Journal of the Faculty of Engineering and Architecture of Gazi University 34:3 (2019) 1483-1500

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

Complexity, size and internal quality in the evolution of mobile applications: An exploratory study

Bahar Gezici[1*], Ayça Tarhan[1], Oumout Chouseinoglou[2]

[1]Department of Computer Engineering, Hacettepe University, Ankara, 06800, Turkey
[2]Department of Industrial Engineering, Hacettepe University, Ankara, 06800, Turkey

Highlights:

- Complexity, size and internal quality in the evolution of mobile applications
- Open source software and software evolution
- Software quality evaluation using Spearman correlation analysis technique

Keywords:

- Open source software
- Software evolution
- Software quality
- C&K metric set
- Lehman laws

Article Info:

Research Article
Received: 09.04.2018
Accepted: 03.07.2018

DOI:

10.17341/gazimmfd.460547

Correspondence:

Author: Bahar Gezici
e-mail: bahargezici@cs.hacettepe.edu.tr
phone: +90 312 780 75 55

Graphical/Tabular Abstract

Mobile applications, though being relatively small and easy to manage in their initial releases, are becoming complex software systems as they evolve rapidly 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.

Hypotheses tested

Lehman Law	Hypotheses	Mobile Applications: Keepass droid	UPM	Passwd Safe
Law 1. Increased Complexity (Lehman 2)	NOC increases over time			
	DIT increases over time			
	RFC increases over time		✓	
	WMC increases over time		✓	
	CBO increases over time	✓	✓	✓
Law 2. Continuing Growth (Lehman 6)	LOC increases over time	✓	✓	✓
	Number of classes increases over time	✓	✓	✓
Law 3. Declining Quality (Lehman 8)	LCOM increases over time			
	Instability increases over time			
	Portability decreases over time			
	Understandability decreases over time			

Figure A. Hypotheses tested and validation results

Purpose: This study aims to better understand the development of complexity, size and internal quality in the evolution of mobile applications and, in particular, to investigate the validity of three of Lehman's laws (increasing complexity, continuous growth, decreasing quality) in mobile applications.

Theory and Methods:
In this context, an exploratory study was carried out by analyzing the evolution of application quality, according to hypotheses established and using object-oriented design metrics in 61 versions of three Android-based mobile applications.

Results:
As a result of the conducted analyses, Lehman's 'continuous growth' law was validated for all apps, while the 'increased complexity' and 'declining quality' laws have not been validated. In addition, the results of the experimental study have been verified with Spearman correlation analysis and it was observed that there is a significant relation between the design metrics and the quality attributes.

Conclusion:
In this study, the complexity, size and internal quality in the evolution of mobile applications was explored. The study is expected to guide similar studies on the evolution and maintenance of mobile software.

Journal of the Faculty of Engineering and Architecture of Gazi University 34:3 (2019) 1483-1500

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

Mobil uygulamaların evriminde karmaşıklık, boyut ve iç kalite gelişimi: Keşifsel bir çalışma

Bahar Gezici[1]*, Ayça Tarhan[1], Oumout Chouseinoglou[2]

[1]Hacettepe Üniversitesi, Bilgisayar Mühendisliği Bölümü, Yazılım Mühendisliği Araştırma Grubu (HUSE), 06800, Ankara, Türkiye
[2]Hacettepe Üniversitesi, Endüstri Mühendisliği Bölümü, 06800, Ankara, Türkiye

ÖNEÇIKANLAR

- Mobil uygulamaların gelişiminde karmaşıklık, boyut ve iç kalite
- Açık kaynaklı yazılım ve yazılım gelişimi
- Spearman korelasyon analizi tekniğini kullanarak yazılım kalitesi değerlendirmesi

Makale Bilgileri

Araştırma Makalesi
Geliş: 09.04.2018
Kabul: 03.07.2018

DOI:
10.17341/gazimmfd.460547

Anahtar Kelimeler:
Açık kaynak kodu, mobil yazılım, yazılım evrimi, yazılım kalitesi, C&K metrik seti, Lehman yasaları

ÖZET

Mobil uygulamalar, kullanıcı gereksinimlerini karşılama ihtiyacı dolayısıyla hızlı bir şekilde büyüyerek sürekli gelişmekte ve buna bağlı olarak karmaşık yazılım sistemleri haline gelmektedir. Bununla birlikte bu gereksinimlerin 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. Bu çalışma, mobil uygulamaların evriminde karmaşıklık, boyut ve iç kalite gelişimini daha iyi anlamayı ve özellikle Lehman yasalarından üçünün (artan karmaşıklık, sürekli büyüme, azalan kalite) mobil uygulamalarda geçerliliğini araştırmayı hedeflemektedir. Bu kapsamda Android tabanlı üç mobil uygulamanın toplam 61 sürümünde uygulama kalitesinin gelişimi, kurulan hipotezler üzerinden ve nesneye yönelik tasarım metriklerine göre analiz edilerek keşifsel bir çalışma gerçekleştirilmiştir. Analizler sonucunda, Lehman'ın 'sürekli büyüme' yasası tüm uygulamalar için geçerlenirken "artan karmaşıklık" ve 'azalan kalite' yasaları geçerli bulunmamıştır. Ayrıca Spearman korelasyon analizi ile keşifsel çalışmanın sonuçları doğrulanmış ve tasarım metrikleri ile kalite özellikleri arasında anlamlı bir ilişki olduğu gözlenmiştir.

Complexity, size and internal quality in the evolution of mobile applications: An exploratory study

HIGHLIGHTS

- Complexity, size and internal quality in the evolution of mobile applications
- Open source software and software evolution
- Software quality evaluation using Spearman correlation analysis technique

Article Info

Research Article
Received: 09.04.2018
Accepted: 03.07.2018

DOI:
10.17341/gazimmfd.460547

Keywords:
Open source, mobile software, software evolution, software quality, C&K metric set, Lehman laws

ABSTRACT

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. This study aims to better understand the development of complexity, size and internal quality in the evolution of mobile applications and, in particular, to investigate the validity of three of Lehman's laws (increasing complexity, continuous growth, decreasing quality) in mobile applications. In this context, an exploratory study was carried out by analyzing the evolution of application quality, according to hypotheses established and using object- oriented design metrics in 61 versions of three Android-based mobile applications. As a result of the analyses, Lehman's 'continuous growth' law was validated for all apps, while the 'increased complexity' and 'declining quality' laws have not been validated. In addition, the results of the experimental study have been verified with Spearman correlation analysis and it was observed that there is a significant relation between the design metrics and the quality attributes.

*Sorumlu Yazar/Corresponding Author: bahargezici@hacettepe.edu.tr, atarhan@hacettepe.edu.tr, uhus@hacettepe.edu.tr /
Tel: +90 312 780 75 55

1. GİRİŞ (INTRODUCTION)

Mobil uygulamalar akıllı telefonlar, tablet bilgisayarlar ve diğer mobil cihazlarda çalışmak üzere tasarlanmış yazılımlardır [1, 2]. Günümüzde sosyal ağlar, seyahat, sağlık, bankacılık, takvimler, oyunlar, haberler ve daha fazlası için milyonlarca mobil uygulama mevcut durumdadır. Son on yılda, mobil uygulamaların popülerliği artmıştır ve 2017 yılı itibariyle Google Play'de 3,5 milyon, Apple App Store'da 2,2 milyon, Amazon App Store'da yaklaşık 385 bin mobil uygulama bulunmaktadır [3]. 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; başka çalışmalar ise bu maliyetlerin, ilk yazılım sürümüne ait maliyetin birkaç katına kadar çıktığını göstermektedir [4-7]. Yazılım üretme maliyetlerini azaltmak için, hem yöneticiler hem de geliştiriciler yazılım geliştirmeyi yönlendiren faktörleri anlamalı ve değişiklikleri kolaylaştıran, yazılımın uzun süreler kullanılmaya devam edebilmesi 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 [8-10].

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 [11] ve bu standartta 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.

1976'da Lehman ve Belady [12], yazılımın evrimi üzerine somut deneysel çalışmalar gerçekleştirmiş ve bir çalışmada yazılım gelişimini karakterize eden bir dizi yasa önermiştir [13]. 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- Geribildirim sistemleri' 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 [14, 15].

Literatürde mobil uygulamaların evrimleri boyunca karmaşıklık, boyut ve kalite açılarından nasıl bir gelişme gösterdikleri konusunda çalışmalar mevcut olmakla birlikte sayıca azdır ve bu açıdan bir boşluk söz konusudur. Bu boşluğu adreslemek amacıyla bu çalışmada, mobil uygulamaların evrimleri sırasında ne gibi değişimlerden geçtiklerini keşifsel olarak incelemek hedeflenmiştir. Keşifsel araştırma, genellikle daha önce yeterli sayıda çalışma yapılmamış konularda uygulanan bir araştırma şeklidir ve gelecekte yapılacak çalışmalar için yön belirlemeye yarar [16]. Katkımız ile gelecekte yapılacak çalışmalar için araştırma tabanının güçlenmesi beklenmektedir.

Bahsedilen hedefe ulaşmak amacıyla, ISO 25010 standardını temel alarak ve Lehman'ın [13] sekiz yazılım geliştirme yasasından karmaşıklık, boyut ve iç kalite ile ilişkili olan ikinci, altıncı ve yedinci yasası kapsamında, oluşturulan hipotezler test edilmiştir. Bu yasaların seçilmesindeki ilk sebep, yaptığımız çalışmanın kalite boyutuna odaklanmasıdır; bu sebeple kalite ile ilişkili yasa ve bunu değerlendirmede tamamlayıcı olabilecek boyut ve karmaşıklık yasaları da çalışmaya dâhil edilmiştir. İkinci bir sebep ise yasaları değerlendirmeyi destekleyecek bağlamın ve metriklerin varlığıdır.

