Uygulamaların mobil ve masaüstü sürümlerinin kod-tabanlı karşılaştırılması: keşifsel bir çalışma

Sena Sönmez Çiçek[1], Vahid Garousi[2,1], Ayça Tarhan[1]

[1] Yazılım Mühendisliği Araştırma Grubu,
Bilgisayar Mühendisliği Bölümü, Hacettepe Üniversitesi, Ankara, Türkiye
[2] Wageningen Üniversitesi, Hollanda

senasonmezcicek@gmail.com, vahid@vgarousi.com,
atarhan@hacettepe.edu.tr

Özet. Belirli bir yazılım platformunun özellikleri, genellikle, bu platformda geliştirilen uygulamaların tasarımını ve kaynak kodlarını derin bir şekilde etkileye biliyor. Literatürde, Android, iOS ve BlackBerry gibi mobil platformların ve uygulamaların çeşitli özellikleri üzerinde birçok araştırma yapılmıştır, ne var ki verilen bir uygulamanın mobil sürümlerine karşı masaüstü sürümlerinin özelliklerini değerlendiren ve karşılaştıran az sayıda araştırma bulunmaktadır. Bu çalışmada, platformların uygulamaların mobil ve masaüstü sürümleri üzerindeki etkilerini ortaya çıkarmak için bir keşifsel çalışma yaparak iki araştırma sorusunu cevaplamayı hedefledik: (1) Aynı özelliklerde bir uygulama, farklı platformlar için geliştirilirken kaynak kodunun büyüklüğü nasıl değişir? (2) Mobil ve masaüstü uygulamalarının kod tasarımları, nesne-tabanlı tasarım ölçütleri açısından birbirinden nasıl farklılık gösterir? Bu soruları cevaplamak için açık kaynaklı iki uygulamanın mobil ve masaüstü sürümlerinin kaynak kodlarını inceledik. Uygulamaları, kod ölçüt aracı kullanarak değerlerine ulaştığımız tasarım ölçütlerinin değerleri üzerinden karşılaştırılmalı olarak analiz ettik. Analizlerimizin sonucunda uygulamaların mobil sürümlerinin masaüstü sürümlerden kod büyüklüğü ölçütleri açısından daha büyük olduğunu gözlemledik. Tasarım açısından ise, Nesne Sınıfları Arasındaki Bağımlılık (İng. CBO-Coupling Between Object classes) ve Sınıf Başına Ağırlıklı Metot Sayısı (İng. WMC-Weighted Methods per Class) ölçütleri uygulamaların her iki platformu için farklılık gösterirken, Metot İçi Uyumsuzluk (İng. LCOM-Lack of Cohesion in Methods) için önemli bir fark bulunmamıştır.

Anahtar kelimeler: Yazılım platformu, platform etkisi, mobil yazılım, masaüstü yazılımı, yazılım tasarımı, yazılım ölçütleri, deneysel çalışma

Code-based comparison of software applications' mobile and desktop versions: an exploratory study

Abstract. Characteristics of a given software platform usually have major effects on the design and source code of applications developed in that platform. Various characteristics of mobile applications and platforms such as Android, iOS and BlackBerry have been studied in the literature, but few studies have as-

sessed and compared the characteristics of mobile versus desktop versions of a given application. In this paper, we report an exploratory study in order to reveal the effects of platforms on applications' mobile and desktop versions, for which we aimed to address two research questions (RQs): (1) How do the sizes of source code change when implementing the same feature in mobile and desktop platforms?, (2) How do designs of mobile and desktop applications differ from each other in terms of object-oriented metrics? We compare two equivalent features of open-source mobile and desktop applications. We do this by analyzing source code via object-oriented metrics obtained using a popular metrics tool. Our initial results indicate that the mobile versions of the applications are larger than the desktop versions in terms of code size metrics. In terms of design, there is no significant difference for LCOM (Lack of Cohesion in Methods), as CBO (Coupling Between Objects) and WMC (Weighted Methods per Class) metrics differ in both platforms.

Keywords: Software platforms, platform effect, comparison, mobile software, desktop software, software design, software metrics, empirical study

1 Giriş

Günümüzde çapraz platform uygulama geliştirmek diğer bir deyişle birden fazla yazılım platformu için uygulama geliştirmek, piyasa ve kullanıcı beklentilerini karşılamak için bir ihtiyaç haline gelmiştir. Farklı platformlar için birden çok uygulama geliştirmek, gerektirdiği kaynak, zaman ve çaba açısından zorlu bir iştir [1]. Belirli bir yazılım platformunun özellikleri, genellikle, bu platformda geliştirilen uygulamaların mimarisi, tasarımı ve kaynak kodu üzerinde derin bir etkiye sahiptir [2]. Bu nedenle, çapraz platform uygulama geliştirme için geliştirme kararları verirken, platformun etkileri de göz önüne alınmalıdır.

Verilen bir uygulamanın, farklı platformlarda (ör: Android, iOS) nasıl daha verimli ve etkili şekilde geliştirilebileceğine yönelik birçok çözüm bulunmaktadır. Bu alanda, Heitkötter ve arkadaşları [1], mobil uygulamaların çapraz platformda geliştirilmesine yönelik genel yaklaşımları sınıflandırıp mevcut çapraz platform çözümlerini analiz etmiş ve karşılaştırmıştır. Bununla birlikte, mobil ve masaüstü platformlar için platformlar arası geliştirme konusunda böyle bir araştırmaya rastlanmamıştır.

