T.C.
DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ANDROİD MOBİL UYGULAMA TESTİ

YÜKSEK LİSANS TEZİ

BÜŞRA TAKGİL

AĞUSTOS 2015

DÜZCE

KABUL VE ONAY BELGESİ

Büşra TAKGİL tarafından hazırlanan ANDROİD MOBİL UYGULAMA TESTİ isimli lisansüstü tez çalışması, Düzce Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararı ile oluşturulan jüri tarafından ..... Anabilim Dalı'nda Yüksek Lisans / Doktora Tezi olarak kabul edilmiştir.

Üye
(Tez Danışmanı)
Doç. Dr. Resul KARA
Düzce Üniversitesi

Üye
Yrd.Doç.Dr.Mehmet ŞİMŞEK
Düzce Üniversitesi

Üye
Yrd.Doç Dr. İbrahim Alper DOĞRU
Gazi Üniversitesi

Tezin Savunulduğu Tarih : .....

ONAY

Bu tez ile Düzce Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu Büşra TAKGİL' in Bilgisayar Mühendisliği Anabilim Dalı'nda Yüksek Lisans / Doktora derecesini almasını onamıştır.

Prof. Dr. Haldun MÜDERRİSOĞLU
Fen Bilimleri Enstitüsü Müdürü

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını beyan ederim.

12 Ağustos 2015

(İmza)

Büşra TAKGİL

Sevgili Aileme…

TEŞEKKÜR

Yüksek lisans öğrenimim ve bu tezin hazırlanması süresince gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Doç. Dr. Resul KARA' ya en içten dileklerimle teşekkür ederim.

Bu çalışma boyunca yardımlarını ve desteklerini esirgemeyen sevgili aileme ve çalışma arkadaşlarıma sonsuz teşekkürlerimi sunarım. Ayrıca Hüseyin BODUR' a teşekkürü borç bilirim.

12 Ağustos 2015 **Büşra TAKGİL**

İÇİNDEKİLER **Sayfa**

ŞEKİL LİSTESİ

SİMGELER VE KISALTMALAR

ADT	Android Development Toolkit (Android Geliştirme Aracı)
AVD	Android Virtual Devices (Android Sanal Cihaz)
SDK	Android Software Development Kit (Android Yazılım Geliştirme Aracı)
JVM	Java Virtual Machine (Java Sanal Makinesi)
UI	User Interface (Kullanıcı Arayüzü)

ÖZET

ANDROİD MOBİL UYGULAMA TESTİ

Büşra TAKGİL
Düzce Üniversitesi
Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi
Danışman: Doç. Dr. Resul KARA
Ağustos 2015, 71 sayfa

Mobil ortamların kullanıcılarının hızla artması, mobil uygulamaların popülaritesini de artırmaktadır. Bu durum mobil uygulamaların kalitesini daha da önemli hale getirmektedir. Test ise bu kaliteyi sağlamanın önemli bir ölçütüdür.
Mobil uygulamaların testi geleneksel uygulama testleriyle benzer özelliklere sahip olsa da mobil uygulamalar için bazı ek gereksinimlere ihtiyaç duyulur. Mobil uygulamalar test edilirken bazı zorluklarla karşılaşılır. Diğer uygulamalarla etkileşim, cihazlar üzerindeki ekran, kamera ve diğer donanımlardaki sensörler, donanım ve yazılım platform aileleri, kullanıcı ara yüzleri, enerji tüketimi, iletişim esnasındaki karmaşıklık bunlardan birkaçıdır. Mobil platformun kullanıcılara uygulamaları kolayca indirip yükleme ve çalıştırmasına izin veren yapısından dolayı, cihazlar üzerindeki veriler aynı ortamda çalışan uygulamalar için hedef haline gelmektedir. Donanım platformunun kaynak kısıtlılığı da mobil uygulamaların gelişiminde bir zorluk olarak görülür. Android testleri de tüm bu zorluklara ek olarak kendine özgü zorluklar içerir. Test esnasında Android yapısına ait özel problemler, açık konular ve çeşitli sorunlar ortaya çıkar. Bunların önemli bir sebebi geliştiricilerin acemiliğidir. Android uygulamalarının testi ve gelişimi taşınabilir cihazlar üzerindeki kısıtlamalardan etkilenir. Bu faktörlerin tümü test süreci ve kalite güvencesi için yeni zorluklar ortaya çıkarır.
Bu çalışmada Android platformunda mobil uygulamaların testi için başlıca zorluklara dikkat çekilmiştir ve mobil test zorlukları için otomasyon çözümü önerilmiştir. Ayrıca tez için geliştirilen bir Android mobil uygulaması Robotium test aracı kullanılarak test edilmiştir.

