösterimi Şekil 16'da verilmiştir. Son durumda, başarı indeksi ile kullanıcı oyları, geri bildirim sayısı ve sürümler arasındaki zaman metrikleri arasında anlamlı ve pozitif, başarı indeksi ile değişiklik sayısı arasında anlamlı ve negatif bir ilişki gözlenmiştir. Başarı indeksi ile diğer metrikler arasında anlamlı bir ilişki tespit edilememiştir.



Şekil 16. Anlamlı ilişkiye sahip metrikler ve başarı indeksi arasındaki ilişki

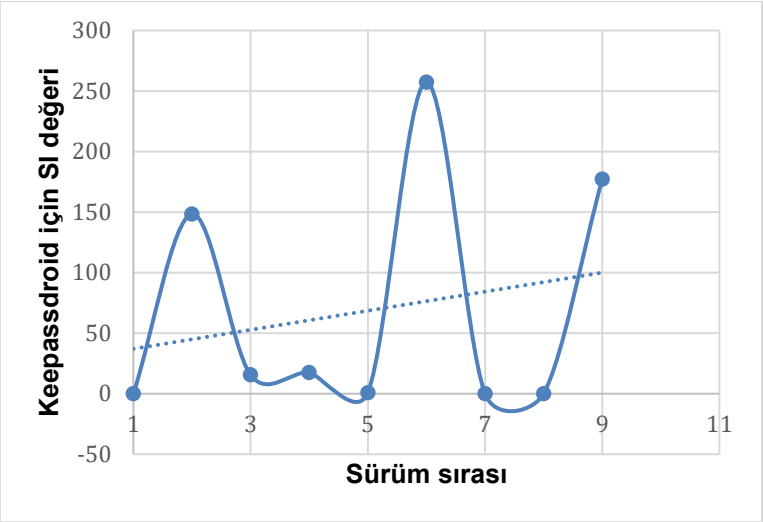
Anlamlılık ilişkilerine göre seçilen kullanıcı oyları, geri bildirim sayısı, sürümler arasındaki zaman ve değişiklik sayısı metriklerinin ilişki yönleri dikkate alınarak oluşturulan başarı indeksi denklemi şu şekildedir:

$$SI = UR*NOF*TBR / CN \quad (4)$$

Bundan sonraki adımda oluşturulan başarı indeksi formülüne göre, seçilen uygulamaların sürümleri boyunca dış kalite gelişimi izlenmiştir.

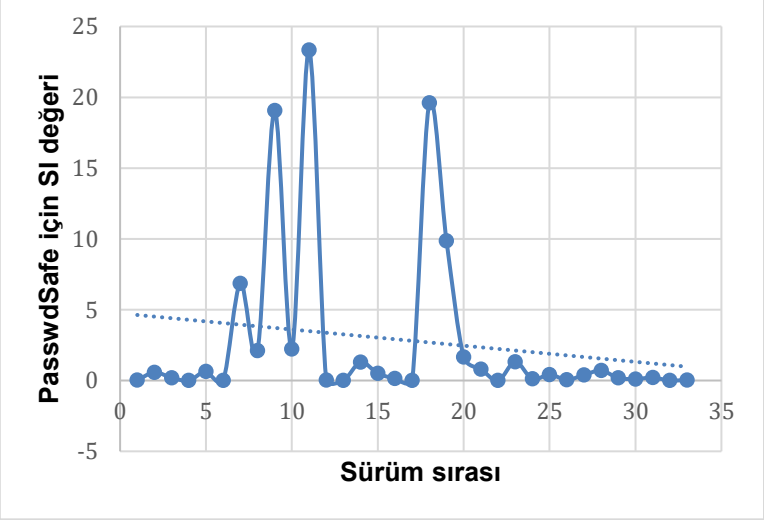
6.2.2. Toplum Tabanlı (Dış Kalite) Değişimi

Lehman'ın azalan kalite yasasını çeşitli perspektiflerden inceleyen araştırmacılar, yazılımın iç ve dış kalite açılarından değerlendirilmesi gerektiğini belirtmişlerdir. Birinci araştırma sorusu iç kalite değişimini izlemeyi hedeflerken ikinci araştırma sorusu dış kalite değişimini görmeyi hedeflemiştir. Başarı indeksinin oluşturulan denklemine göre (Eş. 4), üç uygulamanın sürümleri boyunca değişimi sırasıyla Şekil 17, Şekil 18 ve Şekil 19'da verilmiştir:



Şekil 17. Keepassdroid için sürümler boyunca SI değişimi

Şekil 18. UPM için sürümler boyunca SI değişimi



Şekil 19. PasswdSafe için sürümler boyunca SI değişimi

Keepassdroid'in 9, UPM'in 5 ve PasswdSafe uygulamasının 33 sürümü boyunca yapılan gözlemler sonucunda başarı indeksinin; Keepassdroid ve UPM uygulamaları için zamanla arttığı, PasswdSafe uygulaması için ise zamanla azaldığı görülmüştür.

Ayrıca, başarı indeksinin uygulamaların bazı sürümlerde artıp maksimum seviyede bir değere yükseldiği ve sonrasında bir düşüş gösterdiği gözlenmiştir.

6.3. Kod Tabanlı ve Toplum Tabanlı (İç ve Dış) Kalite Arasındaki İlişkinin İzlenmesi

Üçüncü araştırma sorusuna cevap bulabilmek amacıyla bu bölümde uygulamaların sürümleri boyunca değerlendirilen kod tabanlı (iç) ve toplum tabanlı (dış) kalite değişimleri arasında anlamlı bir ilişkinin var olup olmadığı analiz edilmiştir. Bunun için öncelikle, kod tabanlı değerlendirmede ölçülen Uyumsuzluk, Kararsızlık, Taşınabilirlik ve Anlaşılabilirlik kalite özellikleri ile dış kalite arasındaki ilişki tespit edilmiş ve test sonuçları Çizelge 27'de verilmiştir. Sonrasında ise daha detaylı bir çerçevede ilişki analizi gerçekleştirebilmek için iç kalite özellikleri ile dış kalite ölçümünde kullanılan toplum tabanlı metrikler arasındaki ilişki ayrı ayrı Spearman korelasyon analizi ile tespit edilmiş ve analiz sonuçları Çizelge 28'de gösterilmiştir.