- 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".

Bu çalışma, toplamda 61 sürümü içeren üç mobil uygulamanın (Keepassdroid, UPM ve PasswdSafe) sürümleri boyunca karmaşıklık, boyut ve iç kalitenin gelişimi üzerinde yoğunlaşmaktadır. Keepassdroid ve PasswdSafe uygulamalarının son beş yılda üretilen 12 ve 33 sürümü, UPM uygulamasının tüm sürümleri (16) analiz edilmiştir. Çalışmaya özel olarak oluşturulan araştırma soruları (AS) aşağıda verilmiştir. Araştırma sorularının yanıtlanması için hipotezler oluşturulmuş ve bu hipotezler, sözü edilen üç açık kaynak kodlu mobil uygulamanın sürümleri bazında toplanan yazılım tasarım metrikleri üzerinde istatistiksel analizler yapılarak test edilmiştir.

- AS 1. Mobil uygulamalar evrimleri süresince karmaşıklık, boyut ve iç kalite açılarından nasıl bir gelişim göstermektedir?
- AS 2. Lehman'ın artan karmaşıklık, sürekli büyüme ve azalan kalite yasaları mobil uygulamalar için geçerli midir?

Bu makalenin ikinci bölümünde nesneye yönelik tasarım metrikleri ve ilişkili kalite özelliklerine ait bilgilere, üçüncü bölümde mobil uygulamaların evrim süreci ile ilgili literatürde mevcut olan çalışmalara yer verilmiştir. Dördüncü bölümde keşifsel çalışmanın amacı, metriklerin çıkarılması, verilerin toplanması ve hipotez testlerinin adımları sunulmuştur. Beşinci bölümde analiz sonuçlarının grafiksel değerlendirmesine, altıncı bölümde ise bulgulara yönelik tartışmaya, keşifsel sonuçların korelasyon analizi ile sınanmasına ve geçerliliği tehdit eden faktörlere yer verilmiştir. Yedinci bölümde çalışmanın sonuçları anlatılmıştır.

2. NESNEYE YÖNELİK TASARIM METRİKLERİ VE İLİŞKİLİ KALİTE ÖZELLİKLERİ
(OBJECT ORIENTED DESIGN METRICS AND RELATED QUALITY ATTRIBUTES)

Mobil uygulamalar çoğunlukla Java veya Objective-C gibi nesneye yönelik (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 [17-20]. Kaliteli yazılımın uyumluluğunun (cohesion) yüksek, karmaşıklık (complexity) ve bağımlılığının (coupling) düşük olduğu yaygın olarak kabul edilmektedir [17].

2.1. Nesneye Yönelik Tasarım Metrikleri (OO Design Metrics)

Chidamber ve Kemerer'in tanımlamasından bu yana [17], yazılım kalitesini değerlendirmek için nesneye yönelik metrikler popülerlik kazanmış olup kısaca C&K metrik seti olarak isimlendirilmektedir. C&K metrik seti, nesneye yönelik tasarımın farklı özelliklerini değerlendiren altı metrikten oluşmaktadır.

Sınıfın Ağırlıklı Metot Sayısı (Weighted Methods per Class (WMC)): Bir sınıfın tüm metotlarının karmaşıklığının toplamıdır. Tüm metotların karmaşıklığı 1 kabul edilirse, WMC sınıfın metot sayısına eşit olur. Sınıfın metotlarının sayısı ve metotlarının karmaşıklığı, sınıfın geliştirilmesine ve bakımına ne kadar zaman harcanacağı hakkında fikir verebilir. Metot sayısı çok olan taban sınıflar, tanımlanan tüm metotlar türetilen 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. WMC, anlaşılabilirlik, bakım yapılabilirlik ve yeniden kullanılabilirliği ölçer [17, 20, 21]. Ortalama WMC'deki bir artış hata yoğunluğunu arttırır ve kaliteyi düşürür [22].

Sınıfın Tetiklediği Metot Sayısı (Response For a Class (RFC)): Verilen sınıftan bir nesnenin metotları çağrıldığında, bu nesnenin tetikleyebileceği tüm metotların sayısıdır. RFC, sınıfın test maliyeti hakkında da fikir verir. 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. RFC, anlaşılabilirlik, bakım yapılabilirlik ve test edilebilirliği değerlendirir [17, 20, 21]. RFC, sınıftaki metot sayısı ile sınıftaki metotlar tarafından doğrudan çağrılan metot sayısının toplamına eşittir.

Metot İçi Uyumsuzluk (Lack of Cohesion in Methods (LCOM)): C1 sınıfının M1, M2, …, Mn metotları olduğunu ve {Ii} 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 (LCOM1). Sınıfın uyumluluğunun düşük olması, sınıfın iki veya daha fazla alt parçaya bölünmesi gerektiğini gösterir. Düşük uyumluluk karmaşıklığı artırır, bu nedenle geliştirme aşamasında hata yapılma ihtimali yükselir. Ayrıca metotlar arasındaki ilişkisizliklerin ölçüsü sınıfların tasarımındaki kusurların belirlenmesinde de yardımcı olabilir. Bu metrik, verimliliği ve yeniden kullanılabilirliği değerlendirir [17, 20, 21, 23].

Nesne Sınıfları Arasındaki Bağımlılık (Coupling Between Object Classes (CBO)): Sınıfın bağımlı olduğu sınıf sayısıdır. Bir sınıf içindeki nitelikler (attribute) ya da metotlar başka bir sınıfta kullanılıyor ve bu sınıflar arasında kalıtım yoksa iki sınıf arasında bağımlılık olduğu kabul edilir. Sınıflar arasındaki aşırı bağımlılık modüler tasarıma zarar verir ve tekrar kullanılabilirliği azaltır. Bağımlılıktaki artış, değişime duyarlılığı da arttıracağından, yazılımın bakımı daha zor; testlerin daha özenli yapılması gerektiğinden de test maliyetleri daha yüksektir [17, 23]. CBO, verimliliği ve yeniden kullanılabilirliği değerlendirir [19- 21].

Kalıtım Ağacının Derinliği (Depth of Inheritance Tree (DIT)): Bir sınıfın, kalıtım ağacının köküne uzaklığıdır. Hiçbir sınıftan türetilmemiş sınıflar için DIT 0'dır. Eğer çoklu kalıtım varsa, en uzak köke olan uzaklık kabul edilir. Kalıtım hiyerarşisinde daha derinde olan sınıflar, daha çok metot türettiklerinden davranışlarını tahmin etmek daha zordur. Derin kalıtım ağaçları daha çok tasarım karmaşıklığı oluşturur. Bu metrik öncelikle verimliliği ve yeniden kullanılabilirliği değerlendirir [17, 20] ancak aynı zamanda, anlaşılabilirlik ve test edilebilirlik ile de ilgilidir [21, 23].

Alt Sınıf Sayısı (Number of Children (NOC)): Sınıftan doğrudan türetilmiş alt sınıfların sayısıdır. Eğer 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. NOC, bu nedenle öncelikle verimliliği, yeniden kullanılabilirliği ve

test edilebilirliği değerlendirmede kullanılmaktadır [17, 20, 21]. Birçok çalışmada [24-27] 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 [24].
- C&K metrik setindeki yazılım metrikleri neredeyse tüm önemli kalite özelliklerini (bakım yapılabilirlik, test edilebilirlik vb.) temsil etmektedir [24-26]. Örnek olarak DIT kalıtımı, CBO nesnelerin bağımlılığını, WMC sınıfın karmaşıklığını ve LCOM sınıflar arasındaki uyumluluğu ölçer. Özetle tüm metrikler, nesneye yönelik sistemlerde 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.

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

Kod satır sayısı (Line of code (LOC)): Yazılımın büyüklüğü hakkında bilgi verir. Yazılım büyüklüğü, yazılım geliştirme maliyeti ve süresinin kestiriminde, başarım ve kalite ölçümünde vb. farklı amaçlarla kullanılmaktadır [28].

Yorum satır sayısı (Comment Line of code (CLOC)): Program için yazılmış yorum satırlarının sayısıdır. Yorum satır sayısının fazla olması anlaşılabilirliği ve bakımı kolaylaştırmaktadır [29].

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 ifade edilebilir [30].

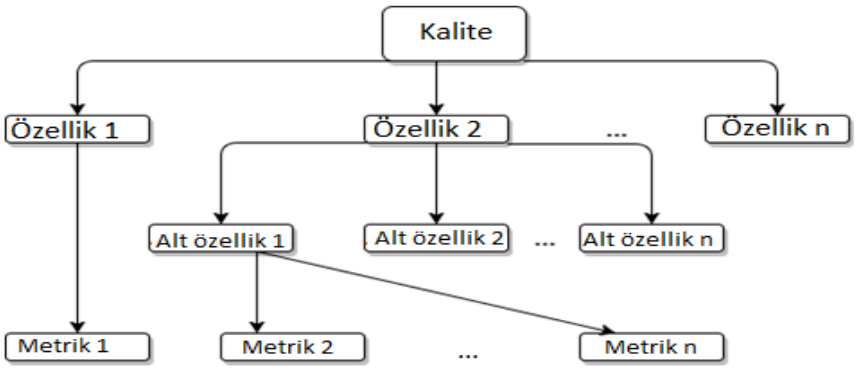
Dışa bağımlılık (Efferent couplings (Ce)): İncelenen sınıfın kendi dışında kaç tane sınıfa bağımlı olduğunu gösterir. Bir sınıfın dışa bağımlılığını ifade eder [30].