Anahtar sözcükler: Android, Mobil Uygulama Testi, Yazılım Testi

ABSTRACT

ANDROID MOBILE APPLICATION TESTING

Büşra TAKGİL
Duzce University
Graduate School of Natural and Applied Sciences, Departmant of Computer Engineering
Master of Science Thesis
Supervisor: Assoc. Prof. Dr. Resul KARA
August 2015, 71 pages

The dramatic increase of the mobile platform users led an increase of the popularity of mobile applications. Since, quality of mobile application becomes more important. Testing is an important parameter to ensure the desired quality. While testing the mobile applications, some difficulties occur. Interaction with other applications, screens on devices, sensors on the cameras and other hardware, user interface, energy consumption, complication of the communication are some of the difficulties mentioned above. Since the mobile platform configuration lets users to download and setup applications easily, the data on devices becomes targets for the applications that are running on the same platform. Another problem for the development of mobile applications is lack of sources. Android tests also have additional distinctive problems. While testing, some specific android configuration problems, open-ended topics and different problems occur. An important source of the problems is inexperienced developers. Testing and development of Android applications are affected by the restrictions on devices. All these factors cause difficulties on testing process and quality assurance. In this study, major difficulties of testing mobile applications on Android platform are discussed. An automation solution is proposed for the challenges of mobile tests. In addition, an Android application developed for the thesis has been tested using Robotium test tool.

Keywords: Android, Mobile Application Testing, Software Testing

EXTENDED ABSTRACT

ANDROID MOBILE APPLICATION TESTING

Büşra TAKGİL
Duzce University
Graduate School of Natural and Applied Sciences, Departmant of Computer Engineering
Master of Science Thesis
Supervisor: Assoc. Prof. Dr. Resul KARA
June 2015, 71 pages

1. INTRODUCTION:

Software test is control of the software which shows respecting acts or not, and basic aim is determining the expected quality of the products. In order to constitute quality software, effort and attention is needed on software development process. Testing is one of the important criterion to provide this quality.

Depend on developing mobile application, qualitiy of application is become increasingly important. Because of the fact that mobile devices becoming functional and having more complex features, testing of the mobile application is brought some difficulties along. Testing of Andorid mobile applications is effected by these difficulties and includes unique rigors.

This study states difficulties about Android mobile application test. In oreder to overcome these difficulties automation solution method is recommendend. Also, developed Android application is tested by using Robotium test automation tool.

2. MATHERIAL AND METHOD:

In this thesis study automated testing tools for mobile application testing challenges are seen as the solution and Android application developed for thesis is tested by using Robotium automation test tool. Test results are evaluated and proposals have been made to fix bugs.

3. RESULTS AND DISCUSSION:

In this thesis, mobile applicaiton test difficulties have mentioned and specifically concerned with the testing of Android mobile applications. Test automation tools to cope with the challenges and complexity of test method emerges as an effective solution. Test automation controls the execution of test and helps to compare actual results with expected results using the software. It increases productivity by reducing the time spent. Error rate can be reduced to using automation tools. Thus, the test challenges of mobile applications has been provided with automation solutions perspective.