Bu çalışmada, mobil platformun ve masaüstünün, yazılım platformları olarak uygulamanın tasarımını ve kaynak kodunu etkileyip etkilemediğini araştırmayı amaçladık. Bulduğumuz sonuçların, geliştiricilere, mobil ve masaüstü platformlar için geliştirilecek bir uygulamanın tasarım kararları alınırken ve geliştirme çabasının değerlendirilmesi aşamasında yardımcı olmasını umuyoruz.

Belirlediğimiz ve cevabını aradığımız iki araştırma sorusu (AS) şöyledir;

- **AS-1:** Aynı özelliklerde bir uygulama, mobil ve masaüstü platformlar için geliştirilirken kaynak kodunun büyüklüğü nasıl değişir?
- **AS-2:** Mobil ve masaüstü uygulamaların tasarımları, nesneye yönelik tasarım ölçütleri açısından birbirinden nasıl farklılık gösterir?

Uygulamaları kaynak kod boyutu açısından karşılaştırmak için sınıf sayısı, kod satır sayısı ve dosya sayısı ölçütlerini kullandık ve uygulamaların mobil ve masaüstü sürümlerinin karşılaştırmalı analizini yaptık. Öte yandan uygulamaların tasarımlarını Chidamber & Kemerer (C&K) ölçüt kümesinden [3] yararlanarak karşılaştırdık.

Bu makalenin kalanı şu şekilde organize edilmiştir: Bölüm 2, bu alandaki ilgili çalışmaları sunmaktadır. Bölüm 3, amacımızı ve araştırma yöntemimizi, bölüm 4 ise vaka incelememizin sonuçlarını açıklar ve tartışır. Son olarak, bölüm 5, çalışmamızın sonuçlarını ve gelecek çalışma önerilerimizi özetler.

2 İlgili çalışmalar

Birden fazla yazılım platformu için çapraz platform uygulaması geliştirmek ihtiyaç olduğu kadar uğraştırıcı bir sorundur. Tüm platformlar birbirinden önemli derecede farklı olduğu için, geniş bir kullanıcı kitlesine ulaşmak isteyen yazılım geliştiricileri, her bir platform için ayrı olarak uygulamalar geliştirmektedirler. Bu noktada çapraz platform geliştirme yaklaşımları, geliştiricilerin bir dizi platform için harcadıkları çabayı tekrarlamayı önleyerek ve verimliliği artırarak bu zorluğa çözüm üretmek için ortaya çıkmıştır. Özellikle mobil platformlar için, bu gibi çözümlerin pek çok uygulaması vardır. Heitkötter ve arkadaşları, mobil uygulamaların platformlar arası gelişimine genel yaklaşımları sınıflandırıp mevcut çapraz platform çözümlerini analiz etmiş ve karşılaştırmıştır [1].

Öte yandan, Android'e karşı iOS veya Android'e karşı BlackBerry gibi farklı mobil platformların karşılaştırılması, literatürde birçok çalışmaya konu olmuştur. Örneğin, Syer ve arkadaşları Android ve BlackBerry uygulamaları kod tabanlı bir analiz yaparak karşılaştırmıştır [4]. Mikro uygulama geliştiricilerin sınırlı kaynaklarla birden fazla platformu hedefleyebileceklerini düşünerek kaynak kodu, bağımlılık ve kod değişiklik (İng. code churn) metriklerini incelemişlerdir. Üç mobil uygulama çifti üzerinde bir vaka incelemesi yapmış ve Android'in mikro uygulamalarının üzerinde çalıştığı platforma çok fazla bağımlı olduğunu ve BlackBerry mikro uygulamalarına göre daha az kod gerektirdiğini bulmuşlardır.

Ayrıca, Syer ve arkadaşları, 15 popüler açık kaynak mobil uygulama ile 5 masaüstü / sunucu uygulaması (2 büyük uygulama, 3 küçük Unix Yardımcı Programı) arasında kod tabanı, platform bağımlılığı ve hata giderme süresinin boyutlarını incelenmiştir [5]. Mobil uygulamaların ve Unix yardımcı programlarının, sıklıkla incelenen masaüstü / sunucu uygulamalarına kıyasla daha küçük kod tabanlarına sahip olduklarını raporlamışlardır. Ayrıca, mobil uygulamaların üzerinde çalıştığı mobil platforma çok fazla bağımlı olduklarını ve mobil uygulama geliştiricilerin, hataları masaüstü / sunucu uygulamaları geliştiricilerine göre daha kısa sürede düzeltme eğiliminde olduklarını keşfetmişlerdir.