2.2. Kalite Özellikleri ve Kaynak Kodu Özellikleri (Quality and Source Code Attributes)

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 kalite değerlendirme metodunun aşamalarından biri, kaynak kod özelliklerini değerlendirmek için uygun metrik kümesini tanımlamaktır. Tablo 1, C&K metrik seti ile ilişkili ve ISO 25010 kalite özelliklerini etkileyen kaynak kodu özelliklerini, tanımlarıyla birlikte genel bilgi olması açısından sunmaktadır. Tablo 1'de verilen kaynak kodu özelliklerinin her biri, bir veya daha fazla iyi tanımlanmış metrik kullanarak nesnel olarak değerlendirilebilir. Kalite modelleri, hiyerarşik yapıya sahiptir (Şekil 1). Kalite özellikler ile tanımlanır ve bazı özellikler, alt özelliklerden oluşur. Özellikler (varsa alt özellikler) ise, ölçülebilen kalite özellikleri (metrikler) ile değerlendirilir. ISO 25010'a göre bir kaynak kodu özelliği, bir veya daha fazla alt özelliğe katkıda bulunabilir [11]. 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 özelliklerini, azalan kalite yasasına (Yasa 3) ilişkin oluşturulan hipotez ise C&K metrik setinin uyumluluk özelliklerini yansıttığından, Tablo 2'ye ilişkin sınıflandırma bu üç başlık altında gerçekleştirilmiştir. Ayrıca her bir kaynak kodu özelliği, C&K metrik setine ait metriklerle ölçülebilir ve bu ölçüm sonuçları, ISO 25010'a ait kalite özellikleri hakkında öngörüde bulunmayı sağlayabilir. Bu sebeple Tablo 2'de, literatürde ele alınan ve çalışmada kullanılan C&K metrik setinin ilişkili olduğu, sadece çalışmada analiz edilecek özelliklere yer verilmiştir. Kalite özelliği, sistemin paydaşlarını tatmin etme kabiliyeti için kullanılan, ölçülebilir bir özelliktir [11]. Bir yazılımın çeşitli bakış açılarından kaliteli olup olmadığını değerlendirmek için bu özelliklerden faydalanılmaktadır. Bu çalışmada kullanılan kalite özelliklerine ait detaylar Tablo 3'te verilmiştir.

Tablo 1. Kaynak kodu özellikleri (Source code attributes)

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



Şekil 1. Yazılım ürün kalite modellerinin hiyerarşik yapısı (Hierarchical structure of software product quality models)

Tablo 2. Kaynak kodu özelliği, C&K metrik seti ve yazılım kalite özellikleri arasındaki ilişki
(Relationship between source code attributes, C&K metric set, and software quality characteristics)

Kaynak Kodu Özelliği	C&K Metriği	İlişkili Çalışmalarda Ele Alınan Kalite Özelliği/Alt-özelliği	İlişkili Çalışmalara Referanslar
Karmaşıklık	-WMC -RFC -DIT -NOC	Bakım yapılabilirlik, Anlaşılabilirlik, Kullanılabilirlik, Yeniden kullanılabilirlik, Test edilebilirlik, Verimlilik, Taşınabilirlik	[17, 20, 21, 23]
Uyumluluk	-LCOM	Yeniden Kullanılabilirlik, Verimlilik, İşlevsellik, Bakım yapılabilirlik, Taşınabilirlik	[17, 20, 21, 23]
Bağımlılık	-CBO	Yeniden Kullanılabilirlik Verimlilik, Bakım yapılabilirlik, Taşınabilirlik	[17, 19-21, 23]

Tablo 3. Çalışmada incelenen her bir kalite özelliği, tanımı ve formülü
(Quality attributes, definitions and formulae examined in the study)

Kalite Özelliği	Tanımı	Formülü
Kararlılık	Yeni gereksinimlerle karşı karşıya kalırken veya platform değiştirirken yazılımın değişime karşı gösterdiği direncin ölçüsü olarak tanımlanır [10]. 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) [31]
Taşınabilirlik	Bir sistem, ürün veya bileşenin bir donanım, yazılım, başka bir işletim veya kullanım ortamından diğerine aktarılabilme, farklı çalışma ortamlarına uyum sağlayabilme yeteneğidir [10]. Taşınabilirlik, özellikle sürekli büyüyen mobil pazarda, bir yazılım ürününün sahip olması arzu edilen bir özelliktir.	1 – DIT [32]
Anlaşılabilirlik	Farklı altyapıya sahip kullanıcıların mimariyi ne ölçüde anlayabileceklerini ifade eder [10]. 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.	- [(0.25 * CBO) + (0.25 * LCOM) + (0.25 * LOC) + (0.25 * (0.5 * LOC) + (0.5 * metot sayısı))] [32]

ISO 25010 kalite modelinin temel amaçlarından biri, ISO 9126'ya dayanan bir dizi standart oluşturmak ve kalite şartlarının belirlenmesi ve değerlendirilmesi ile yazılım ürünlerinin geliştirilmesine rehberlik etmektir. Bu çalışmada, araştırmamıza temel oluşturan ilişkili çalışmaların ISO 9126 standardını ve onun kalite özelliklerini kullanmış olmaları sebebiyle, ISO 25010'da yer alan kalite özellikleri yerine, izlenirliği sağlamak adına ISO 9126'nın kalite özellikleri isimlendirmeye temel alınmıştır. Bu bakımdan Tablo 3'teki kalite özellikleri, aşağıdaki kısıtlara göre okunmalıdır. Anlaşılabilirlik (understandability) kalite alt-özelliği, ISO 9126 standardında Kullanılabilirlik (usability) kalite özelliği altında yer alırken ISO 25010'da yer almamaktadır. Anlaşılabilirlik (understandability) kalite alt-özelliği, ISO 9126 standardında aynı isimli özelliğin altında yer alırken ISO 25010'da, Kullanılabilirlik kalite özelliği altında Uygunluğun Fark Edilebilirliği (appropriateness recognizability) alt-özelliği olarak yer almaktadır.

- Kararlılık (stability) kalite alt-özelliği, ISO 9126 standardında Bakım-yapılabilirlik (maintainability) kalite

özelliği altında yer alırken ISO 25010'da yer almamaktadır. Güncel standartta bakım-yapılabilirlik kalite özelliği altındaki Değişebilirlik (modifiability) kalite alt-özelliği, ISO 9126'nın kararlılık kalite alt-özelliğine denk olarak düşünülebilir.

- Taşınabilirlik (portability) kalite alt-özelliği, ISO 9126 standardında aynı isimli özelliğin altında yer alırken ISO 25010'da taşınabilirlik, üç kalite alt-özelliğinden oluşmaktadır. Güncel standartta taşınabilirlik kalite özelliği altındaki Adapte edilebilirlik (adaptability) kalite alt-özelliği, ISO 9126'nın taşınabilirlik kalite alt-özelliğine denk olarak düşünülebilir.

Yukarıdakilere ek olarak, literatürde LCOM metriğinin farklı çalışmalarda uyumluluk özelliğini yeterli ölçemediği iddia edilmiş, onun yerine TCC (Tight Class Cohesion) metriği önerilmiştir [33, 34]. Bununla birlikte bu çalışmada yine de uyumluluk kalite özelliğinin analizinde, literatürdeki ilişkili çalışmalarda yer alması ve C&K metrik seti ile tutarlı olması açısından LCOM metriği kullanılmıştır.

3. İLİŞKİLİ ÇALIŞMALAR (RELATED WORKS)

Belady ve Lehman, ilk olarak OS/360'ın evrimi hakkındaki çalışmalarında, yazılım gelişimine ilişkin deneysel bir çalışmaya öncülük etmişlerdir [12]. Çalışmada, yazılım geliştirme yasaları olarak sonraki araştırmalarında [12, 13, 35] formüle edilen sistemin, boyut ve karmaşıklık büyümesi hakkında birtakım gözlemlerde bulunmuşlardır ve bu yasalar, yazılımın evrim sürecindeki karakterini ve özelliklerini tanımlamaktadır. Lehman vd. yaptıkları başka bir çalışmada [36], daha önceki OS/360 çalışmasında elde edilen sonuçlarla bir finansal ticaret sistemi çalışma sonuçlarını karşılaştırarak yeni analiz sonuçlarının, yazılım geliştirme yasalarını kısmi olarak destekler nitelikte olduğu sonucuna varmışlardır (5. ve 7. yasaları doğrulamak için yeterli veri bulamamışlardır). Bu deneysel çalışmada incelenen 'Azalan kalite' yasasına ilişkin hipotez sonuçlarına bakıldığında Lehman'ın yasalarıyla tutarlı olmayan bir eğilim gözlemlenmiş; mobil uygulamaların kalitesinin arttığı, daha sonra azaldığı ve bu eğilimin tekrarladığı görülmüştür. Bu yasaları daha popüler ve yaygın kabul gören bir hale getirmek için yazarlar, yapmış oldukları iki çalışmada daha ayrıntılı kurallar, araçlar ve uygulamalar ileri sürmüşlerdir [36, 37].