Çizelge 27. İç kalite özellikleri ile dış kalite arasındaki ilişki

Kod tabanlı kalite özelliği	Test Sonuçları	Dış kalite
Uyumsuzluk	r_s	-0,156
	p	0,300
Kararsızlık	r_s	-0,163
	p	0,278
Taşınabilirlik	r_s	0,184
	p	0,221
Anlaşılabilirlik	r_s	-0,371
	p	0,011

Çizelge 28. Kod tabanlı ve toplum tabanlı kalite değerleri arasındaki ilişki

Toplum tabanlı kalite metriği	Test Sonuçları	Uyumsuzluk	Kararsızlık	Taşınabilirlik	Anlaşılabilirlik
NOR	r_s	0,054	0,092	-0,132	0.020
	p	0,717	0,540	0,376	0.893
UR	r_s	-0,031	-0,030	0,018	-0,454
	p	0,836	0,840	0,904	0.001
NOP	r_s	-0,009	-0,008	0,026	-0,053
	p	0,952	0,959	0,861	0,723
NOD	r_s	-0,131	-0,084	0,074	0,287
	p	0,379	0,572	0,623	0,051
CN	r_s	0,298	0,284	-0,293	0,419
	p	0,042	0,053	0,045	0,003
TBR	r_s	-0,222	-0,267	0,286	-0,110
	p	0,134	0,070	0,051	0,460

*Gri renkli alanlar ilgili hücredeki metriklerin anlamlı bir ilişkiye sahip olduklarını göstermektedir ($p<.05$)

Bölüm 6.1'de, seçilen üç mobil uygulamanın sürümleri boyunca iç kalite değişimleri kod tabanlı metriklerle değerlendirilmiş ve karmaşıklık, boyut ve kalite değişimi üzerine yapılan analizler sonucunda genel olarak, iç kalitenin arttığı gözlenmiştir. Bölüm 6.2'de aynı uygulamaların sürümleri boyunca dış kalite değişimleri toplum tabanlı metriklerle değerlendirilmiş ve bazı sürümlerde en üst seviyeye çıkmış olmasına rağmen dış kalitenin, genel olarak Keepassdroid için artan, UPM ve PasswdSafe uygulamaları için ise azalan bir eğilim gösterdiği gözlenmiştir. Bu bölümde ise analizler sonucunda elde edilen iç kalite özellikleri ile dış kalite metrikleri (toplum tabanlı) arasında anlamlı bir ilişkinin varlığını incelemek amacıyla Spearman korelasyon analizi gerçekleştirilmiştir. Bulgular, Stamelos ve arkadaşları [133] ile Capiluppi ve arkadaşları'nın çalışmalarının [58] aksine Kawin' in yapmış olduğu çalışma ile [136] benzerlik göstermiş olup p anlamlılık seviyesine göre iç kalite özellikleri ve bazı dış kalite metrikleri arasında anlamlı bir ilişki olduğu görülmüştür. Test sonuçlarına göre, değişiklik sayısı ile uyumsuzluk ve anlaşılabilirlik kalite özellikleri arasında anlamlı ve pozitif bir ilişki gözlenirken taşınabilirlik kalite özelliği arasında anlamlı ve negatif bir ilişki gözlenmiştir. Kullanıcı oyları metriği ile anlaşılabilirlik kalite özelliği arasında ise anlamlı ve negatif bir ilişki gözlenmiştir.

6.4. Geçerliliği Tehdit Eden Faktörler

Geçerliliği tehdit eden faktörler dört kategoride incelenmektedir [148]: Yapısal geçerlilik, iç geçerlilik, dış geçerlilik ve güvenilirlik. Yapısal geçerlilik, araştırmacının hedeflerini ve araştırma sorularını ne derece doğru gözlediği ve ölçtüğü ile ilgilidir. Araştırmacı, bir faktörün diğer bir faktörü etkileyip etkilemediğini araştırdığında, araştırılan faktörün üçüncü bir faktörden de etkileneceği riski vardır. Eğer araştırmacı, üçüncü faktörün farkında değilse ve/veya araştırılan faktörü ne derecede etkilediğini bilmiyorsa, iç geçerliliğe tehdit oluşmaktadır. Dış geçerlilik, bulguları genelleştirmenin ne derece mümkün olduğuyla ilgilidir. Güvenilirlik ise bir ölçme aracının farklı ölçümlerde tekrar edilebilirliği ile ilgilidir. Bu husus, uygulanan araştırma yönteminin belirli araştırmacılara ne ölçüde bağımlı olduğu ile ilgilidir.

Bu çalışmada kullanılan kod tabanlı metriklerle ilgili yapısal geçerliliği güvence etmeye yönelik olarak, C&K metrik seti kullanılmıştır. Bu metrikler literatürde en çok kullanılan metrik seti olmasının yanı sıra mobil uygulamaların karakteristiklerini yakalamada ve hipotezleri modellemede etkili olmuştur. Toplum tabanlı değerlendirmede kullanılan metriklerin DeLone ve McLean modeline dayandırılarak kullanılması da yapısal geçerliliğe yönelik kısıtları ortadan kaldırmayı amaçlamaktadır. Ayrıca yapısal geçerliliğe tehditleri önleme amacıyla hipotezler oluşturulmuş ve istatistiksel yöntemler kullanarak test edilmiştir. Dış kalite değerlendirmesinde kullanılan toplum tabanlı metrikler arasındaki ilişki, gereksiz değişkenlerin sebep olacağı karmaşıklığı ortadan kaldırmaya yönelik olarak analiz edilmiştir. Bununla birlikte çalışmada çok sayıda karmaşıklık metriğinin kullanılmış ve bu metriklerin arasında yüksek oranda bir ilişki olup olmadığının analiz edilmemiş olması, yapısal geçerliliğe yönelik bir tehdit olarak düşünülebilir.