Ayrıntılı tasarım ve uygulama geliştirme, uygulama mimarisinin net bir yansımasıdır ve iç kalite yazılım tasarımının bir çıktısı olarak kabul edilir. Yazılım tasarım düzeyinde iken son ürünün tasarım ve kalite özelliklerini ölçmenin her zaman mümkün olmadığı bilinmektedir [6]. Nesneye yönelik (İng. Object-Oriented OO) kod ve tasarımın yapısal kalitesini değerlendirmek için literatürde birçok ölçüt önerilmiştir [3][7][8][9]. Literatürde önerilen ölçütlerin çoğunu, deneysel olarak araştıran çalışma-

lar bulunmaktadır. Briand ve arkadaşları, nesneye yönelik tasarımın yapısal özelliklerini yakalayan metrikler üzerine bir deneysel (İng. empirical) çalışma yapmıştır [10]. Analiz sonuçları, birçok bağımlılık (İng. coupling) ve kalıtım (İng. inheritance) metriğinin, bir sınıftaki hata tespit olasılığı ile güçlü bir şekilde ilişkili olduğunu göstermiştir. Öte yandan, 2002'deki çalışmalarından elde edilen sonuçlarda[11], bağımlılık, karmaşıklık ve boyut ölçütlerinin, kalıtım ve uyumluluk (İng. cohesion) ölçütlerine göre daha iyi olduğu savunulmuştur. Yazarlar bu savın, kalıtımın etkisinin, kullanılan kalıtım stratejisine ve tasarımcıların deneyimine göre değişiminden kaynaklanabileceğini iddia etmişlerdir. Jabangwe ve arkadaşları, özellikle nesneye yönelik sistemlerin kaynak kodundan elde edilebilen nesneye yönelik metrikleri belirleyip bir araya topladıkları bir sistematik literatür taraması yapmışlardır [12]. 99 adet çalışmayı saptamışlar ve bu araştırmalardan nesne odaklı metrikleri kullananların, çoğunlukla C&K ölçüt kümesini seçtiğini gözlemlemişlerdir. Aynı zamanda C&K ölçüt kümesinin [3] ölçme ve değerlendirmede MOOD (İng. Metrics for Object Oriented Design) [13] ve QMOOD (İng. Quality Model for Object Oriented Design) [14] ölçüt kümelerinden daha iyi performans sergilediklerini ve karmaşıklık, uyumluluk, boyut ve bağımlılık ölçütlerinin, kalıtım ölçütlerine göre yazılımın güvenilirlik ve sürdürülebilirlik özellikleriyle daha bağlantılı olduğunu saptamışlardır.

Bu bilgiler ışığında, çalışmamızda uygulamaların tasarım özelliklerini karşılaştırmak için, nesneye yönelik yazılımlar için önerilmiş altı ölçütten oluşan C&K ölçüt kümesinden [3] yararlandık. Bu kümeden Nesne Sınıfları Arasındaki Bağımlılık (CBO-Coupling Between Object classes), Sınıf Başına Ağırlıklı Metot Sayısı (WMC-Weighted Methods per Class) ve Metot İçi Uyumsuzluk (LCOM-Lack of Cohesion in Methods) ölçütlerini seçtik.

3 Araştırma amacı ve yöntemi

Bu çalışmada, mobil ve masaüstü yazılım geliştirme platformlarının uygulamanın tasarımını ve kaynak kodunu etkileyip etkilemediğini araştırmayı amaçladık. Belirlediğimiz araştırma sorularını, iki mobil-masaüstü uygulama çiftini değerlendirdiğimiz ve karşılaştırdığımız bir vaka çalışması ile yanıtladık.

3.1 Vaka çalışması tasarımı

Bu bölüm, uygulamalar üzerindeki platformların etkisini keşfetme yaklaşımımızı özetlemektedir. Öncelikle, hem mobil sürümü hem de masaüstü sürümü olan iki uygulama çifti seçtik. İkinci olarak, araştırma sorularımıza cevap vermek için uygun ölçütleri belirledik ve bu uygulamaların kaynak kodu, bağımlılık ve tasarım özelliklerini ölçtük. Ardından, sonuçları mobil ve masaüstü sürümleri arasında karşılaştırdık.

Uygulama seçimi. Çalışmamızda farklı platformlar için eşdeğer özelliklere sahip mobil uygulamaları ve masaüstü uygulamaları inceledik. Uygulamalarımızı aşağıdaki kriterlere göre seçtik;

- Açık Kaynak Kodlu Uygulamalar. Kod tabanlı bir araştırma yaptığımız için uygulamaların kod havuzlarına erişmemiz gerekiyordu. Açık kaynaklı uygulamaların GitHub gibi kod depolarından kaynak kodlarına erişilebilmektedir.
- Nesne Tabanlı Programlama Dilleriyle Geliştirilmiş Uygulamalar (Java, C++). Geleneksel nesne odaklı ölçüt kümelerini uygulamak için nesneye yönelik bir dil ile geliştirilmiş uygulamalara ihtiyacımız vardı. Bu nedenle, masaüstü sürümleri C dilinde geliştirilmiş uygulamalar hariç tutulmuştur.
- Uygulama Özellik Denkliği. Mobil ve masaüstü uygulamaları arasında nesnel bir karşılaştırma yapabilmemiz için aynı uygulamanın birbirine denk özellikler içeren mobil ve masaüstü sürümlerinin olması gerekmektedir.

Yukarıdaki kriterlere ek olarak, küçük kod hacmi, az katkıda bulunan ve az sayıda taahhüt içeren uygulamalar hariç tutulmuştur.