Kemerervd. [38], yazılım karmaşıklığı ve yazılım bakımı arasındaki ilişkiyi araştırmış ve McCabe döngüsel karmaşıklığını (McCabe's Cyclomatic Complexity – MCC) LOC'a normalleştirmenin yazılım bakım üretkenliğinin anlaşılması açısından iyi bir metrik olduğunu fark etmişlerdir. Yutao vd. [39] nesneye yönelik büyük yazılım projelerinin karmaşıklığının hesaplanmasında WMC, RFC, LCOM, DIT, NOC metriklerini ölçmüşler ve ölçüm sonuçlarına göre yazılım karmaşıklıklarına ilişkin ölçümlerin, tasarım kalitesi hakkında bilgi sağlayacağını ileri sürmüşlerdir. Başka bir çalışmada Subramanyam ve Krishnan [40], yazılım hatalarını belirlemede nesneye yönelik karmaşıklık metriklerinin (WMC, CBO, DIT) önemini deneysel çalışmalarıyla kanıtlamışlar ve yazılım kalitesini değerlendirmede nesneye yönelik karmaşıklık metriklerinin geçerlenmesine dair önemli sonuçlar elde etmişlerdir. Basili vd. [24] yaptıkları bir çalışmada ise karmaşık yazılımların daha fazla hataya yatkın (fault-prone) olduklarını, erken aşamalarda yazılımın kalitesinin öngörülebileceğini savunmuşlar ve yaptıkları deneysel çalışma sonucunda C&K metrik setinde mevcut olan beş metriğin (WMC, NOC, DIT, RFC, CBO) karmaşıklığı ölçmede iyi birer metrik olacağını gözlemlemişlerdir. Mevcut çalışmada da, yapılan keşifsel gözlemler ve araştırmalar ışığında aynı beş C&K metriği karmaşıklık hipotezlerini test etmek üzere kullanılmıştır.

Yukarıda verilen yazılım evrimi üzerine çalışmaların çoğu [12, 13, 35-38] ticari sistemler üzerinde yoğunlaşırken açık kaynak yazılım (AKY) ile ilgili ilk çalışmalardan biri Godfrey ve Tu [41] tarafından gerçekleştirilmiştir. Yazarlar ilk olarak ticari bir sistemin açık kaynak bir yazılım ile olan gelişim sürecindeki farkı belirlemeye yönelik araştırmalar yapmışlar, daha sonra Linux çekirdeğinin 1994 ve 1999 yılları arasındaki gelişimini incelemişler 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 vd. [42], Linux modülleri arasındaki bağımlılığın ve karmaşıklığın üssel olarak arttığını ve bakım yapılabilirliğin zamanla azaldığını gözlemlemişlerdir. Wang vd. [43], 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.

Godfrey ve Tu'a göre [41], 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 vd. [10] 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 [8] çalışmalarında, mobil uygulamaların yapısal ve tarihsel bir perspektiften anlaşılmasına yönelik yeni bir yaklaşım sunmuştur. Örneğin, hazır metot

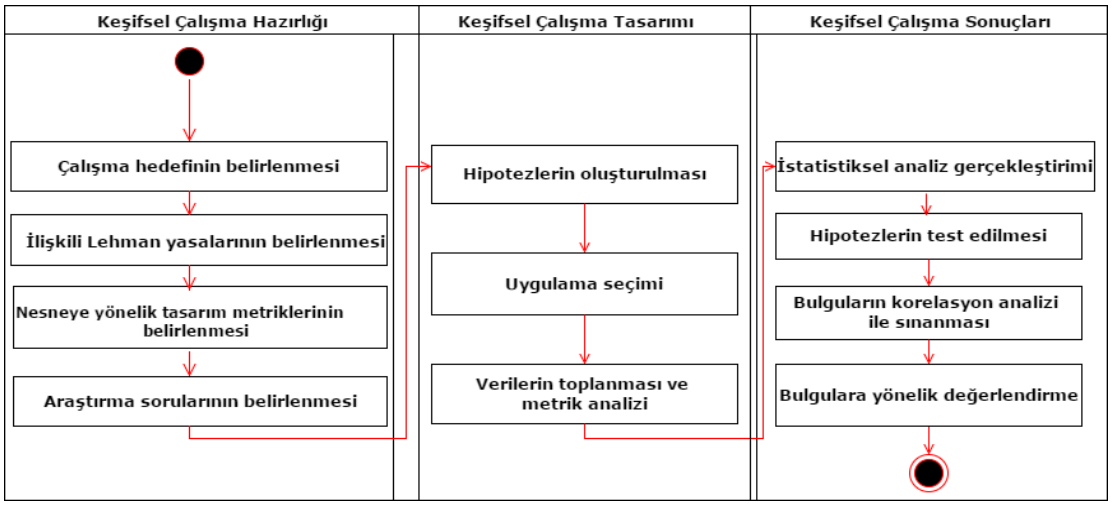
çağrılarının eklenmesiyle uygulama boyutlarını ve karmaşıklığını ilişkilendirmişlerdir ve incelenen mobil uygulamaların, geleneksel yazılımlardan önemli ölçüde farklılık taşıdığını savunmuşlardır. Harman vd. [44], veri madenciliğini Blackberry uygulama mağazasındaki mobil uygulamaların özellik bilgilerini almak için kullanmışlar ve kullanıcı oyları ile indirme sırası arasında bir korelasyon olduğunu gözlemlemişlerdir. Hecht vd. [45], Android uygulamalarının evrimleri boyunca yazılım kalitesini izlemek için deneysel bir çalışma yapmışlar ve negatif örüntülere dayalı mobil uygulama kalitesinin gelişimini izlemek için bir yaklaşım önermişlerdir. Pagano ve Maalej [46] 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. Bir başka çalışmada ise Li vd. [14] mobil uygulamaların evriminde Lehman'ın sekiz yasasının geçerliliğini, sekiz açık kaynak kodlu mobil uygulamanın 348 sürümü üzerinde çeşitli hipotez testleri gerçekleştirerek test etmişlerdir. '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. Bu çalışmada, daha önceki çalışmalardan farklı olarak, benzer işlevselliğe sahip üç mobil uygulamanın 61 sürümü alınmış ve her bir yasa için farklı perspektiflerden incelemeler yapmak üzere farklı hipotezler oluşturulmuştur. Bu hipotezleri test etmek için ise özellikle C&K metrik seti kaliteyi değerlendirmek üzere kullanılmıştır. Tablo 4'te ilişkili çalışmalarda kullanılan metrikler ile bu çalışmada kullanılan metriklerin özetine yer verilmiştir.

4. KEŞİFSEL ÇALIŞMA TASARIMI (EXPLORATORY STUDY DESIGN)

Yaptığımız çalışmanın tasarımı; keşifsel çalışma hazırlığı, tasarım aşaması ve keşifsel çalışma sonuçları olmak üzere, üç adımdan oluşmaktadır. Çalışmada uygulanan yönteme ilişkin adımlar Şekil 2'de özetlenmiştir. Keşifsel çalışma hazırlığına ait adımlardan önceki bölümlerde bahsedilmiştir. Keşifsel çalışma tasarımına ilişkin adımlar bu bölümde ve keşifsel çalışma sonuçları ise 5. bölümde verilmektedir.

Tablo 4. İlişkili çalışmalarda ve bu çalışmada kullanılan metrikler (Metrics used in previous works and in current study)

Yasa	İlişkili Kaynak Kodu Özelliği	İlişkili Kalite Özelliği	Önceki Çalışmalarda Kullanılan Metrikler	Çalışmamızda Kullanılan Metrikler
1. Artan karmaşıklık (Lehman 2)	-Karmaşıklık -Bağımlılık		LOC [10,47], WMC [14], CBO [15], MCC [14, 15]	NOC, DIT, RFC, WMC, CBO
2. Sürekli büyüme (Lehman 6)	-Büyüklük		LOC [15, 41, 48, 49], Modül sayısı [13, 15], Fonksiyon sayısı [49], Sınıf sayısı [14]	LOC, Sınıf sayısı
3. Azalan kalite (Lehman 7)		-Uyumsuzluk -Kararsızlık -Taşınabilirlik -Anlaşılabilirlik	LCOM [14], Kararsızlık [14], Hata sayısı ve yoğunluğu [15], Çözülen hataların sayısı [10, 15]	LCOM, Ca, Ce, CLOC, DIT, CBO, RFC, metot sayısı



Şekil 2. Çalışmada uygulanan yöntemin adımları (Steps of the method applied in the study)

4.1. Test Edilecek Hipotezler (Hypothesis to be tested)

Araştırma sorularına bağlı olarak ve Lehman'ın üç yasasını test etmek üzere, Tablo 5'teki hipotezler oluşturulmuştur. Hedeflenen araştırma sorularına cevap bulabilmek ve hipotezlerin doğruluğunu araştırmak için C&K metrik seti [17] ve bazı geleneksel metrikler (örn. kod satır sayısı) kullanılmıştır.

4.2. Mobil Uygulama Seçimi (Mobile Application Selection)

Keşifsel çalışmayı gerçekleştirmek üzere Google Play marketten 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 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 kalite gelişimi inceleneceğinden, farklı işlevselliğin çalışma üzerindeki olası etkilerini azaltmak adına, benzer fonksiyonelliğe sahip uygulamalar seçilmelidir.
- *Sürüm çeşitliliği.* Uygulamaların sürümleri boyunca iç 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. Keepassdroid ve PasswdSafe uygulamalarının son 5 yıldaki tüm sürümleri, UPM uygulamasının ise tüm (16) sürümleri çalışmada analiz edilmek üzere seçilmiştir. Uygulamalara ait kısa bir özet izleyen paragraflarda verilmiştir. Uygulamalar hakkında genel bilgiler ise Tablo 6'da verilmiştir.