İç geçerliliği güvence etmek için alınan önlemler arasında, çalışmada kullanılan üç uygulamanın kaynak kodlarının doğru ve eksiksiz olması ve elde ettiğimiz metriklerin sayısal değerlerinin nesnel ölçümü gösterilebilir. Seçilen uygulamalar, açık kaynak kodlu uygulamalar olup Github ve Sourceforge gibi güvenilir veri depolarından bu uygulamalara erişim sağlanmıştır. Ayrıca ölçümler, Understand statik kod analiz kullanılarak otomatik olarak gerçekleştirilmiştir. Bu sayede, insan kaynaklı hataların ortaya çıkması büyük ölçüde engellenmiştir. Ne var ki, dış kalite değerlendirmede kullanıcı yorumlarına göre sürümler seçildiğinden ve iç kalite değerlendirmede yapıldığı gibi uygulamaların tüm sürümleri analize dâhil edilmediğinden, sürüm seçimi iç geçerliliğe ve sonuçlara yönelik bir kısıt olarak görülebilir. Ek olarak, kullanıcı oylarının kullanıcı yorumlarıyla tutarlılığının sağlanması adına herhangi bir filtreleme işleminin gerçekleştirilmemiş olması ve istatistiksel analizlerde teste tabi tutulan örneklem sayısı da iç geçerliliğe tehdit oluşturmuş olabilir. Ayrıca, bulguların yorumlanmasında araştırmacı etkisi de iç geçerliliğe yönelik olarak düşünülebilir. Bu tehditleri azaltmak adına, korelasyon analizi ile keşifsel çalışmanın sonuçları doğrulanmış ve sonuçlar daha somut verilerle yorumlanmıştır.

Dış geçerlilik, bulguların genelleştirilmesi ile ilgilidir. Yalnızca Android platformu uygulamaları incelendiğinden sonuçlar, diğer platformların uygulamaları için

geçerli olmayabilir; ayrıca elde edilen sonuçları, incelenen üç uygulama dışında başka uygulamalar için genelleştirmek de zordur.

Çalışmanın güvenilirliği ile ilgili olarak, izlenen adımların detaylı açıklanması ve oluşturulan modelin istatistiksel yöntemlerle incelenip ölçümlerin otomatik bir araçla gerçekleşmesi, bu çalışmanın farklı araştırmacılar tarafından da tekrarlanabileceğini güvence etmektedir.

7. SONUÇLAR

Bu tez çalışması ile açık kaynaklı mobil uygulamalar için Lehman'ın yazılım geliştirme yasalarının geçerli olup olmadığı ve mobil uygulamaların evrim sürecinde kod tabanlı ve toplum tabanlı açılardan evrim benzerlikleri olup olmadığı incelenmiştir.

Kod tabanlı (iç kalite) ve toplum tabanlı (dış kalite) olmak üzere iki boyutlu bir değerlendirme gerçekleştirilmiştir. İç kalite değerlendirme kapsamında karmaşıklık değişimi için C&K metrik seti, büyüme için kod satır sayısı ve sınıf sayısı metrikleri, kalite değişimi için uyumsuzluk, kararsızlık, taşınabilirlik ve anlaşılabilirlik kalite özelliklerine ait metrikler analiz edilmiştir.

Dış kalite değerlendirme için açık kaynak yazılımların başarısını ölçmede temel alınan DeLone ve McLean modeli uyarlanarak üç boyut ve altı metrik içeren bir model oluşturulmuştur. Bu model temel alınarak mobil uygulama mağazalarından elde edilen bilgiler ile eleştirici sayısı, geri bildirim sayısı, kullanıcı oyları, geliştirici sayısı, değişiklik sayısı ve sürümler arasındaki zaman metrikleri ölçülmüştür.

İlk araştırma sorusu ile mobil uygulamaların evriminde kod tabanlı (iç) kalite gelişimini değerlendirmek amaçlanmıştır. Bunun için seçilen C&K metrik setine ait metrikler Understand statik kod analiz aracı ile çıkarılmış ve sonuçlar grafiksel olarak analiz edilmiştir. Ölçüm aşamasında, yazılımların evrimi üzerine mevcut Lehman yasalarından üçü (artan karmaşıklık, sürekli büyüme ve azalan kalite) ışığında çeşitli hipotezler oluşturulmuş ve bu hipotezleri test etmek amacıyla Mann-Kendall eğilim analizi gerçekleştirilmiştir. Böylelikle, Lehman yasalarının mobil uygulamaların evriminde geçerlenip geçerlenmediği test edilmiştir. Ölçüm sonuçlarına göre Lehman'ın 'karmaşıklığın artması' yasası üzerine oluşturulan hipotezlerden sadece birinin (H_{1e}) ve 'sürekli büyüme' yasasına ilişkin oluşturulan hipotezlerin hepsinin (H_{2a}, H_{2b}) geçerliliği sağlanırken, 'azalan kalite' yasasına ilişkin kurulan iç ve dış kalite ile ilişkili hipotezlerden hiçbiri geçerli bulunmamıştır.