Uygulamalarda özellik denkliği ve geliştirme dilinin nesne tabanlı olması kriterlerini aramamız, uygulama havuzumuzu azaltan baskın ölçütlerdir. Bu kriterleri uyguladıktan sonra iki uygulama çifti seçtik: Telegram ve KeePass. Uygulamaların tüm mobil sürümleri Android platform için geliştirilmiştir. Uygulamaların kaynak kodu depolarına Nisan 2017'de ulaştık. Seçilen uygulamaların ayrıntıları Tablo 1'de verilmiştir.

Tablo 1. Seçilen uygulamalar

Uygulama	**Geliştirme Dili**	**Kaynak kod web adresi**
KeePass	C++	https://github.com/keepassx/keepassx
KeePassDroid	Java	https://github.com/bpellin/keepassdroid
Telegram Masaüstü	C++	https://github.com/telegramdesktop/tdesktop
Telegram Mobil	Java	https://github.com/DrKLO/Telegram

Kaynak kod özellikleri. AS-1'i cevaplandırmak adına üç kaynak kod metriği belirledik;

- Kaynak kodu içeren toplam dosya sayısı
- Sınıfların toplam sayısı
- Toplam kod satırı sayısı (İng. Lines of Code - LOC)

Kaynak kod boyutu ölçütlerinin, bir yazılım sisteminin karmaşıklığı ile önemli ölçüde ilişkili olduğu kanıtlanmıştır [15]. Bir sistemin karmaşıklığı ve boyutu, bir yazılımı anlamak, geliştirmek ve bakımını yapmak için gereken çabayı ölçmeye yardımcı olur. Bu çalışmada bir uygulamada aynı özelliklerin gerçeklenmesi için hangi platformun daha fazla kod geliştirmeyi gerektirdiğini araştırmayı amaçladık.

Üçüncü parti kütüphaneleri, geliştirilen uygulamadan bağımsız, farklı geliştiriciler tarafından ortaya konmuş, yeniden kullanılabilir yazılım bileşenleridir. Yazılım uygulamaları genelde üçüncü parti kütüphanelerden yararlanır, ihtiyaca göre bunları özelleştirir ve bakımını yapar. Bu nedenle üçüncü parti kütüphane kaynak kodunu ve projeye özgü kaynak kodunu ayrı ayrı analiz etmek önemlidir [4].

Öncelikle, bu ölçütlerin değerlerini kaynak kodun tamamı için elde ettik. Sonra ölçümlerimizi kaynak kodlarını projeye özel kaynak kodu ve üçüncü parti kaynak kodu olmak üzere iki farklı alt kümeye ayırıp tekrarladık. Bu iki alt küme için yapılan ölçümler, bir platformun diğerine kıyasla ne kadar projeye özel kaynak kodu geliştirme

gerektirdiğini ve uygulamaların mobil ve masaüstü platformlarının üçüncü parti kaynak koduna ne kadar dayandığını göstermektedir.

Projelerin kaynak kodlarından üçüncü parti kaynak kodlarını ayırt ederken, proje dizininde üçüncü parti kod içermesi muhtemel alt dizinleri aradık (örn. Android için src/com gibi). Ardından, kaynak kod dosyalarının başındaki yorumlarda, lisans sözleşmeleri ve hak taleplerini belirten her bir kaynak dosyasını tarayarak bunları projeye özel kaynak kodu dosyası veya üçüncü taraf kaynak kodu dosyası olarak tespit ettik.

Tasarım özellikleri. C&K ölçüt kümesinden aşağıdaki üçünü seçerek uygulamaların tasarım özelliklerini ölçtük;

- Sınıf Başına Ağırlıklı Metot Sayısı (WMC),
- Nesne Sınıfları Arasındaki Bağımlılık (CBO),
- Metot İçi Uyumsuzluk (LCOM).

Jabangwe ve arkadaşlarının yaptığı sistematik literatür araştırmasına göre C&K ölçüt kümesi, nesneye yönelik yazılımlar için en çok kullanılan ölçüt kümesi olarak karşımıza çıkar (99 çalışmanın %79'u) [12]. Aynı zamanda MOOD ve QMOOD ölçüt kümelerinden daha iyi performans gösterdikleri raporlanmaktadır. Yazılım karmaşıklık, uyum, boyut ve bağımlılık ölçütlerinin kalıtım ölçütlerine kıyasla yazılımın güvenilirlik ve bakım yapılabilirlik özelliklerini değerlendirmede daha etkili olduğu da bulunmuştur. Bu bulgular doğrultusunda çalışmamızda, C&K ölçüt kümesinin WMC, CBO ve LCOM ölçütleri seçmeye karar verdik. Bu ölçütleri, her uygulama çifti için hesapladık. Daha sonra, ölçümlerimizi uygulama çiftleri için karşılaştırmalı olarak analiz ettik.

4 Sonuçlar

Bu bölüm, vaka çalışmamızın sonuçlarını sunmaktadır. Her bir araştırma sorusu, ölçüm sonuçlarının analizi ile cevaplanmıştır.

4.1 AS-1: Aynı özelliklerde bir uygulama, farklı platformlar için geliştirilirken kaynak kodunun büyüklüğü nasıl değişir?