- Keepassdroid, rastgele şifreler üretebilen, tüm şifreleri güvenli bir veri tabanında saklayabilen, farklı şifre yönetimini sınıflandıran ve bunları yönetebilen mobil cihazlar için şifre yönetimi aracıdır. Yazılımın en son sürümü 2.0.6.4, kaynak kodu boyutu 2,5 MB'tır.
- UPM, kullanıcıların tüm giriş bilgilerini tek bir yerde tutmak, böylece onları unutmamak için kullanılabilen bir uygulamadır. UPM, bir ana parola ile korunan ve AES ile şifrelenmiş bir veri tabanında tüm kullanıcı adlarının, şifrelerinin, URL'lerin ve notların saklanmasına izin verir. Yazılımın en son sürümü 1.15 olup kaynak kod boyutu 574 KB'tır.
- PasswdSafe, Android için PasswdSafe (http://pwsafe.org) masaüstü uygulamasının bir gerçekleştirimidir. Parola dosyaları, PasswdSafe Sync uygulamasını kullanarak bulut servislerinden senkronize edilebilir. Yazılımın en son sürümü 6.10.0 olup kaynak kod boyutu 7,8 MB'tır.

4.3. Metrik Verilerinin Toplanması ve Analizi (Metric Data Collection and Analysis)

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

Tablo 5. Test edilecek hipotezler (Hypotheses to be tested)

Yasa 1. Artan Karmaşıklık (Lehman 2)	Hipotez 1a: Alt sınıf sayısı (NOC) zamanla artar. Hipotez 1b: Kalıtım ağacının derinliği (DIT) zamanla artar Hipotez 1c: Sınıfın tetiklediği metot sayısı (RFC) zamanla artar. Hipotez 1d: Sınıfın ağırlıklı metot sayısı (WMC) zamanla artar. Hipotez 1e: Sınıflar arasındaki bağımlılık (CBO) zamanla artar.
Yasa 2. Sürekli Büyüme (Lehman 6)	Hipotez 2a: Uygulamaların kod satır sayısı (LOC) zamanla artar Hipotez 2b: Uygulamaların sınıf sayısı zamanla artar.
Yasa 3. Azalan Kalite (Lehman 7)	Hipotez 3a: Uygulamaların uyumsuzluğu zaman içinde artar. Hipotez 3b: Uygulamaların kararsızlığı zamanla artar. Hipotez 3c: Uygulamaların taşınabilirliği zamanla azalır. Hipotez 3d: Uygulamaların anlaşılabilirliği zamanla azalır.

Tablo 6. Seçilen mobil uygulamalara ait bilgiler (Information on selected mobile applications)

				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	26.01.2009	4226	2.0.6.4	04.10.2016	33090
UPM	Java	https://github.com/adrian/ upm-android	16	1.0	01.02.2010	1345	1.15	13.06.2016	4340
Passwd-Safe	Java	https://sourceforge.net/pro jects/passwdsafe/	92	1.0.0	03.01.2010	5728	6.10.0	29.07.2017	45096

çıkarılmıştır. C&K metrikleri sınıf temellidir ve metrik değerleri, üzerinde medyan hesaplanarak yazılım sistemi (proje) seviyesinde elde edilmiştir. Örneğin, bütün bir yazılım sistemi için bir NOC değerinin hesaplanmasında, elde edilen metrikler sıralı ölçekte (ordinal scale) olduğundan, medyan değerleri üzerinden analizler gerçekleştirilmiştir. 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. 1) kullanılarak normalize edilmiş ve tüm değerler 0 ile 1 arasındaki bir değere dönüştürülmüştür.

$$X_{inorm} = X_i - X_{min} \,/\, X_{max} - X_{min} \quad (1)$$

Çalışma kapsamında incelenen hipotezlerin doğrulanması için elde edilen grafikler, normalize edilmiş metrik değerleri kullanılarak oluşturulmuştur. 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 kararlaştırmak için, çıktılara bakarak elde edilen değerlerin karşılaştırılmasına dayanan yorumlama kurallarını önerir [50]. Mevcut 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. KEŞİFSEL ÇALIŞMANIN SONUÇLARI (EXPLORATORY STUDY RESULTS)

5.1. AS1 için Elde Edilen Bulgular (Findings for research question 1)

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

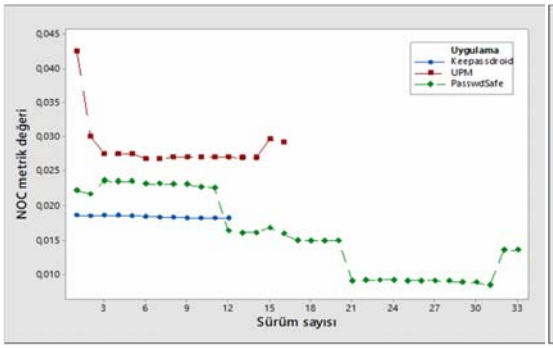
5.1.1. Karmaşıklığın değişimi (Evolution of complexity)

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 kullanmış, Godfrey ve Tu [41] yazılımın karmaşıklığını ve bağımlılığını ölçmede WMC, CBO ve RFC metriklerinin etkinliğini kanıtlarken Li vd. [14], WMC metriğini karmaşıklığı ölçmede önermiştir. Li vd. [14] ile Xie vd. [15] çalışmalarında karmaşıklığı ölçmede ortalama MCC karmaşıklığının kullanılması önerilmiştir. Ayrıca, Newhook [47] 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 (Şekil 3). 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 3a). Bu durum yazılım karmaşıklığının olumlu yönde geliştiği şeklinde yorumlanmaktadır. DIT metriğinin bulgularına göre, Keepassdroid ve PasswdSafe'de değerlerin benzer seyir gösterdiği, UPM için ise son sürümlere doğru artış gösterdiği gözlenmektedir (Şekil 3b). Ağaçların derinliği, daha fazla metot ve sınıf içereceği için, tasarımı daha karmaşık hale getirmektedir; bu sebeple DIT değerinin düşük olması istenmektedir. Şekil 3'deki bulgular DIT metriğine göre incelendiğinde; Keepassdroid'in son sürümlerine doğru, PasswdSafe'in ise ilk sürümlerinde karmaşıklığın daha az olduğu gözlenmektedir. Üç uygulama için de son sürümlere doğru RFC değeri azalan bir seyir gösterirken Şekil 3c'deki RFC değeri bulgularına göre, sınıfın testi ve hata ayıklaması son sürümlere doğru daha az karmaşık hale gelmektedir. PasswdSafe uygulamasının 5.4.0 sürümü ile birlikte sınıf sayısındaki çok hızlı bir büyüme RFC değerinin büyük olmasına yol açmaktadır. WMC değeri, üç uygulamanın son sürümlerine doğru azalmış olup bu durum yazılım karmaşıklığının olumlu yönde geliştiğini gösterir (Şekil 3d). Bazı dönemlerde Keepassdroid ve PasswdSafe uygulamalarında bu metriğin artış gösterip tekrar azaldığı görülmektedir. Bu da o sürümler boyunca sınıf sayısının artmasına rağmen yazılımdaki metot sayısında değişimin gözlenmemesi ile açıklanabilir. CBO metriği bulgularına göre, incelenen üç yazılım için de değerlerin son sürümlere doğru arttığı ve bağımlılık metriğinin ilk sürümden sonra azalıp sınıf sayılarının artmasına bağlı olarak bazı sürümler boyunca arttığı gözlenmektedir (Şekil 3e).

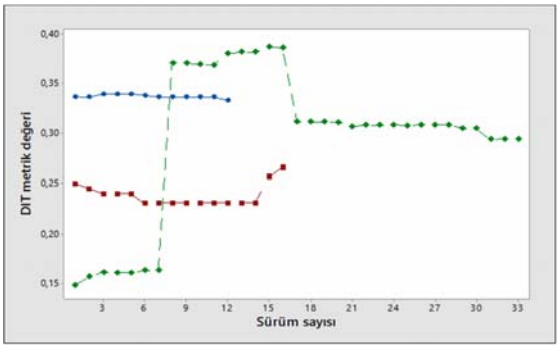
Sonuç olarak, karmaşıklığın genelde son sürümlere doğru azaldığı, bağımlılığın az da olsa bir artış gösterdiği gözlenmiştir.

5.1.2. Boyut değişimi (Evolution of size)

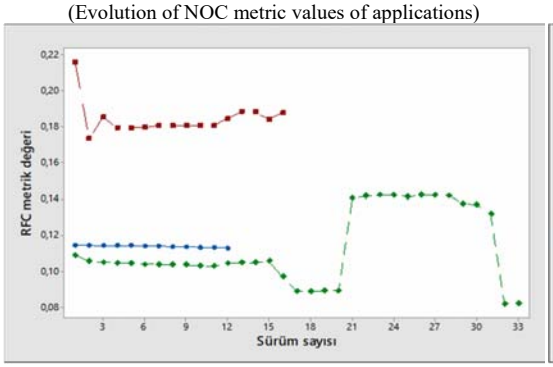
Performans iyileştirmesi ve yeni platformlara uyum sağlaması gerekliliği nedeniyle gereksinimler artmaktadır, buna bağlı olarak da yazılımın işlevi ve içeriği sürekli olarak artmakta ve sonuçta sistemin boyutu artmaktadır. Daha önceki çalışmalarda [12, 15, 51, 52] araştırmacılar, sistem boyutu için çeşitli metrikler kullanmışlarsa da çoğunlukla büyüklük ve fonksiyonellik metriklerini tercih etmişlerdir. Lehman, yazılım boyutunu ölçmek için sistemin modül sayısını kullanırken, Godfrey [41] ve Gonzalez [48] boyut metrikleri olarak kod satır sayısını, Israeli [49] 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 4'de sürekli büyüme yasasına ilişkin analiz edilen metriklerin değişimi verilmiştir.