İkinci araştırma sorusu ile mobil uygulamaların evriminde toplum tabanlı (dış) kalite gelişimini değerlendirmek amaçlanmıştır. Bunun için, DeLone ve McLean

modelini temel alarak uyarlanan model ile üç boyut (kullanıcı memnuniyeti, kullanım ve geliştirici topluluğu hizmet kalitesi) ve bunlarla ilişkili altı metrik (kullanıcı oyları, eleştirici sayısı, geri bildirim sayısı, katkıda bulunan geliştirici sayısı, değişiklik sayısı ve sürümler arasındaki zaman) ölçülmüştür. Çalışmada dış kalite tek bir açıdan değil de birden fazla açıdan değerlendirilmek istendiğinden öncelikle, seçilen metriklerle bir başarı indeksi (SI) denklemi oluşturulmuştur. Oluşturulan başarı indeksi denklemi ile uygulamaların sürümleri boyunca gelişimleri izlenmiş ve Keepassdroid için artan, UPM ve PasswdSafe uygulamaları için azalan bir seyir gözlenmiştir.

Çalışmanın son aşamasında üçüncü araştırma sorusunu yanıtlamak üzere, kod tabanlı (iç) ve toplum tabanlı (dış) kalite arasında anlamlı bir ilişkinin varlığı incelenmiştir. Spearman korelasyon analizi sonucunda iç kalite özelliklerinden anlaşılabilirlik ve dış kalite arasında anlamlı bir ilişki olduğu görülmüştür.

7.1. Gelecek Çalışmalar

Bu tez çalışmasında mobil uygulamaların evriminde kalite değerlendirilmesi için bir dizi adımı içeren bir yöntem uygulanmıştır. Gelecekte bu yöntemin, farklı mobil yazılımlara da uygulanması ve sonuçların genellenebilirliğinin daha geniş bağlamda sınanması planlanmaktadır.

Kod tabanlı değerlendirme aşamasında C&K metrik setine ek olarak MOOD metrik setinin ve karmaşıklık analizi için McCabe döngüsel karmaşıklığının çalışmaya dâhil edilerek metrik tabanlı analizlerin daha ayrıntılandırılması planlanmaktadır.

Toplum tabanlı değerlendirmede elde edilen metrikler manuel olarak incelenmiştir. Örneğin, eleştirici sayısı metriği, uygulamaya yorum yapan kullanıcı sayısının manuel olarak tarih bazlı ayrıştırılmasıyla elde edilmiştir. Gelecekte bunların otomatik araçlar ya da metin madenciliği vb. yöntemler ile tespit edilmesi hedeflenmektedir. Ayrıca, farklı metriklerin (hata sayısı, indirme sayısı) çalışmaya dâhil edilerek elde edilen bulguların doğruluğunun sınanması da hedeflenenler arasındadır.

KAYNAKLAR

[1] Cortimiglia, M.N., A. Ghezzi, and F. Renga, Mobile applications and their delivery platforms. IT Professional, 13(5): sf. 51-56. **2011.**

[2] Corral, L., A. Janes, and T. Remencius, Potential advantages and disadvantages of multiplatform development frameworks–a vision on mobile environments. *Procedia Computer Science,* 10: sf. 1202-1207. **2012.**

[3] Statista. Number of Apps Available in Leading App Stores as of July 2014. 2017 (Mart, **2018**).

[4] Gonçalves, V., N. Walravens, and P. Ballon. "How about an App Store?" Enablers and Constraints in Platform Strategies for Mobile Network Operators. *in Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR), 2010 Ninth International Conference on. IEEE*. **2010.**

[5] Ghezzi, C., M. Jazayeri, and D. Mandrioli, Fundamentals of software engineering. *Prentice Hall PTR*. **2002.**

[6] Koskinen, J., H. Lahtonen, and T. Tilus, Software maintenance cost estimation and modernization support. *ELTIS-project, University of Jyväskylä,* **2003.**

[7] Seacord, R.C., D. Plakosh, and G.A. Lewis, Modernizing legacy systems: software technologies, engineering processes, and business practices. *Addison-Wesley Professional.* **2003.**

[8] Sommerville, I. and L. Prechelt, Verification and validation. Software Engineering, **2004.**

[9] Islam, R., R. Islam, and T. Mazumder, Mobile application and its global impact. *International Journal of Engineering & Technology (IJEST)*, 10(6): sf. 72-78. **2010.**

[10] Minelli, R. and M. Lanza. Software Analytics for Mobile Applications--Insights & Lessons Learned. in Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on. IEEE. **2013.**

[11] Zhang, J., S. Sagar, and E. Shihab. The evolution of mobile apps: an exploratory study. *in Proceedings of the 2013 International Workshop on Software Development Lifecycle for Mobile. ACM.* **2013.**

[12] 25010, I.I. ıso/ıec 25010. http://iso25000.com/index.php/en/iso-25000-standards/iso-250102011, (Nisan, **2018**).

[13] https://www.appbrain.com/, (Nisan, **2018**).

[14] Belady, L.A. and M.M. Lehman, A model of large program development. *IBM Systems journal,* 15(3): sf. 225-252, **1976.**

[15] Lehman, M.M., Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE, 68(9):* sf. 1060-1076, **1980.**

[16] Lehman, M.M. Laws of software evolution revisited. *in European Workshop on Software Process Technology. Springer.* **1996.**

[17] Li, D., et al., The evolution of open-source mobile applications: An empirical study. *Journal of Software: Evolution and Process,* 29(7), **2017.**

[18] Neamtiu, I., G. Xie, and J. Chen, Towards a better understanding of software evolution: an empirical study on open-source software. Journal of Software: Evolution and Process, 25(3): sf. 193-218, **2013.**

[19] Delone, W.H. and E.R. McLean, The DeLone and McLean model of information systems success: a ten-year update. Journal of management information systems, 19(4): sf. 9-30, **2003.**

[20] Spearman Correlation Analysis. https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php, (Mayıs, **2018**).