4. CONCLUSION AND OUTLOOK:

Testing is an important quality criterion for applications. More successful products may be obtained by testing the Mobile applications. However, some difficulties arise when mobile applications tested. The test of the Android mobile application can be affected by the challenges and also includes Android 's unique difficulties. In this study we suggest to use of automated testing tools to overcome the difficulties and we tested Android application developed for thesis using Robotium test tools. Software has revealed that errors by testing the application, through automation tool tests could be implemented more quickly and more easily. New solutions for mobile testing challenges can be worked for the future development, new testing principles for the testing of Android applications can be developed.

1. GİRİŞ

1.1. AMAÇ VE KAPSAM

Yazılım testleri yazılım geliştirme sürecinde test edilen yazılımın kalitesini, güvenilirliğini, doğruluğunu ve tamlığını kontrol etmek için yapılır. Test edilen yazılımların karmaşıklığı ve kritikliği giderek arttığı için yazılımların test edilmesi daha fazla önem kazanmaktadır ve bu durum yazılımların daha dikkatli ve ayrıntılı test edilmesini gerektirir.

Yazılım test mühendisliği alanında birçok çalışma yapılmasına rağmen ülkemizde bu alandaki çalışmalar yeni yeni yaygınlaşmaya başlamıştır.

Mobil uygulamaların testi geleneksel uygulama testleriyle benzer özelliklere sahip olsa da mobil uygulamalar test edilirken bazı ek gereksinimlere ihtiyaç duyulur. Mobil uygulamalar test edilirken karşılaşılan zorluklara şunlar dâhil edilebilir: Diğer uygulamalarla etkileşim, Sensörler: dokunmatik ekran, mikrofon, kamera, Donanım ve yazılım platform aileleri, Kullanıcı ara yüzleri, Cihaz enerji tüketimi/batarya, İletişim esnasındaki karmaşık testler. Ayrıca mobil platformun kullanıcılara uygulamaları kolayca indirip yükleme ve çalıştırmasına izin veren yapısından dolayı, cihazlar üzerindeki hassas veriler aynı ortamda çalışan uygulamalar için kolayca hedef haline gelmektedir. Bunlara ek olarak mobil cihazlardaki kısıtlamalardan dolayı cihazlar üzerinde direk olarak test yapmak pratik değildir. Bu sebeple iki aşamalı olarak uygulama testi gerçekleşir. Donanım platformunun kaynak kısıtlılığı da modern mobil uygulamaların gelişiminde bir zorluk olarak görülür. Android testleri de tüm bu zorluklara ek olarak kendine özgü zorluklar içerir. Android uygulamaları etkili test tekniklerine, stratejilerine ve araçlarına ihtiyaç duyar ve test esnasında Android yapısına ait özel problemler, açık konular ve çeşitli sorunlar ortaya çıkar. Örnek olarak geliştiricilerin çoğu Android geliştirme platformuna büyük ölçüde yabancıdır, geri kalanlar ise yeni hatalara karşı bilgisizdir ve sürekli gelişen ve yenilenen yapısıyla Android geliştirme platformu hala olgunlaşmamıştır. Bu hamlık da Android

uygulamalarının kalitesinden emin olmayı amaçlayan test aktivitelerinin gerekliliğini ortaya koyar. Tüm mobil uygulamalarda olduğu gibi Android uygulamalarının test stratejileri ve gelişimi taşınabilir cihazlar üzerindeki kısıtlamalardan etkilenir. Örneğin mobil cihazların yazılım ve donanım konfigürasyonları arasındaki heterojenlik kısıtlaması gibi. Bu faktörlerin tümü test süreci ve kalite güvencesi için yeni zorluklar ortaya çıkarır. Bu durum birçok araştırmacıyı, endüstriyel girişimciyi mobil uygulamalar için etkili test prensipleri, teknikleri ve araçları hakkında çalışmaya yöneltmiştir.