Bölüm 3'te sunulan kaynak kodu ölçümlerini önce toplam uygulama kodu için sonrasında ise toplam kaynak kodunu, projeye özel kaynak kodu ve üçüncü parti kaynak kodu olarak iki kümeye ayırıp bu kümeler için ölçümlerimizi tekrarladık.

Tablo 2, bu ölçümlerin sonuçlarını ve mobil uygulama ile masaüstü uygulaması arasındaki farkın yüzdesini göstermektedir.

Sonuçlar, Android platformu için geliştirilen bir uygulamanın %50 - %100 daha fazla kod satırı içerdiğini göstermektedir. Sınıf sayısındaki fark, masaüstü sürüm sınıflarının iki katıdır. Masaüstü sürümlerinde Android platformundan %63 - %86 arasında daha fazla dosya bulunmaktadır. Geliştirici tecrübesi ve geliştirme araçlarının (IDE) etkileri göz ardı edildiğinde, kaynak kodu boyutu geliştirme çabasının bir göstergesi olarak kabul edilmektedir [16]. Bu açıdan bakıldığında sonuçlar, Android platformunun aynı özelliklerin uygulanması için daha fazla çaba gerektirdiğini göstermektedir.

Tablo 2. Toplam Kaynak-Kodu Boyut Ölçütleri ve Platforma Göre Farklılıkları

Telegram			
Ölçüt	Masaüstü sürümü	Android sürümü	Fark oranı (%)
# Dosya	515	839	+%63
# Sınıf	1435	4472	+%211
# Kod satırı	220.986	450.952	+%104
KeePass			
-Ölçüt	Masaüstü sürümü	Android sürümü	Fark oranı (%)
# Dosya	194	362	+%86
# Sınıf	109	350	+%221
# Kod satırı	18.438	27.542	+%49

Kaynak kodu dosyalarını projeye özel veya üçüncü taraf (İng. third-party) bir diğer deyişle proje-dışı kaynak kodu dosyası olarak ayırt ettikten sonra, bu gruplar için yaptığımız kaynak kodu ölçütleri ölçümlerini Tablo 3 ve Tablo 4'te paylaştık.

Tablo 3, üçüncü parti kaynak-kodu ölçümleri için sonuçlarımızı göstermektedir. KeePass uygulamasının masaüstü ve Android sürümleri, kaynak kodlarında test kodu ve test kodu ile ilgili üçüncü parti kütüphaneler içerir. Gerçek uygulama kodları olmadığından bu kodlar ölçümlerden hariç tutulmuştur.

KeePass Masaüstü ve Telgram masaüstü uygulamaları QT üçüncü parti kütüphanesini kullanmıştır ancak QT kaynak kodu dosyaları, kod tabanına dahil edilmemiştir. Bu nedenle üçüncü parti kodları için bir karşılaştırma yapılamamıştır.

Tablo 4, tüm Android mobil uygulamalarının sayı olarak daha fazla sınıf ve kaynak kodu dosyası içerdiğini göstermektedir. Telegram Android için projeye özgü kaynak kodu farklılıklarının boyutu Masaüstü sürümünden %25 daha fazlaysa da KeePass uygulama çifti için tersi geçerlidir.

Tablo 3. Üçüncü Parti Kodu için Kaynak Kodu Boyut Ölçütlerinin Dağılımı

TELEGRAM			
Ölçüt	Masaüstü sürümü	Android sürümü	Fark oranı (%)
# Dosya	7	1015	+%14,400
# Sınıf	0	202	-
# Kod satırı	1,826	215,966	+%11,720
KeePass			
Ölçüt	Masaüstü sürümü	Android sürümü	Fark oranı (%)
# Dosya	-	59	-
# Sınıf	-	35	-
# Kod satırı	-	6,121	-

Tablo 4. Proje Spesifik Kod için Kaynak Kodu Boyutu Ölçütlerinin Dağılımı

Telegram			
Ölçüt	Masaüstü sürümü	Android sürümü	Fark oranı (%)
# Dosya	505	836	+%65
# Sınıf	1,435	4,270	+%197
# Kod satırı	185,195	234,830	+%26
KeePass			
Ölçüt	Masaüstü sürümü	Android sürümü	Fark oranı (%)
# Dosya	194	204	+%5
# Sınıf	109	291	+%167
# Kod satırı	18,438	13,953	-%24

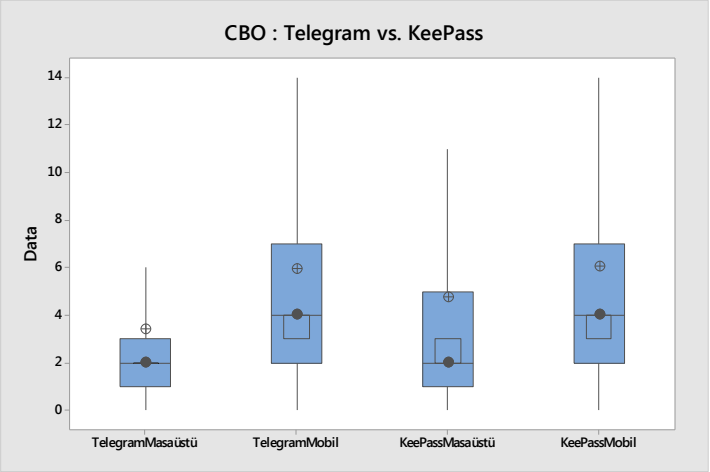
4.2 AS 2: Mobil ve masaüstü uygulamalarının tasarımları, tasarım nesneye yönelik ölçütleri açısından birbirinden nasıl farklılık gösterir?