[21] Mann, H.B., Nonparametric tests against trend, Econometrica,. Web of Science, sf. 245–259. **1945.**

[22] Özdaş, M.R. Kamuda Açık Kaynak Kodlu Yazılım Kullanımı. 2012; https://tr.wikipedia.org/wiki/A%C3%A7%C4%B1k_kaynak, (Mayıs, **2018**).

[23] Wikipedia, Richard Stallmann, https://en.wikipedia.org/wiki/Richard_Stallman, (Mayıs, **2018**).

[24] Wikipedia. GNU. https://en.wikipedia.org/wiki/GNU, (Mayıs, **2018**).

[25] Wikipedia. Open source software. https://en.wikipedia.org/wiki/Open-source_software, (Mayıs, **2018**).

[26] Bretthauer, D., Open source software: A history. Information Technology and Libraries, 21(1): sf. 3, **2002.**

[27] Wong, K. and P. Sayo, Free/Open Source Software-A general introduction. *UNDP-APDIP, Kuala Lumpur, MY,* **2004.**

[28] App, M., https://en.wikipedia.org/wiki/Mobile_app,(Mayıs, **2018**).

[29] Wikipedia. IBM Simon. https://en.wikipedia.org/wiki/IBM_Simon, (Mayıs, **2018**).

[30] Statista. https://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/, (Mayıs, **2018**).

[31] Statista, https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, (Mayıs, **2018**).

[32] Statista. https://www.statista.com/statistics/444476/google-play-annual-revenue/, (Mayıs, **2018**).

[33] Statista. https://www.statista.com/statistics/263794/number-of-downloads-from-the-apple-app-store/, (Mayıs, **2018**).

[34] Miller, M., Build: More Details On Building Windows 8 Metro Apps. **2011.**

[35] Rosoff, M. Here's Everything You Wanted To Know About Microsoft's Upcoming iPad Killers". **2012.**

[36] Statista. https://www.statista.com/statistics/307330/number-of-available-apps-in-the-amazon-appstore/, (Mayıs, **2018**).

[37] Ericsonn, https://www.ericsson.com/en/mobility-report. **2017.**

[38] Statista, https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/], (Mayıs, **2018**).

[39] Alliance, O.H. https://www.openhandsetalliance.com/, (Mayıs, **2018**).

[40] History, A.v. https://en.wikipedia.org/wiki/Android_version_history, (Mayıs, **2018**).

[41] http://en.wikipedia.org/wiki/Android_(operating_system), (Mayıs, **2018**).

[42] Egham. Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent. **2011.**

[43] Madhavji, N.H., J. Fernandez-Ramil, and D. Perry, Software evolution and feedback: *Theory and practice. John Wiley & Sons,* **2006.**

[44] Demeyer, T.M.a.S., Software Evolution. *Springer,* **2008**.

[45] Yu, L., S. Ramaswamy, and J. Bush, Symbiosis and software evolvability. IT Professional, 10(4), **2008.**

[46] Nehaniv, C. and P. Wernick. Introduction to software evolvability. *in International Workshop on the Principles of Software Evolution (IWPSE).* **2007.**

[47] Lehman, M.M., Programs, cities, students—limits to growth?, *in Programming Methodology. Springer*. sf. 42-69, **1978.**

[48] Lehman, M.M., Laws of program evolution-rules and tools for programming management. **1978.**

[49] Lehman, M.M., On understanding laws, evolution, and conservation in the large-program life cycle. Journal of Systems and Software, 1: sf. 213-221, **1979.**

[50] Lehman, M.M. and L.A. Belady, Program evolution: processes of software change. Academic Press Professional, Inc, **1985.**

[51] Lehman, M.M., D.E. Perry, and J.F. Ramil. On evidence supporting the feast hypothesis and the laws of software evolution. *in Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International. IEEE,* **1998.**

[52] Ramil, J.F. Laws of software evolution and their empirical support. *in Software Maintenance, 2002. Proceedings. International Conference on. IEEE,* **2002.**

[53] Lakatos, I., Proofs and refutations: The logic of mathematical discovery. *Cambridge university press,* **2015.**

[54] Popper, K.R., Conjectures and Refutations. The Growth of Scientific Knowledge.(Essays and Lectures.). Routledge & Kegan Paul. **1963.**

[55] Arafat, O. and D. Riehle. The commenting practice of open source. *in Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. ACM.* **2009.**

[56] Capiluppi, A. Models for the evolution of OS projects. *in Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on. IEEE.* **2003.**

[57] Capiluppi, A., M. Morisio, and J.F. Ramil. Structural evolution of an open source system: A case study. *in Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on. IEEE.* **2004.**

[58] Capiluppi, A. and J.F. Ramil. Studying the evolution of open source systems at different levels of granularity: Two case studies. *in Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of. IEEE.* **2004.**

[59] Capra, E., C. Francalanci, and F. Merlo. The economics of open source software: an empirical analysis of maintenance costs. *in Software Maintenance, 2007. ICSM 2007. IEEE International Conference on. IEEE.* **2007.**

[60] Conley, C.A. and L. Sproull. Easier said than done: An empirical investigation of software design and quality in open source software development. *in System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on. IEEE.* **2009.**

[61] Kaur, T., N. Ratti, and P. Kaur, Applicability of lehman laws on open source evolution: a case study. *International Journal of Computer Applications, 93(18)*. **2014.**

[62] Newhook, R., et al., Evolution of the mobile enterprise app: A design perspective. *Procedia Manufacturing,* 3: sf. 2026-2033. **2015.**

[63] Gyimothy, T., R. Ferenc, and I. Siket, Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Transactions on Software engineering, 31(10)*: sf. 897-910. **2005.**