Bu tezin genel bilgiler kapsamında yazılım mühendisliği, yazılım mühendisliğinin alt dalı olan yazılım test mühendisi, yazılım test süreci, mobil uygulamalar için yazılım testi konuları incelenip, yazılım test düzey ve teknikleri hakkında bilgi verilmiştir.

Yazılım test araçları ve test otomasyonunun önemi vurgulanıp, test otomasyon araçları hakkında bilgi verilerek Robotium test aracı kullanılarak test edilmek için tez kapsamında geliştirilen bir uygulama test edilmiştir.

1.2.LİTERATÜR TARAMASI

Yazılım geliştirmenin önemli aşamalarından biri olan yazılım testinin mobil uygulamalar üzerinde uygulanması ile ilgili literatürde bazı çalışmalar yer almaktadır. Bu çalışmaların bir kısmı mobil testlerin zorluklarına dikkat çeken çalışmalar, bir kısmı test ortamı önerileri ile ilgili çalışmalar, önemli bir kısmı da test araçlarının uygulamalar üzerinde uygulanması ile ilgili çalışmalardır. Aşağıda bu çalışmalar detaylandırılmıştır.

Knych ve Baliga tarafından yapılan çalışmada mobil uygulamaların kalitesine ve mobil uygulamaları test etmenin zorluğuna dikkat çekilmiştir teste dair zorlukları azaltmak için test stratejisi geliştirilmiştir. Bu test stratejisinde sistem testine birim testleriyle başlanılması gerektiği savunulmuştur [1].

Dantas ve arkadaşları tarafından yapılan çalışmada mobil uygulamalar için test gereksinimleri belirlenmiş ve bu gereksinimleri uygulamanın test sürecinde verimliliği arttıracağı belirtilmiştir. Mobil uygulama geliştirme dinamiklerini anlamak için 2 anket geliştirilmiştir. Anketlerin ilkinde geliştiricilerin ikincisinde ise testi gerçekleştirecek olan kişilerin deneyimlerine yer verilerek karşılaştırma yapılmıştır. Elde edilen sonuçların mobil cihaz uygulamalarını test etmede yardımcı olacağı savunulmuştur [2].

Franke ve Weise, mobil uygulamaların testi için kalite çerçevesi sunmuşlardır. Bu çerçeve mobil yazılımların temel niteliklerini tanımlayan kalite modeline dayandırılmıştır ve mobil uygulama testi için ölçütler içermektedir [3].

Mobil yazılım kalitesi ile ilgili Franke ve arkadaşları tarafından yapılan başka bir çalışmada bir mobil yazılım kalite modeli önerilmiştir. Esneklik, taşınabilirlik gibi kalite modeli bileşenleri incelenmiş önerilen model iki ayrı Android uygulamasına uygulanmış ve mobil uygulamaya özgü nitelikler belirlenmeye çalışılmıştır [4].

Kirubakaranve ve Karthikeyani'nın yaptığı mobil uygulama test zorlukları ve otomasyon yoluyla çözüm yaklaşımlarının yer aldığı çalışmada mobil uygulamaların geleneksel uygulamalardan farkı incelenmiş, mobil uygulama testlerindeki yeni yaklaşımlar ve zorluklar tartışılmış ve mobil uygulamaya otomasyonun etkisine dikkat çekilmiştir. Mobil uygulamadaki zorlukların başlıca sebebinin mobil uygulamaların hareketli yapısından kaynaklandığı öne sürülmüştür. Ayrıca çevre şartlarının değişkenliğinin performansı ciddi bir şekilde etkilediği belirtilmiştir [5].

Kaasila ve arkadaşları otomatik Android UI testleri üzerine çalışmışlardır ve bu çalışmada fiziksel Android telefonların çeşitleriyle kullanıcı arayüz testlerinin yürütüldüğü Testroid çevrimiçi platformu tanıtılmıştır. Mobil uygulamaların testinde tek bir telefonun yeterli olmadığı ve bu sebeple kapsamlı bir test için birden fazla cihaz üzerinde uygulamayı test etmenin gerekliliği vurgulanmıştır. Ayrıca bu çalışmada otomatik test araçları değerlendirilmiş avantaj ve dezavantajlarına yer verilmiştir [6].