C&K ölçüt kümesinden sınıf başına ağırlıklı metot sayısı (WMC), nesne sınıfları arası bağımlılık (CBO) ve metot içi uyumsuzluk (LCOM) ölçütleri tasarım özelliklerini karşılaştırırken kullanılmak üzere seçilmiştir. Bu ölçütler için uygulamaların kod tabanlarında yapılan ölçümlerin ortalaması Tablo 5'te görülmektedir.

Tablo 5. Masaüstü ve Mobil Sürümler için Ölçütlerin Ortalama Değerleri

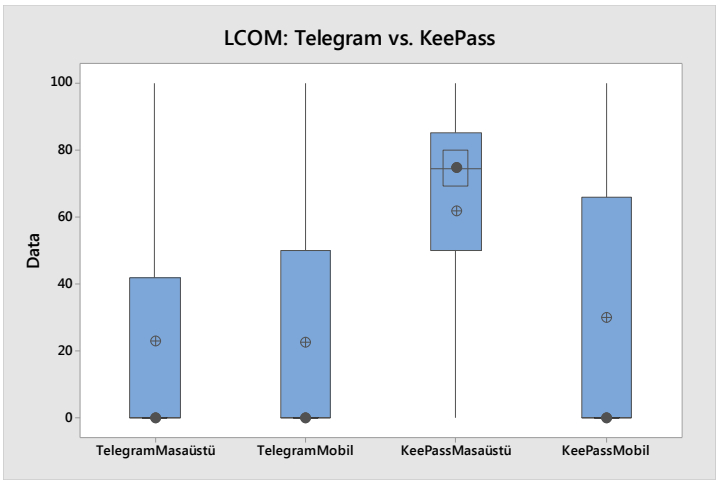
Telegram		
Ölçüt	Masaüstü sürümü	Android sürümü
WMC	7,96	11,28
CBO	3,43	5,92
LCOM	22,57	22,51
KeePass		
Ölçüt	Masaüstü sürümü	Android sürümü
WMC	23,77	9,13
CBO	4,70	6,01
LCOM	61,14	29,00

Aynı zamanda her iki uygulamanın mobil ve masaüstü sürümleri için her bir ölçüt bazında değerler karşılaştırmalı olarak incelenmiştir. CBO, sınıfın bağımlı olduğu sınıfların sayısı olarak tanımlanır. Nesne sınıfları arasındaki aşırı bağlanma, modüler tasarım için zararlıdır ve yeniden kullanılabilirliği olumsuz etkiler [17]. Bir sınıfın bağımsızlığı ne kadar fazlaysa, başka bir uygulamada onu tekrar kullanmak daha kolaydır. Dolayısıyla, düşük CBO arzu edilir. Telegram ve KeePass'in masaüstü sürümleri için CBO değerlerinin mobil sürümlerdeki CBO değerlerinden düşük olduğu gözlemlenmiştir (Bkz. **Şekil *1***).



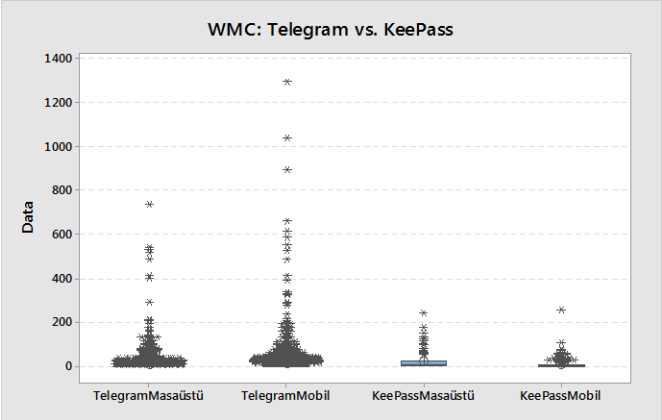
Şekil 1. CBO ölçüt değerlerinin kutu grafiği (boxplot)

Uyum eksikliği veya düşük bağlılık karmaşıklığı artırır, böylece geliştirme süreci sırasında hataların olasılığını artırır. Telegramın masaüstü ve mobil sürümünün ortalama LCOM değerleri birbirinden önemli ölçüde farklılık göstermemektedir. Öte yandan, KeePass için uygulama geneline baktığımızda uyum eksikliğinin masaüstü platform için daha fazla olduğu görülmektedir (Bkz. **Şekil *2***).



Şekil 2. LCOM ölçüt değerlerinin kutu grafiği (boxplot)

WMC, bireysel bir sınıfın karmaşıklığını ölçer. Çok sayıda yöntem içeren sınıfların, daha fazla uygulamaya özgü olmasının, yeniden kullanma olasılığını sınırladığı söylenir [9]. Telegram ve KeePass uygulama çiftleri için WMC değerleri dağılımı belirsiz çıkmışken (Bkz. **Şekil *3***), ortalama değerlere baktığımızda Telegram'ın mobil sürümünde WMC ortalaması yüksek iken, KeePass'in masaüstü sürümünde WMC ortalaması yüksektir.