[64] Syer, M.D., et al., Studying the relationship between source code quality and mobile platform dependence. *Software Quality Journal, 23(3*): sf. 485-508. **2015.**

[65] Gonzalez-Barahona, J.M., et al., Studying the laws of software evolution in a long-lived FLOSS project. *Journal of Software: Evolution and Process,* 26(7): sf. 589-612. **2014.**

[66] Israeli, A. and D.G. Feitelson, The Linux kernel as a case study in software evolution. *Journal of Systems and Software, 83(3):* sf. 485-501. **2010.**

[67] Côté, M.-A., W. Suryn, and E. Georgiadou, In search for a widely applicable and accepted software quality model for software quality engineering. *Software Quality Journal, 15(4*): sf. 401-416. **2007.**

[68] Samadhiya, D., S.-H. Wang, and D. Chen. Quality models: Role and value in software engineering. *in Software Technology and Engineering (ICSTE), 2010 2nd International Conference on. IEEE.* **2010.**

[69] ISO, I. and T. IEC, 9126-2: Software Engineering-Product Quality-Part 2: External Metrics. *International Organization for Standardization, Geneva, Switzerland,* **2003.**

[70] 25010, I.I. https://www.iso.org/standard/35733.html. 2011, (Mayıs, **2018**).

[71] Goodman, P., The Practical Implementation of Software Metrics. McGraw-Hill, Inc., **1993.**

[72] Humphrey, W.S., Managing the software process. *Addison-Wesley Longman Publishing Co., Inc.* **1989.**

[73] Chidamber, S.R. and C.F. Kemerer, A metrics suite for object oriented design. IEEE Transactions on software engineering, 20(6): sf. 476-493. **1994.**

[74] Lee, Y.-S., B.-S. Liang, and F.-J. Wang, Some Complexity Metrics for Object-Oriented Programs Based on Information Flow: A study of C++ Programs. J. Inf. Sci. Eng., 10(1): sf. 21-50. **1994.**

[75] Li, W. and S. Henry. Maintenance metrics for the object oriented paradigm. *in Software Metrics Symposium, Proceedings., First International. 1993. IEEE.* **1993.**

[76] Lorenz, M. and J. Kidd, Object-oriented software metrics: a practical guide. Prentice-Hall, Inc. **1994.**

[77] Hudli, R.V., C.L. Hoskins, and A.V. Hudli. Software metrics for object-oriented designs. *in Computer Design: VLSI in Computers and Processors, 1994. ICCD'94. Proceedings., IEEE International Conference on. IEEE.* **1994.**

[78] McCabe, T.J., A complexity measure. IEEE Transactions on software Engineering,: sf. 308-320. **1976.**

[79] Misra, S.C. and V.C. Bhavsar. Relationships between selected software measures and latent bug-density: Guidelines for improving quality. *in International Conference on Computational Science and Its Applications. Springer.* 2003.

[80] Basili, V.R., L.C. Briand, and W.L. Melo, A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering, 22(10): sf. 751-761. **1996.**

[81] Dubey, S.K. and A. Rana, Assessment of maintainability metrics for object-oriented software system. ACM SIGSOFT Software Engineering Notes, 36(5): sf. 1-7. **2011.**

[82] Olague, H.M., et al., Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Transactions on software Engineering, 33(6):* sf. 402-419. 2007.

[83] Singh, G., D. Singh, and V. Singh, A study of software metrics. *IJCEM International Journal of Computational Engineering & Management,* 11: sf. 22-27. **2011.**

[84] Ayyildiz, T.E. and A. Koçyigit. Correlations between problem domain and solution domain size measures for open source software. *in Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on. IEEE.* **2014.**

[85] Pascarella, L. and A. Bacchelli. Classifying code comments in Java open-source software systems. *in Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on. IEEE.* **2017.**

[86] Anwer, S., et al. Effect of coupling on software faults: An empirical study. *in Communication, Computing and Digital Systems (C-CODE), International Conference on. IEEE.* **2017.**

[87] Rosenberg, L.H. and L.E. Hyatt, Software quality metrics for object-oriented environments. *Crosstalk journal, 10(4):* sf. 1-6. **1997.**

[88] Chengalur-Smith, S. and A. Sidorova, Survival of open-source projects: A population ecology perspective. ICIS 2003 proceedings, sf. 66. **2003.**

[89] O'Reilly, T., Lessons from open-source software development. *Communications of the ACM*, 42(4): sf. 32-37. **1999.**

[90] Dedrick, J. and J. West. An exploratory study into open source platform adoption. *in System Sciences, Proceedings of the 37th Annual Hawaii International Conference on. IEEE.* **2004.**

[91] Stewart, K. and T. Ammeter, An exploratory study of factors influencing the level of vitality and popularity of open source projects. ICIS 2002 Proceedings, sf. 88. **2002.**

[92] DeLone, W.H. and E.R. McLean, Information systems success: The quest for the dependent variable. Information systems research, 3(1): sf. 60-95. **1992.**

[93] DeLone, W.H. and E.R. McLean. Information systems success revisited. *in System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on. IEEE.* **2002.**

[94] Crowston, K., H. Annabi, and J. Howison, Defining open source software project success. Former Departments, Centers, Institutes and Projects, sf. 4. 2003.