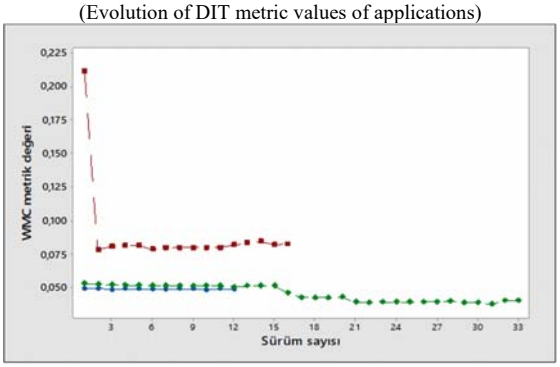
a) Uygulamaların NOC metrik değerinin değişimi (Evolution of NOC metric values of applications)



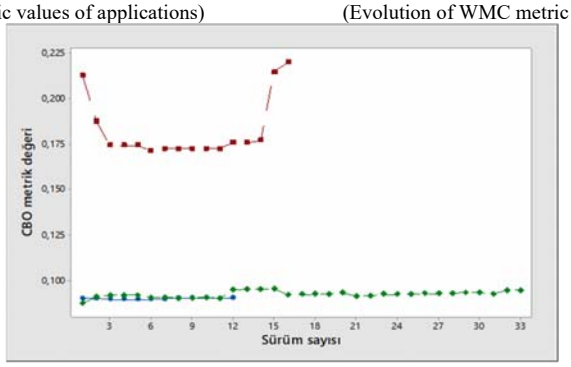
b) Uygulamaların DIT metrik değerinin değişimi (Evolution of DIT metric values of applications)



c) Uygulamaların RFC metrik değerinin değişimi (Evolution of RFC metric values of applications)

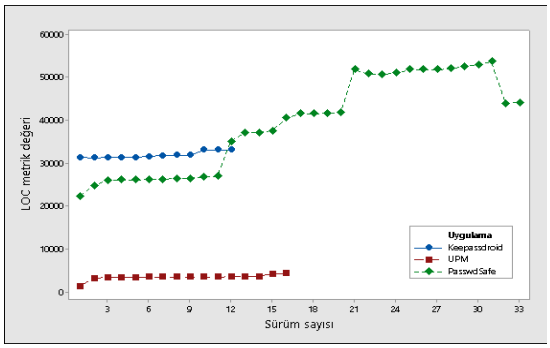


d) Uygulamaların WMC metrik değerinin değişimi (Evolution of WMC metric values of applications)

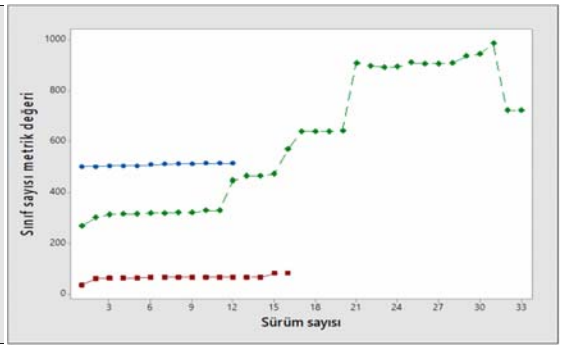


e) Uygulamaların CBO metrik değerinin değişimi (Evolution of CBO metric values of applications)

Şekil 3. Uygulamaların karmaşıklık değişimi (Evolution of complexity of applications)



a) Uygulamaların kod satır sayısının değişimi (Evolution of LOC of applications)



b) Uygulamaların sınıf sayısının değişimi (Evolution of number of classes of applications)

Şekil 4. Uygulamaların boyut gelişimi (Evolution of size of applications)

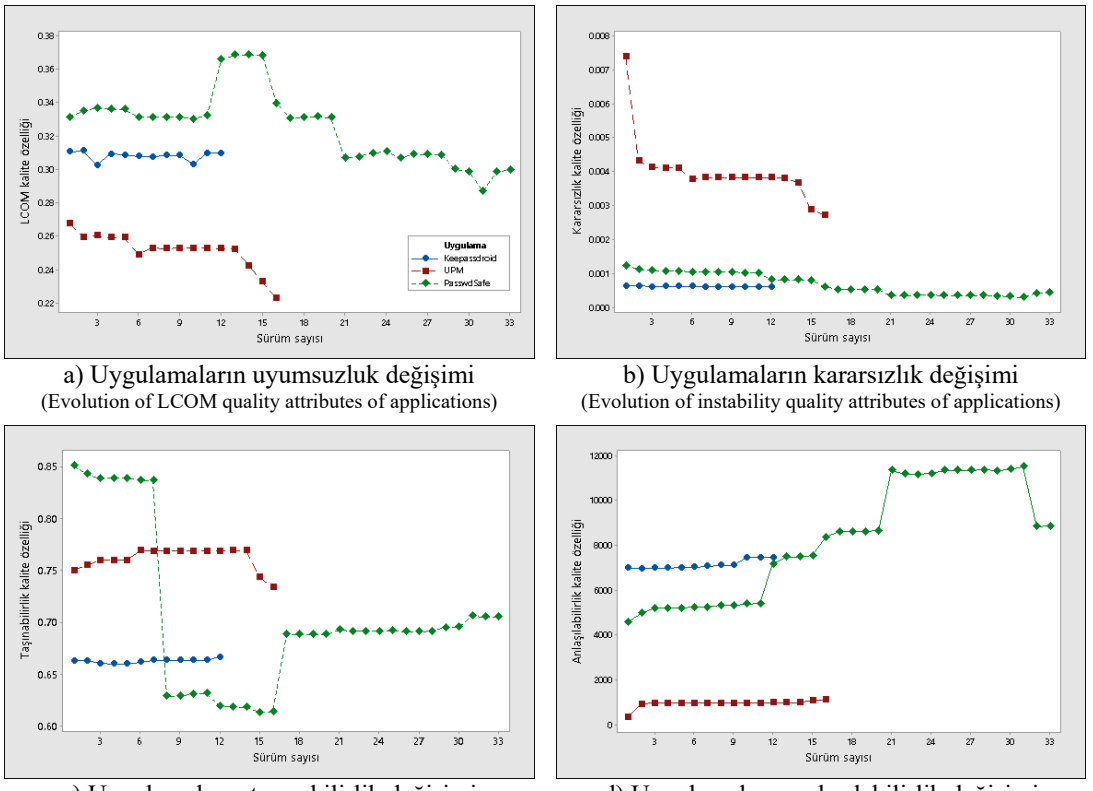
Yapılan analizlere göre tüm uygulamalar, evrimleri boyunca benzer eğilim göstermişlerdir. Şekil 4a ve 4b'de tüm uygulamalar için kod satır sayısına ve sınıf sayısına ait grafikler yer almakta ve sonuçlar, tüm uygulamaların evrim sürecinde büyümeye devam ettiklerini göstermektedir.

5.1.3. Kalite değişimi (Evolution of quality)

Önceki araştırmacılar [12, 14, 15] 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. Xie vd. [15], yazılımın kalitesini değerlendirmek için hata yoğunluğunu (toplam hata/toplam değişim) kullanmayı önermişse de bu yöntemin bazen hatalar rastgele keşfedildiğinden yazılımın kalitesini ölçmede çok doğru bir araç olmayacağı, yazılım sisteminin kalitesinin iç ve dış kalite açılarından değerlendirilmesi gerektiğini belirtmiştir.

Bu çalışmada sadece iç kalite incelenmiş, Gyimothy vd.'nin [52] 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 5'de uyumsuzluk, kararsızlık, anlaşılabilirlik ve taşınabilirlik kalite özelliklerinin sürümler boyunca gelişimleri verilmiştir.

Grafik sonuçlarına göre, her üç uygulamanın belli sürümlerinde, uyumsuzluk ve kararsızlık kalite özelliği değerlerinin azaldığı, daha sonra arttığı, sonra tekrar azaldığı gözlenmektedir (Şekil 5a ve Şekil 5b). Genelde, ilk sürümler yayınlandıktan sonra, uygulamaların hataları sürekli olarak keşfedilmekte, yazılımdaki metot sayısı artmaya devam etmekte ve bu nedenle metotların uyumsuzluk yüzdesi azalmaktadır [14]. 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 [53, 54]. Sürümlerin ilk ortaya çıkış sürecinde paketlerin daha durağan oldukları düşünülürse uygulamaların kararsızlığı başlangıçta azalırken sonrasında, özellikle hazır kütüphanelerin programa dâhil



a) Uygulamaların uyumsuzluk değişimi
(Evolution of LCOM quality attributes of applications)

b) Uygulamaların kararsızlık değişimi
(Evolution of instability quality attributes of applications)

c) Uygulamaların taşınabilirlik değişimi
(Evolution of portability quality attributes of applications)

d) Uygulamaların anlaşılabilirlik değişimi
(Evolution of understandability quality attributes of applications)

Şekil 5. Uygulamaların kalite gelişimi (Evolution of quality of applications)

edilmeleriyle kararsızlığın arttığı gözlenmektedir. Sonuç olarak, uyumsuzluk ve kararsızlık kalite özellikleri için uygulamaların kalitelerinin önce arttığı, sonra azalıp tekrar arttığı gözlenmiştir. Şekil 5c ve 5d'de taşınabilirlik ve anlaşılabilirlik kalite özelliklerine bakılacak olursa üç uygulama için de anlaşılabilirlik benzer bir artan eğilim gösterirken taşınabilirlik özelliği, 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. Yazılımın ilk aşamasında, sınırlı zaman ve zorlu piyasa rekabetinden dolayı ilk sürümlerin kalitesi genellikle düşük olurken uygulamaların hataları keşfedilmeye devam edilip kodun yeniden yapılandırılması sonrasında, kalite giderek artmakta ve süreç bu şekilde devam etmektedir.