Satoh mobil cihazların test uygulamaları için çerçeve sunmuştur. Sunulan bu çerçeve mobil cihazın hareketiyle yeni bir ağa bağlanıldığında cihaz üzerinde çalışan uygulamaların işleyişiyle ilgili problemleri ortadan kaldırmak için mobil cihazlara uygulama seviyesinde bir emülatör sağlamaktadır. Bu makalede ağa bağımlı uygulamalar için oluşturulan çerçevenin yararları gösterilmiştir [7].

Liu ve arkadaşları ise otomatik birim testi için Android tabanlı yaklaşımlar sunmuşlardır. V model yazılım testinin ilk aşaması olan birim testleri için tam otomatik test yaklaşımı önerilmiştir. Bu yaklaşım android projeleri için görsel bir test raporu oluşturma imkanı sağlamaktadır. Bu sayede çoklu birim testlerinin verimliliği artmaktadır ve yazılım ürünleri kullanıcıya hızlı bir şekilde ulaşmaktadır. Otomatik test için JEnkins aracı kullanılmıştır [8].

Shin ve arkadaşları bir karşılaştırma algoritması olan Moment Invariants algoritmasını kullanarak Android platformu ve cihaz özelliklerinden kaynaklanan bozuk görüntüleri tespit ve analiz etmeyi amaçlayan deneysel bir model önermişlerdir. Deneyde ilk olarak normal ekran görüntüleriyle test ekran görüntüleri karşılaştırılmış ve sonuç olarak çözünürlüğün görüntüyü etkileyen en önemli faktör olduğu yargısına varılmıştır [9].

Geogy ve Dharani yaptıkları çalışmada Android için geliştirilen uygulamaların otomatik testinin zorluklarından bahsedilmiş ve Android uygulamaların regresyon testi için bir teknik sunulmuştur. Bu teknik GUI uygulamaların modellerini otomatik olarak oluşturan ve elde edilen test durumlarının otomatik çalıştırılmasına dayalıdır [10].

Mirzaei ve arkadaşları mobil uygulamayı test ederken test olaylarının sistematik olarak oluşturulma zorluğuna dikkat çekmiş, Java programlarında sistematik test olayı üretmek için yöntemlerden biri olan sembolik çalıştırmayı kullanarak Android uygulamaları test etmişlerdir. Ayrıca Android uygulamaların yürütülmesi sırasında JVM(Java Virtual Machine) ile ilgili uyumsuzlukları gidermek için Android uygulamalarının çalışabileceği Java Pathfinder (JPF)'da Android kütüphanelerinin bir modelini geliştirmişlerdir [11].

Guo ve arkadaşları Android iç bileşenlerinde var olan hassasiyetleri tespit etmek için yeni bir çalışma yapmışlardır. Bu çalışmada Android iç uygulama(inter-application) bileşenlerinin güvenlik mekanizması analiz edilmiş ve bunlara göre güvenlik kuralları oluşturulmuştur. Bileşenler arası mesajlaşmanın sebep olduğu güvenlik açıklarını tespit etmek için dinamik ve statik otomatik test tekniklerini içeren yeni bir yaklaşım önerilmiştir [12].

Delamaro ve arkadaşlarının yaptığı çalışmada yalnızca emulatör üzerinde test edilmeyip aynı zamanda gerçek hedef cihaz üzerinde test edilebilen mobil cihaz yazılımlarının kapsama testini destekleyen stratejiler sunulmuştur. Basit bir test olayının farklı dil desteği sağlayan bir platform olan Jabuti/ME test aracı ile nasıl gerçekleştirileceği anlatılmıştır [13].