[95] Seddon, P.B., A respecification and extension of the DeLone and McLean model of IS success. Information systems research, 8(3): sf. 240-253. **1997.**

[96] Boehm, B.W., J.R. Brown, and M. Lipow. Quantitative evaluation of software quality. *in Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press.* **1976.**

[97] Gorton, I. and A. Liu. Software component quality assessment in practice: successes and practical impediments. *in Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on. IEEE.* **2002.**

[98] Seddon, P.B., et al., Dimensions of information systems success. *Communications of the AIS, 2(3es):* sf. 5. **1999.**

[99] Rai, A., S.S. Lang, and R.B. Welker, Assessing the validity of IS success models: An empirical test and theoretical analysis. Information systems research, 13(1): sf. 50-69. **2002.**

[100] Scacchi, W., Understanding the requirements for developing open source software systems. IEE Proceedings-Software, 149(1): sf. 24-39. **2002.**

[101] Feller, J. and B. Fitzgerald, Understanding open source software development. Addison-Wesley London. **2002.**

[102] Stewart, K.J., A.P. Ammeter, and L.M. Maruping. A preliminary analysis of the influences of licensing and organizational sponsorship on success in open source projects. *in System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on. IEEE.* **2005.**

[103] Raymond, E.S., The cathedral and the bazaar-musings on Linux and open source by an accidental revoltionary (rev. ed.)., O'reilly Beijing. **2001.**

[104] Crowston, K., J. Howison, and H. Annabi, Information systems success in free and open source software development: Theory and measures. Software Process: Improvement and Practice, 11(2): sf. 123-148. **2006.**

[105] Grewal, R., G.L. Lilien, and G. Mallapragada, Location, location, location: How network embeddedness affects project success in open source systems. Management Science, 52(7): sf. 1043-1056. **2006.**

[106] Hindle, A., D.M. German, and R. Holt. What do large commits tell us?: a taxonomical study of large commits. *in Proceedings of the 2008 international working conference on Mining software repositories. ACM.* **2008.**

[107] Midha, V. and P. Palvia, Factors affecting the success of Open Source Software. *Journal of Systems and Software,* 85(4): sf. 895-905. **2012.**

[108] Singh, P. and Y. Tan. Stability and Efficiency of Communication Networks in Open Source Software Development. *in Proceedings of the Fifteenth annual Workshop on Information Technology and Systems WITS, Las Vegas.* **2005.**

[109] Colazo, J.A., Innovation success: An empirical study of software development projects in the context of the open source paradigm. University of Western Ontario. **2007.**

[110] Subramaniam, C., R. Sen, and M.L. Nelson, Determinants of open source software project success: A longitudinal study. Decision Support Systems, 46(2): sf. 576-585. **2009.**

[111] Amrollahi, A., M. Khansari, and A. Manian, How Open Source Software Succeeds? A Review of Research on Success of Open Source Software. **2014.**

[112] Hu, N., P.A. Pavlou, and J. Zhang. Can online reviews reveal a product's true quality?: empirical findings and analytical modeling of Online word-of-mouth communication. *in Proceedings of the 7th ACM conference on Electronic commerce. ACM.* **2006.**

[113] Chevalier, J.A. and D. Mayzlin, The effect of word of mouth on sales: Online book reviews. *Journal of marketing research,* 43(3): sf. 345-354. **2006.**

[114] Corral, L. and I. Fronza. Better code for better apps: a study on source code quality and market success of Android applications. *in Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems. IEEE Press.* **2015.**

[115] Lehman, M.M., et al. Metrics and laws of software evolution-the nineties view. *in Software metrics symposium, 1997. proceedings., fourth international. IEEE*. **1997.**

[116] Lehman, M.M. and J.F. Ramil, Rules and tools for software evolution planning and management. Annals of software engineering, 11(1): sf. 15-44. **2001.**

[117] Kemerer, C.F., Software complexity and software maintenance: A survey of empirical research. Annals of Software Engineering, 1(1): sf. 1-22. **1995.**

[118] Paulson, J.W., G. Succi, and A. Eberlein, An empirical study of open-source and closed-source software products. IEEE Transactions on Software Engineering, 30(4): sf. 246-256. **2004.**

[119] Kemerer, C.F. and S. Slaughter, An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering, 25(4)*: sf. 493-509. **1999.**

[120] Tu, Q. Evolution in open source software: A case study. *in Software Maintenance, 2000. Proceedings. International Conference on. IEEE.* **2000.**

[121] Wang, Y., D. Guo, and H. Shi, Measuring the evolution of open source software systems with their communities. ACM SIGSOFT Software Engineering Notes, 32(6): sf. 7. **2007.**

[122] Mockus, A., R.T. Fielding, and J. Herbsleb. A case study of open source software development: the Apache server. in Proceedings of the 22nd international conference on Software engineering. ACM. **2000.**

[123] Antoniol, G., et al. Mining the lexicon used by programmers during sofware evolution. *in Software Maintenance, 2007. ICSM 2007. IEEE International Conference on. IEEE.* **2007.**

[124] Izurieta, C. and J. Bieman. The evolution of FreeBSD and Linux. *in Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM.* **2006.**

[125] Gonzalez-Barahona, J.M., et al., Macro-level software evolution: a case study of a large software compilation. Empirical Software Engineering, 14(3): sf. 262-285. **2009.**

[126] Peng, Y., F. Li, and A. Mili, Modeling the evolution of operating systems: An empirical study. Journal of Systems and Software, 80(1): sf. 1-15. **2007.**

[127] Herraiz, I., J.M. Gonzalez-Barahona, and G. Robles. Determinism and evolution. *in Proceedings of the 2008 international working conference on Mining software repositories. ACM.* **2008.**

[128] Herraiz, I. A statistical examination of the evolution and properties of libre software. *in Software Maintenance, 2009. ICSM 2009. IEEE International Conference on. IEEE.* **2009.**

[129] Fernandez-Ramil, J., D. Izquierdo-Cortazar, and T. Mens. What does it take to develop a million lines of open source code? *in IFIP International Conference on Open Source Systems. Springer.* **2009.**

[130] Harman, M., Y. Jia, and Y. Zhang. App store mining and analysis: MSR for app stores. *in Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on. IEEE.* **2012.**