5.2. AS2 için Elde Edilen Bulgular
(Findings for research question 2)

Bu bölümde ikinci araştırma sorusuna cevap bulabilmek amacıyla Lehman yasalarının mobil uygulamalara uygulanabilirliğini ve uygulamaların sürümleri boyunca metrik değerlerinin artışını (ya da azalışını) test etmek için Mann-Kendall eğilim (trend) analizi gerçekleştirilmiştir.

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.

Test sonuçlarına göre (Tablo 7) 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}). 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 hipotezlerden hiçbiri geçerli bulunmamıştır.

Tablo 7. Hipotez testlerine ait elde edilen değerler (Obtained values of hypotheses tests)

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.

6. SONUÇLAR VE TARTIŞMALAR
(RESULTS AND DISCUSSIONS)

Önerilen yöntem, ISO 25010 [11] kalite özelliklerini değerlendirmek için kaynak kodu özelliklerini kullandığından bu kaynak kodu özellikleri, belirli bir kalite seviyesi sergileyen sistemlerin gerektirdiği eğilimleri belirtmelidir. Bu eğilimler yazılım evrim yasaları ile ilgilidir [13]. 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 [13]. Önceki çalışmalarda [14, 15] bu yasaya ilişkin oluşturulan hipotezler doğrulanmış olup bu bağlamda, çalışmamızda oluşturulan hipotezler de kod satır sayısı ve sınıf sayısı metrikleri (Hipotez 2a ve Hipotez 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 [10, 14, 15] ve bu çalışmada, Lehman'ın artan karmaşıklık yasasının test edilen tüm uygulamalar için doğrulanmadığı gözlenmiştir (Hipotez 1.e hariç). Bu bağlamda, çalışmada test edilen beş hipotez (Hipotez 1.a, Hipotez 1.b, Hipotez 1.c, Hipotez 1.d, Hipotez 1.e) sonucuyla uyumlu olarak belli sürümler boyunca, karmaşıklığın artıp daha sonra azaldığı çizilen grafiklerde gözlenmiştir. Bu bulgu ise bazı sürümler boyunca, geliştiricilerin tasarım ve kod karmaşıklığını azaltmaya odaklandığı sonucunu doğurmaktadır.

Azalan kalite yasasına ilişkin bulgular, [14, 15] çalışmalarıyla paralellik gösterirken sonuçlar, hipotezlerin tüm uygulamalar için doğrulanmadığını göstermiştir. Oluşturulan hipotezlerin (Hipotez 3.a ve Hipotez 3.b) 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 [13]. 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. 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.

6.1. Keşifsel Çalışma Sonuçlarının Korelasyon Analizi ile Sınanması
(Testing of Exploratory Study Results by Correlation Analysis)

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 Tablo 8'de gösterilmiştir. Her üç uygulama için anlamlılık değerine göre kategorize edilen Tablo 8 verileri, keşifsel çalışmamızın sonuçlarını doğrulamak amacıyla elde edilmiştir.

İlişkileri tespit etmek ve saptanan ilişkilerin anlamlılığını sınamak için, Spearman korelasyon analizi gerçekleştirilmiştir [55]. 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.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 (Tablo 9).

Üç uygulamanın 61 sürümüne ait veriler üzerine yapılan korelasyon analizi sonucunda elde edilen ilişkiler Tablo 9'da 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.

Tablo 8. Yasa 1 ve Yasa 2'ye ilişkin metriklerin kalite özellikleri ile ilişkisi
(Relation between metrics of quality attributes (Law 3) and metrics for Law 1 & Law 2)

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.

Tablo 9. 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, çalışmamızın ikinci bölümünde yer alan Tablo 2'de 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 Geçerliliği Tehdit Eden Faktörler (Threats to validity)

Geçerliliği tehdit eden faktörler dört kategoride incelenmektedir [56]: 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 kararlı ve benzer sonuçlar elde etme yeteneğidir. Bu husus, verilerin ve analizin belirli araştırmacılara ne ölçüde bağımlı olduğu ile ilgilidir. Bu çalışmada kullanılan 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. Ayrıca yapısal tehditleri önleme amacıyla istatistiksel yöntemler kullanarak çeşitli hipotez testleri gerçekleştirilmiştir. 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 sebeple, insan kaynaklı hataların ortaya çıkması büyük ölçüde engellenmiştir. Ancak, seçilen uygulamaların tüm sürümleri analiz edilmediğinden, sürüm seçimi bu açıdan bir kısıt olarak görülebilir. 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. Ek olarak, istatistiksel analizlerde teste tabi tutulan örneklem sayısı da dış geçerliliğe tehdit oluşturmuş olabilir.

Çalışmanın güvenilirliği ile ilgili olarak, izlenen adımların detaylı açıklanması, oluşturulan modelin istatistiksel yöntemlerle incelenip ölçümlerin otomatik bir araçla gerçekleşmesi, bu çalışmanın farklı araştırmacılar tarafından da tekrarlanabileceğini güvence etmektedir. Ne var ki bulguların yorumlanmasında araştırmacı etkisi ve analiz edilen uygulamaların seçimi, çalışmanın güvenilirliğine tehdit oluşturmuş olabilir. Güvenilirliğe yönelik tehditleri azaltmak adına, korelasyon analizi ile keşifsel çalışmanın sonuçları doğrulanmış ve sonuçlar daha somut verilerle yorumlanmıştır.

7. SONUÇLAR (CONCLUSIONS)

Bu çalışmada, Lehman'ın yasalarından yola çıkarak oluşturulmuş hipotezleri doğrulamak üzere kullanılan metrikler ile mobil uygulamaların nasıl geliştiği gözlemlenmiştir. Metrikler yazılım kalitesinin belirlenmesi ve iyileştirilmesi çalışmalarında etkili biçimde kullanılmaktadır. Bu sayede aşırı bağımlı, karmaşık ve hataya eğilimli modüllerin belirlenmesi gibi önemli bilgiler elde edilebilmektedir. Bu bilgiler yazılım kalitesinin iyileştirilmesinde, daha sonra hangi kısımların öncelikli olarak test edileceğine karar verilmesinde, bakım için gereken bütçe ve zaman analizlerinde kullanılabilmektedir. Ancak, çalışmaların başarımı büyük ölçüde doğru metriklerin tanımlanmasına, bu metriklerin doğru biçimde ölçülmesine ve doğru yorumlanmasına bağlıdır.

Çalışma kapsamında, üç açık-kaynak mobil yazılımın evrimi boyunca karmaşıklık, boyut ve iç kalite değişiminin görülmesi amaçlanmıştır. Bunun için, Lehman'ın yazılımın evrimiyle ilgili yasalarından yola çıkarak 11 hipotez oluşturulmuş, bu hipotezlerin doğruluğunun test edilmesi için 11 metrik kullanılmış ve ISO 25010 kalite standardı temel alınarak uygulamaların sürümleri boyunca karmaşıklık, boyut ve iç kalitelerinin ne yönde değiştiği üzerine gözlemlerde bulunulmuştur. Birinci araştırma sorusu (AS 1), uygulamaların evrimi boyunca karmaşıklık, boyut ve iç kalite değişimlerinin analizine odaklanmıştır. Karmaşıklık değişimi için incelenen metriklerin analizi ile sürümler boyunca karmaşıklığın genel olarak sürekli artıp azalan bir eğilim gösterdiği gözlenmiştir. Boyut değişimi için analiz edilen metrikler sonucunda, üç uygulama için de benzer bir eğilim gözlenmiş olup uygulamaların son sürümlere doğru büyüdüğü tespit edilmiştir. Kalite değişimi için incelenen metriklerin analizi ile uygulamaların kalitelerinin önce azaldığı fakat son sürümlere doğru artan bir eğilim gösterdikleri saptanmış, bu ise tüm uygulamaların son sürümlerinin genelde daha anlaşılabilir, kararlı ve taşınabilir olduğu şeklinde yorumlanmıştır. Ayrıca, uygulamaların uyumsuzluk ve kararsızlık kalite özelliği değerlerinin son sürümlere doğru azalması, anlaşılabilirlik ve taşınabilirlik kalite özelliği değerlerinin ise son sürümlere doğru artması, genel olarak, kalitenin arttığı yönünde yorumlanmıştır. İkinci araştırma sorusu (AS 2) ile Lehman yasalarının (artan karmaşıklık, sürekli büyüme, azalan kalite) mobil uygulamalar için geçerliliği incelenmiştir. Yapılan istatistiksel analizler sonucunda, karmaşıklığın artması ve azalan kalite yasalarından yola çıkarak oluşturulan hipotezlerin tüm uygulamalar için doğrulanmadığı gözlenmiş ve sürekli büyüme yasası için oluşturulan hipotezlerin doğrulandığı sonucuna varılmıştır. Son olarak, tasarım metrikleri ile kalite özellikleri arasındaki ilişkiyi sınamak üzere korelasyon analizi gerçekleştirilmiş ve bu analizin, keşifsel çalışma sonuçlarını doğrular nitelikte olduğu tespit edilmiştir.

Bu çalışma ile mobil uygulamaların evriminde karmaşıklık, boyut ve iç kalitenin gelişimi keşifsel olarak incelenmiştir. Çalışmanın, mobil yazılımların gelişimi ve bakımı üzerine yapılacak benzer çalışmalara rehber olması beklenebilir. Bu çalışma devam eden bir çalışma olup gelecek çalışmalarda, mobil uygulamalar için elde edilen iç kalite bulguları ile dış kalite (market başarısı) arasındaki ilişkinin araştırılması hedeflenmektedir.