Şekil 3. WMC ölçüt değerlerinin kutu grafiği (boxplot)

4.3 Geçerliliğe yönelik tehditler

Bu bölümde, standart bir kontrol listesi temel alınarak [18], hazırlanmış çalışmamıza sınır teşkil edebilecek geçerlilik tehditlerinden ve bunları nasıl azalttığımızdan, bahsedilmektedir. Dört tip geçerliliğe tehdit dikkate alınmıştır.

İçsel geçerlilik. Bu geçerlilik, bir çalışmanın elde edilen verilere dayalı nedensel bir sonuç garanti etmesi ile ilgilidir. AS-1 için, her bir uygulamadaki üçüncü parti kütüphanelerine ait kaynak kodlarının belirlenmesi elle yapılmıştır. Bazı üçüncü parti kütüphanelerinin yanlış tanımlanması mümkündür. Ayrıca, uygulamalarda kullanılmış fakat kaynak kodu paylaşılmamış eksik üçüncü parti kütüphaneleri olabilir.

Bunlar sonuçlarımızı etkileyebilecek faktörlerdir. Uygulamaların kaynak kodlarını değerlendirmek için nesne tabanlı diller için önerilmiş olan bir metrik kümesini tercih ettik. Bu sayede mobil ve masaüstü uygulamaların farklı programlama diliyle geliştirilmiş olmasından kaynaklanabilecek tehdidi azaltmayı hedefledik. Ölçütler için gerekli değerleri statik kod analiz aracı olan Understand [12] aracılığı ile elde ettik. Değerlendirdiğimiz uygulama çiftlerinin (Telegram ve KeePass) farklı işlevsel özelliklere sahip olması, bir iç tehdit oluşturmaktadır. Ancak uygulamaları birbiri arasında karşılaştırmadığımız için bu tehdit göz ardı edilebilir.

Yapısal geçerlilik. Bu geçerlilik, çalışmanın nesnelerinin gerçekten çalışmanın arkasındaki teoriyi temsil etme ölçüsü ile ilgilidir. Çalışmamızda kullandığımız yaklaşım mobil uygulamalar üzerine olan ve kod tabanlı incelemeler yapan literatürdeki çalışmalara benzemektedir [4][5]. Çalışmalarında nesne odaklı metrikleri kullananların, çoğunlukla tercih etmesinden [12] yola çıkarak çalışmamızda C&K ölçüt kümesini seçtik. Her araştırma sorusunu adreslemek adına belirlenen ölçütlerin değerleri elde edilmiş ve istatistiksel analizler yapılmıştır.

Sonuç geçerliliği. Bir çalışmanın sonuç geçerliliği, onun sonuçlarının titiz ve tekrarlanabilir bir yöntem ile ulaşılabilir olup olmadığı ile ilgilidir. Bu çalışmada uygulamaların kaynak kodu büyüklüğü ve tasarım özellikleri Bölüm 3'te bahsettiğimiz yöntem ile niceliksel olarak ölçülmüştür. Sonuçlarımız uygulamaların kod depolarına eriştiğimiz tarihteki sürümleri için geçerlidir. Öte yandan, kaynak kodu ve yazılım

tasarımı üzerinde etkili olabilecek faktörler olan geliştiricilerin deneyimi ve sayısı, uygulamanın varsa sürüm sayısı ve geliştirme tarihi hakkında detaylı bilgiye sahip olamadığımızdan, vaka incelememizin sonuç geçerliliği için bir tehdittir.

Dış geçerlik. Dış geçerlilik bir çalışmanın sonuçlarının genelleştirebilir olma ölçüsü ile ilgilidir. Çalışılan uygulama çiftleri, mobil ve masaüstü platformları için eşdeğer özellikteki açık kaynak kodlu uygulamaların küçük bir alt kümesini temsil eder. Uygulama boyutları, 8.000-450.000 LOC arasında değişmektedir. Bu açıdan sonuçlarımız, diğer uygulamalar veya platformlar için genellenemez.

5 Sonuç ve gelecek çalışmalar

Bu çalışmada Telegram ve KeePass açık kaynak uygulamalarının mobil ve masaüstü sürümleri analiz edilmiş ve karşılaştırılmıştır. Motivasyonumuz mobil ve masaüstü sürümlerinin kaynak kod ve tasarımlarının üzerindeki platform etkisini ortaya koymaktır.

Uygulamaları kaynak kod boyutu açısından karşılaştırmak için sınıf sayısı, kod satır sayısı (LOC) ve dosya sayısı ölçütlerini kullandık. Ölçümlerimizi, öncelikle uygulama geneli için, sonrasında ise uygulamaları projeye özel ve üçüncü parti kaynak kodu olmak üzere iki gruba ayırıp her bir grup için ayrı ayrı yaptık. Ölçümlerimiz sonucunda her iki uygulamada da toplam kaynak kodu boyutunun (sınıf sayısı, dosya sayısı, kod satır sayısı) mobil sürümlerde masaüstü sürümlerden daha büyük olduğunu gözlemledik. Bu nedenle aynı özellikte bir uygulamayı mobil (Android) platform için geliştirmenin daha çok efor gerektirmesinin olası olduğu sonucuna varabiliriz.