Fetaji ve arkadaşları tarafından literatürdeki başarılı m-learning sistemlerini gözden geçirerek mobil öğrenme sistemlerinin verimliliği, etkililiği ve kullanılabilirliği hakkındaki araştırmalardaki eksiklere dikkat çekmişlerdir ve bu çalışmada başarılı

kullanılabilir bir m-learning'in nasıl uygulanacağı stratejisi önerilmiş ve öğrenilmiş çevrenin kullanılabilirliği tartışılmıştır [14].

Kovacevic ve arkadaşları Android tabanlı dijital tv alıcısı için otomatik test olaylarının çalıştırılması, test planının doğrulanması ve gereksinim tanımlamalarından ürün testinin tüm sürecini kapsayabilen otomatik bir test sistemi önermişlerdir. Önerilen sistem kara kutu test stratejisine dayanan otomatik test web araçlarından oluşmaktadır. Sistem için sunulan otomatik test tv alıcısının stres testi, performans testi ve fonksiyonel testinde de kullanılabilmektedir [15].

Shahriar ve arkadaşları Android uygulamalarındaki bellek sızıntı testlerini gerçekleştirmişlerdir. Android uygulamalara özgü bellek sızıntı modelleri geliştirilmiştir. Daha sonra bu modele dayalı olarak bellek sızıntısını taklit eden test olayları üretilmiştir. Elde edilen sonuçlar önerilen test yaklaşımının hafıza sızıntılarını etkili bir şekilde bulduğunu göstermiştir [16].

Bu tez çalışmasında mobil yazılımları test etmenin zorluklarına değinilmiş, bu zorluklara çözüm olarak otomasyon test araçlarının kullanılması önerilmiştir ve Robotium test aracı kullanılarak, tez için geliştirilen bir Android uygulaması test edilerek, yazılım hataları ortaya çıkarılmıştır. Test sonuçları değerlendirilmiştir.

1.3.YAZILIM MÜHENDİSLİĞİ

Yazılım mühendisliği için ilk tanımlama 1969 yılında gerçekleşen bir konferansta Fritz Bauer tarafından "yazılım mühendisliği gerçek makineler üzerinde etkin ve güvenilir çalışan ekonomik yazılımların geliştirilmesi için mühendislik ilkelerini kullanmak olarak ifade edilmiştir" [17].

Yazılım mühendisliği terimi NATO Bilim Komitesi kongresinde tartışılmaya başlanmış ve gün geçtikçe önem kazanan bir çalışma alanı olmuştur [18]. 1968'den bugüne kadar 40 yılı aşkın sürede teknolojik gelişmeler ile birlikte yazılım mühendisliği disiplini çok gelişme kaydetmiş ve ilerlemiş olmasına rağmen günümüzde hala kendini tanımlamaya ve diğer mühendislik dalları arasında yer edinmeye çalışmaktadır [19].

Yazılım mühendisliği, karmaşıklığı ve kritikliği artan yazılımların güvenilirliğini, doğruluğunu ve tamlığını arttırmayı hedeflemektedir. Yazılım mühendisliğinin amacı

başarılı yazılım projeleri üretmektir. Teknolojik gelişmelerle beraber yazılımlar karmaşıklaştıkça, proje boyutları büyüdükçe programlama ekibinde yer alan kişi sayısı arttıkça, planlama, iletişim, yönetim daha da önemli hale gelmektedir [20]. Bu durum yazılım mühendisliğine duyulan ihtiyacı ortaya çıkarmaktadır.

Yazılım mühendisliğinin kapsamını oluşturan öğeler ise şunlardır [21]:

- Yazılım gereksinim analizi
- Yazılım tasarımı
- Yazılım gerçekleştirme (Programlama)
- Yazılım testi
- Yazılım bakımı
- Yazılım mühendislik yönetimi
- Yazılım konfigürasyon yönetimi
- Yazılım mühendisliği süreçleri
- Yazılım mühendisliği araç ve yöntemleri
- Yazılım kalitesi

1.4.YAZILIM GELİŞTİRME SÜRECİ

Yazılım kelimesinin sözlük anlamı; "bir bilgisayarda donanıma hayat veren ve bilgi işlemde kullanılan programlar, yordamlar, programlama dilleri ve belgelemelerin tümü" olarak ifade edilmektedir [22]. Yazılım ayrıca, mevcut bir problemi çözmek amacıyla değişik cihazların birbirleriyle haberleşebilmesini sağlayan ve görevlerini ya da kullanılabilirliklerini geliştirmeye yarayan bilgisayar dili kullanılarak oluşturulmuş anlamlı ifadeler bütünü olarak da nitelendirilebilir [23].