KAYNAKLAR (REFERENCES)

1. Silva D. B., Eler M. M., Durelli V. H. S. ve Endo A. T., Characterizing mobile apps from a source and test code viewpoint, Inf. Softw. Technol., Mayıs, 32–50, 2018.

2. Utku A. ve Doğru İ.A., Permission based detection system for android malware, Journal of the Faculty of Engineering and Architecture of Gazi University, 32 (4), 1015-1024, 2017.

3. Statista. Number of Apps Available in Leading App Stores as of July 2014, http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/. 2014, Yayın tarihi 2014. Erişim tarihi Mart, 2018.

4. Koskinen J. ve Tilus T., Software maintenance cost estimation and modernization support, 62, 2003.

5. Seacord R. C., Plakosh D. ve Lewis G. A., Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices, Addison-Wesley Professional, 2003.

6. Ghezzi C., Jazayeri M. ve Mandrioli D., Software Engineering Principles, in Fundamentals of Software Engineering, 5961, 2002.

7. Sommerville I., Verification and validation, Software Engineering (8th Edition), Addison Wesley, United Kingdom, 2006.

8. Minelli R. ve Lanza M., Software analytics for mobile applications - Insights & lessons learned, in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 144–153, 2013.

9. Islam R. ve Mazumder T. A., Mobile Application and Its Global Impact., Int. J. Eng. Technol., 104–111, 2010.

10. Zhang J., Sagar S. ve Shihab E., The evolution of mobile apps: an exploratory study, Proceedings of the 2013 International Workshop on Software Development Lifecycle for Mobile. ACM, 1–8, 2013.

11. ISO/IEC, ISO/IEC 25010:2011 - Systems and software quality requirements and evaluation (SQuaRE) - System and software quality models, International Standard ISOIEC 25010 (2011), 1–25, 2011.

12. Belady L. A. ve Lehman M. M., A model of large program development, IBM Syst. J., 225–252, 1976.

13. Lehman M. M., Programs, Life Cycles, and Laws of Software Evolution, Proc. IEEE, 1060–1076, 1980.

14. Li D., Guo B., Shen Y., Li J. ve Huang Y., The evolution of open-source mobile applications: An empirical study, J. Softw. Evol. Process, 2017.

15. Xie G., Chen J. ve Neamtiu I., Towards a better understanding of software evolution: An empirical study on open source software, Int. Conf. Softw. Maint., 51–60, 2009.

16. Easterbrook S., Singer J., Storey M.-A. ve Damian D., Selecting empirical methods for software engineering research, in Guide to Advanced Empirical Software Engineering, 285–311, 2008.

17. Chidamber S. R. ve Kemerer C. F., A Metrics Suite for Object Oriented Design, IEEE Trans. Softw. Eng., 476–493, 1994.

18. Li W. ve Henry S., Maintenance metrics for the object oriented paradigm, Proc. First Int. Softw. Metrics Symp., 52–60, 1993.

19. Lee Y.S., Liang B.S. ve Wang F.L., Some complexity metrics for object-oriented programs based on information flow: a study of C++ programs, J. Inf. Sci. Eng., 21–50, 1994.

20. Lorenz M. ve Kidd J., Object-Oriented Software Metrics, J. Syst. Softw., 147–154, 1994.

21. McCabe T. J., A Complexity Measure, IEEE Trans. Softw. Eng., 308–320, 1976.

22. Misra S. ve Bhavsar V., Relationships Between Selected Software Measures and Latent Bug-Density: Guidelines for Improving Quality, Springer-Verlag, 724–732, 2003.

23. Hudli R. V, Hoskins C. L. ve Hudli A. V, Software metrics for object-oriented designs, In Proceedings 1994 IEEE International Conference on Computer Design: VLSI in Computers and Processors, 492-495, IEEE, 1994.

24. Basili V. R., Briand L. C. ve Melo W. L., A validation of object-oriented design metrics as quality indicators, IEEE Trans. Softw. Eng., 751–761, 1996.

25. Olague H. M., Etzkorn L. H., Gholston S. ve Quattlebaum S., 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 Trans. Softw. Eng., 402–419, 2007.

26. Dubey S.K. ve Rana A., Assessment of maintainability metrics for object-oriented software system, ACM SIGSOFT Softw. Eng. Notes, 2011.

27. Singh G., Singh D. ve Singh V., A Study of Software Metrics, IJCEM International Journal of Computational Engineering & Management, 11, 22-27, 2011.

28. Erçelebi Ayyıldız T. ve Koçyiğit A., Correlations between problem and solution domain measures of open source software, Journal of the Faculty of Engineering and Architecture of Gazi University, 32 (3), 887–900, 2017.

29. Pascarella L. ve Bacchelli A., Classifying Code Comments in Java Open-Source Software Systems, in IEEE International Working Conference on Mining Software Repositories, 227–237, 2017.

30. Anwer S., Adbellatif A., Alshayeb M., Anjum M. S., Effect of coupling on software faults: An empirical study, in Proceedings of 2017 International Conference on Communication, Computing and Digital Systems (C-CODE), 211–215, 2017.

31. Martin R.C., Agile Software Development, Principles, Patterns, and Practices, Prentice Hall. 529, 2002.

32. Amin F., Mahmood A. ve Oxley A., Reusability Assessment of Open Source Components for Software Product Lines, International Journal on New Computer Architectures and Their Applications (IJNCAA), 1 (3), 519-533, 2011.

33. Al Dallal J. ve Briand L.C., An object-oriented high-level design-based class cohesion metric, Inf. Softw. Technol., 1346–1361, 2010.

34. Aggarwal K. K., Singh Y., Kaur A. ve Malhotra R., Empirical study of object-oriented metrics, Journal of Object Technology, 5 (8), 149-173, 2006.

35. Lehman M. M., Laws of software evolution revisited, in Lecture Notes in Computer Science (including subseries

Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 108–124, 1996.

36. Lehman M. M. ve Ramil J. F., Rules and Tools for Software Evolution Planning and Management, Annals. Softw. Eng., 15–44, 2001.

37. Lehman M. M. ve Ramil J. F., Software evolution—Background, theory, practice, Information Processing Letters, 88 (1-2), 33-44, 2003.

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

39. Ma Y., He K., Du D., Liu J. ve Yan Y., A complexity metrics set for large-scale object-oriented software systems, in Proceedings - Sixth IEEE International Conference on Computer and Information Technology, CIT 2006, 2006.

40. Subramanyam R. ve Krishnan M. S., Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects, IEEE Trans. Softw. Eng., 297–310, 2003.

41. Godfrey M. W. ve Tu Q., Evolution in open source software: a case study, Proc. Int. Conf. Softw. Maint. ICSM-94, 131–142, 2000.

42. Yu L., Schach S. R., Chen K. ve Offutt J., Categorization of common coupling and its application to the maintainability of the linux kernel, IEEE Proc. Softw., 18–23, 2002.

43. Wang Y., Guo D. ve Shi H., Measuring the evolution of open source software systems with their communities, ACM SIGSOFT Softw. Eng. Notes, 7, 2007.

44. Harman M., Jia Y. ve Zhang Y., App store mining and analysis: MSR for app stores, in IEEE International Working Conference on Mining Software Repositories, 108–111, 2012.

45. Hecht G., Benomar O., Rouvoy R., Moha N. ve Duchien L., Tracking the software quality of android applications along their evolution, in Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, 236–247, 2016.

46. Pagano D. ve Maalej W., User feedback in the appstore: An empirical study, In 2013 21st IEEE international requirements engineering conference (RE), 125-134, IEEE, 2013.

47. Newhook R., Jaramillo D., Temple, J. G. ve Duke K. J., Evolution of the Mobile Enterprise App: A Design Perspective, Procedia Manuf., 3, 2026–2033, 2015.

48. Gonzalez-Barahona J. M., Robles G., Herraiz I. ve Ortega F., Studying the laws of software evolution in a long-lived FLOSS project, J. Softw. Evol. Process, 589–612, 2014.

49. Israeli A. ve Feitelson D. G., The Linux kernel as a case study in software evolution, J. Syst. Softw., 485–501, 2010.

50. Rosenberg L. Dr., Hammer T. ve Shaw J., Software metrics and reliability, Proc. 9th Int. Symp. Softw. Reliab. Eng., 1–8, 1998.

51. Syer M. D., Nagappan M., Adams B. ve Hassan A. E., Studying the relationship between source code quality and mobile platform dependence, Softw. Qual. J., 485–508, 2015.

52. Gyimothy T., Ferenc R. ve Siket I., Empirical validation of object-oriented metrics on open source software for fault prediction, IEEE Trans. Softw. Eng., 897–910, 2005.

53. Linares-Vásquez M., Bavota G., Bernal-Cárdenas C., Penta M. Di, Oliveto R. ve Poshyvanyk D., 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 - ESEC/FSE 2013, 477, 2013.

54. McDonnell T., Ray B. ve Kim M., An empirical study of API stability and adoption in the android ecosystem, in IEEE International Conference on Software Maintenance, ICSM, 70–79, 2013.

55. Kılıç O. ve Çerçioğlu H., Application of compromise multiple criteria decision making methods for evaluation of TCDD's railway lines projects, Journal of the Faculty of Engineering and Architecture of Gazi University, (31) 1, 211-220, 2017.

56. Runeson P. ve Höst M., Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2), 131–164, 2009.