Uygulamaların tasarım özelliklerini karşılaştırmak için C&K ölçüt kümesinden CBO, LCOM ve WMC ölçütlerini seçtik [12]. LCOM ölçütü için elde ettiğimiz değerlerde karşılaştırdığımız iki uygulama çifti için sonuçlar arasında bir tutarlılık gözlemlenmemiştir. CBO, yani sınıflar arası bağımlılığın her iki uygulamanın (Telegram ve KeePass) mobil sürümlerinde daha fazla olduğunu gözlemledik. WMC ölçütü değerlerine baktığımızda ise her iki uygulamanın da masaüstü sürümünde WMC değerlerinin daha yüksek olduğunu gözlemledik.

Bu çalışmanın devamında, platformlar arası bağımlılık için Syer ve arkadaşlarının önermiş olduğu bir yöntemi kullanarak [5] mobil ve masaüstü sürümleri karşılaştırmayı hedefliyoruz. Buna ek olarak, bu çalışmanın devamında uygulamaların üst düzey mimarileri incelenebilir ve platform etkisi hakkında daha fazla bilgiye sahip olunabilir. Ayrıca, yazılım hataları ile incelenen kaynak kodu, bağımlılık ve tasarım ölçütleri arasındaki ilişki araştırılabilir. Son olarak, bu çalışmada incelenen açık kaynak kodlu iki uygulama çiftine ek olarak, endüstri içindeki firmalarla iş birliği ile hususi uygulamaların da mobil ve masaüstü sürümleri incelenerek elde edilecek sonuçların daha genelleştirilebilir olması araştırılabilir.

Kaynaklar

1. Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Evaluating Cross-Platform Development Approaches for Mobile Applications. In: Cordeiro, J. and Krempels, K.-

H. (eds.) Web Information Systems and Technologies: 8th International Conference, WEBIST 2012, Porto, Portugal, (2012).
2. Schmidt, D.C.: Model-Driven Engineering. IEEE Comput. 39, 25–31 (2006).
3. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented Design. IEEE Trans. Softw. Eng. 20, 476–493 (1994).
4. Syer, M.D., Adams, B., Zou, Y., Hassan, A.E.: Exploring the development of micro-apps: A case study on the blackberry and android platforms. In: Proceedings - 11th IEEE International Working Conference on Source Code Analysis and Manipulation. pp. 55–64 (2011).
5. Syer, M.D., Nagappan, M., Adams, B., Hassan, A.E.: Studying the relationship between source code quality and mobile platform dependence. Softw. Qual. J. 23, 485–508 (2015).
6. Bosch, J., Molin, P.: Software architecture design: evaluation and transformation. In: Proceedings ECBS'99. IEEE Conference and Workshop on Engineering of Computer-Based Systems. pp. 4–10 (1999).
7. Harrison, R., Counsell, S.J., Nithi, R. V.: An evaluation of the MOOD set of object-oriented software metrics. IEEE Trans. Softw. Eng. 24, 491–496 (1998).
8. Hitz, M., Montazeri, B.: Measuring coupling and cohesion in object-oriented systems. Proc. Int. Symp. Appl. Corp. Comput. Oct. 25-27, 1995. 75–76, 197, 78–84 (1995).
9. Lake, A., Cook, C.: Use of factor analysis to develop OOP software complexity metrics. … . 6th Annu. Oregon Work. Softw. Metrics, …. 1–15 (1994).
10. Briand, L.C., Wüst, J., Daly, J.W., Victor Porter, D.: Exploring the relationships between design measures and software quality in object-oriented systems. J. Syst. Softw. 51, 245–273 (2000).
11. Briand, L.C., Melo, W.L., Wüst, J.: Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Trans. Softw. Eng. 28, 706–720 (2002).
12. Jabangwe, R., Börstler, J., Šmite, D., Wohlin, C.: Empirical evidence on the link between object-oriented measures and external quality attributes: A systematic literature review. Empir. Softw. Eng. 20, 640–693 (2015).
13. Abreu, F.B., Carapuça, R.: Object-Oriented Software Engineering : Measuring and Controlling the Development Process. 4th. Int. Conf. Softw. Qual. 4, 3–5 (1994).
14. Bansiya, J., Davis, C.G.: A hierarchical model for object-oriented design quality assessment. IEEE Trans. Softw. Eng. 28, 4–17 (2002).
15. Fenton, N.E.: Software Metrics: Successes, Failures and New Directions. J. Syst. Softw. 47, 149–157 (1999).
16. Mendes, E., Mosley, N., Counsell, S.: Web metrics — estimating design and authoring effort. IEEE Multimed. 8, 50–57 (2001).
17. Cankurtaran, E., Çilden, E., Tarhan, A.: Bileşen Tabanlı ve Ürün Hattı Yazılım Geliştirme Yaklaşımlarında Yeniden Kullanılabilirlik Metrikleri Yeniden Kullanım ve Yeniden Kullanılabilirlik. Presented at the .
18. Feldt, R., Magazinius, A.: Validity Threats in Empirical Software Engineering Research - An Initial Survey. Proc. Int'l Conf. Softw. Eng. Knowl. Eng. 374–379 (2010).