Yazılım geliştirme, yazılımın hem üretim hem de kullanım süreci boyunca geçirdiği tüm aşamalar olarak tanımlanabilir.

Yazılım geliştirme süreci bir yazılım ürününün geliştirilmesi ile ilgili süreci kapsar ve bir dizi eylemden oluşur. Yazılım geliştirme süreci; tanımlama (definition), ayrıntılandırma (elaboration), gerçekleştirme (construction), değerlendirme (evaluation), yayma (transition) ve yönetim (management) aşamalarından oluşmaktadır [18].

Tanımlama: Yazılım geliştirmenin ilk aşamasıdır. Bu aşamada amaçlanan yazılımın işlevi ve hangi probleme çözüm getireceği belirlenir. Yazılımın hedeflenen işlevleri yerine getirirken nasıl bir ortamda çalışacağı ve ne gibi kısıtlamaları olacağı tanımlanır.

Ayrıntılandırma: Bu aşamada, problemin yazılımla nasıl çözüleceğine odaklanılır. Tanımlama aşamasında problem tüm yönleriyle ortaya konulmuştur. Yazılımın hedefi belirlidir ve bu hedefe nasıl gidileceğine dair çalışmalar yapılır, farklı düzeylerde (genel ve ayrıntılı) modeller geliştirilir. Bu modeller ile geliştirilecek olan yazılımın hangi alt sistemlerden oluşacağı, bu sistemler arasındaki etkileşimlerin (bilgi alışverişi) nasıl olacağı tanımlanır.

Gerçekleştirme: Ayrıntılandırma aşamasında geliştirilen modeller programlama dilleri kullanılarak kodlanır. Her bir kod parçasının testi gerçekleştirilir. Daha sonra çalışan kod parçaları bir araya getirilerek hedeflenen yazılım geliştirilir.

Değerlendirme: Geliştirilen yazılımın istenen amacı yerine getirip getirmediği değerlendirilir. Bu amaçla gözden geçirmeler ve testler kullanılarak geliştirilen yazılımın doğru çalıştığı ve kendinden beklenen işlevleri yerine getirdiği doğrulanır.

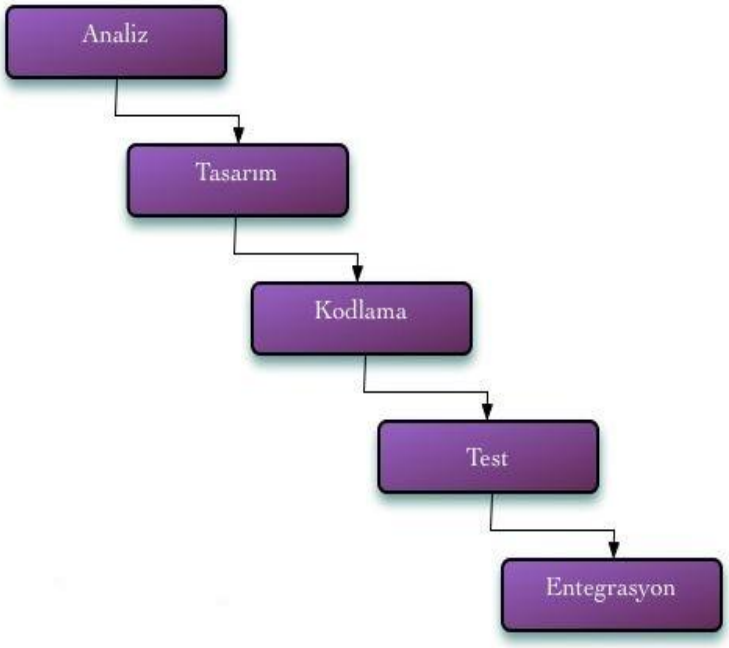
Yayma: Yazılımın piyasaya sürüldüğü ve kullanıcıların kullanımına sunulduğu aşamadır. Bu aşamadan sonra gelen geri bildirimler ile yazılım üzerinde düzeltici, uyarlayıcı, iyileştirici ve önleyici bakımlar gerçekleştirilebilir.