[131] Pagano, D. and W. Maalej. User feedback in the appstore: An empirical study. *in Requirements Engineering Conference (RE), 2013 21st IEEE International. IEEE.* **2013.**

[132] Hecht, G., et al. Tracking the software quality of android applications along their evolution (t). *in Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. IEEE.* **2015.**

[133] Stamelos, I., et al., Code quality analysis in open source software development. *Information Systems Journal,* 12(1): sf. 43-60. **2002.**

[134] Goeminne, M. and T. Mens. Towards the analysis of evolution OSS ecosystems. *in BENEVOL 2009 The 8 th BElgian-NEtherlands software eVOLution seminar.* **2009.**

[135] Abreu, R. and R. Premraj. How developer communication frequency relates to bug introducing changes. *in Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops. ACM.* **2009.**

[136] Ngamkajornwiwat, K., et al. An exploratory study on the evolution of oss developer communities. *in Hawaii International Conference on System Sciences, Proceedings of the 41st Annual. IEEE.* **2008.**

[137] Guinan, P.J., J.G. Cooprider, and S. Faraj, Enabling software development team performance during requirements definition: A behavioral versus technical approach. Information systems research, 9(2): sf. 101-125. **1998.**

[138] Lakhani, K.R. and R.G. Wolf, Why hackers do what they do: Understanding motivation and effort in free/open source software projects. **2003.**

[139] Rosenberg, L., T. Hammer, and J. Shaw. Software metrics and reliability. *in 9th International Symposium on Software Reliability Engineering.* **1998.**

[140] Linares-Vásquez, M., et al. API change and fault proneness: a threat to the success of Android apps. *in Proceedings of the 2013 9th joint meeting on foundations of software engineering. ACM.* **2013.**

[141] McDonnell, T., B. Ray, and M. Kim. An empirical study of api stability and adoption in the android ecosystem. *in Software Maintenance (ICSM), 2013 29th IEEE International Conference on. IEEE.* **2013.**

[142] Mann Kendall Trend Analysis. http://www.statisticshowto.com/mann-kendall-trend-test/. (Mayıs, **2018**).

[143] Statistics. https://www.statisticssolutions.com/correlation-pearson-kendall-spearman/. (Mayıs, **2018**).

[144] Kolmogorov, A.N., Sulla determinazione empirica di una legge di distributione. sf. 83-91, **1934.**

[145] Shapiro, S.S. and M.B. Wilk, An analysis of variance test for normality (complete samples). Biometrika, 52(3/4): sf. 591-611. **1965.**

[146] Razali, N.M. and Y.B. Wah, Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. Journal of statistical modeling and analytics, 2(1): sf. 21-33, **2011.**

[147] Spearman, C., The proof and measurement of association between two things. The American journal of psychology, 15(1): sf. 72-101. **1904.**

[148] Runeson, P. and M. Höst, Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering, 14(2): sf. 131. **2009.**

ÖZGEÇMİŞ

Kimlik Bilgileri

Adı Soyadı: Bahar GEZİCİ

Doğum Yeri: Batman

Medeni Hali: Bekâr

E-posta: bahargezici@hacettepe.edu.tr

Adresi: Hacettepe Üniversitesi Bilgisayar Mühendisliği Bölümü, Araştırma Laboratuvarı 5

Eğitim

Lise: Ziya Gökalp Anadolu Lisesi, Batman, 2007

Lisans: Kocaeli Üniversitesi, Bilgisayar Mühendisliği, Kocaeli, 2014

Yabancı Diller

İngilizce

İş Deneyimi

Araştırma Görevlisi, Hacettepe Üniversitesi, 2016

Deneyim Alanları

Yazılım Mühendisliği, Yazılım Kalite

Tezden Üretilmiş Projeler ve Bütçesi

-

Tezden Üretilmiş Yayınlar

1. Özdemir, N., Gezici, B., & Dinçer, K. Mobil Uygulamaların Kalite Özelliklerinin Ölçümü. In CEUR Workshop Proceedings (Vol. 1721, pp. 337–348). (2016)
2. Gezici, B., Tarhan, A., & Chouseinoglou, O. Mobil Uygulamaların Evriminde Karmaşıklık, Boyut ve İç Kalite Gelişimi. Gazi Üniversitesi Mühendislik-Mimarlık Dergisi (ilk kabul sonrası revizyon altında). (2018)

Tezden Üretilmiş Tebliğ ile Katıldığı Toplantılar

-

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS/DOKTORA TEZ ÇALIŞMASI ORJİNALLİK RAPORU

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ
Bilgisayar Mühendisliği ANABİLİM DALI BAŞKANLIĞI'NA

Tarih: 04/06/2018

Tez Başlığı / Konusu: Mobil Uygulamaların Evriminde Kalıtımın Gelişimi

Yukarıda başlığı/konusu gösterilen tez çalışmamın a) Kapak sayfası, b) Giriş, c) Ana bölümler d) Sonuç kısımlarından oluşan toplam 110 sayfalık kısmına ilişkin, 04/06/2018 tarihinde ~~şahsım~~/tez danışmanım tarafından *Turnitin* adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 8 'tür.

Uygulanan filtrelemeler:

1- Kaynakça hariç
2- Alıntılar hariç/~~dâhil~~
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

Adı Soyadı: Baha Gezici — 04/06/2018
Öğrenci No: N19226816
Anabilim Dalı: Bilgisayar Mühendisliği
Programı: Bilgisayar Mühendisliği
Statüsü: ☒ Y.Lisans ☐ Doktora ☐ Bütünleşik Dr.

DANIŞMAN ONAYI

UYGUNDUR.

Dr. Öğretim Üyesi Ayça Tarhan
(Unvan, Ad Soyad, İmza)