Yönetim: Yazılım projeleri kapsamında ilk evreden son evreye kadar gerçekleştirilen faaliyetlerin yönetimsel işlemleri bu aşamada gerçekleştirilir. Yazılım geliştirme süreci içerisinde yönetim aşaması ve değerlendirme aşaması diğer aşamalara bir arada yürütülür [18].

1.4.1.Yazılım Geliştirme Süreç Modelleri

1.4.1.1.Çağlayan Modeli (Waterfall Model)

Yazılım geliştirmede çok sayıda farklı model ve süreç değerlendirmelerinden söz etmek mümkündür. Bununla birlikte; yazılım mühendisliğindeki diğer modellere temel teşkil eden "Çağlayan Modeli (Waterfall Model)" yazılım yaşam döngüsünü analiz, tasarım, kodlama, test ve bakım olmak üzere beş aşamada ele almaktadır [26].



Şekil 1.1. Waterfall(Şelale) Modeli [25].

Analiz: Yazılımın işlevi ve bu işlevi nasıl yerine getireceğinin belirlendiği aşamadır. Yazılan kod işlevini doğru şekilde yerine getirebiliyorsa başarılı bir yazılım olarak adlandırılır. Bu sebeple yazılımdan ne istendiğinin doğru bir biçimde tanımlanması gereklidir. Analiz aşaması personel, donanım ve sistem gereksinimlerinin belirlenmesi, sistemin fizibilite çalışmasının yapılması, kullanıcıların gereksinimlerinin analizi, sistemin ne yapıp ne yapmayacağının kısıtlamalar göz önüne alınarak belirlenmesi, bu bilginin kullanıcılar tarafından doğrulanması ve proje planı oluşturulması adımlarından oluşur.

Tasarım: Analiz aşaması sonucunda belirlenen gereksinimlere yanıt verecek yazılımın tasarımının oluşturulduğu aşamadır. Yazılım tasarımı, bir bileşen veya sistemin nasıl gerçekleştirileceğini belirlemek için kullanılan teknikler, stratejiler, gösterimler ve desenleri içerir. Bu aşama yazılım bileşenleri arasındaki içsel ara yüzler, mimari tasarım, veri tasarımı, kullanıcı ara yüzü tasarımı, tasarım araçları ve tasarımın değerlendirilmesi alt süreçlerini de kapsamaktadır. Tasarım aşaması, yazılımın hem kullanıcı ara yüzünü hem de programın temel yapısını oluşturur. Yapılacak tasarım, yazılımın işlevsel gereksinimlere uygun olmasının yanı sıra kaynaklar, performans ve güvenlik gibi kavramları da göz önüne alınarak gerçekleştirilmelidir.

Kodlama: Kodlama aşaması, tasarım sürecinde ortaya konan veriler doğrultusunda yazılımın gerçekleştirilmesi aşamasıdır. Bu süreç programlamanın yanı sıra yazılımın geliştirilmesi ve kullanıcıya ulaştırılması sürecindeki bütün çalışmaları kapsar. Tasarım sonucu üretilen fiziksel modelin yazılıma dönüştüğü süreç olarak da nitelendirilebilir Yazılım geliştirme ortamı, programlama dili, veri tabanı yönetim sistemi, yazılım geliştirme araçları seçimi kodlama aşamasında gerçekleştirilir.

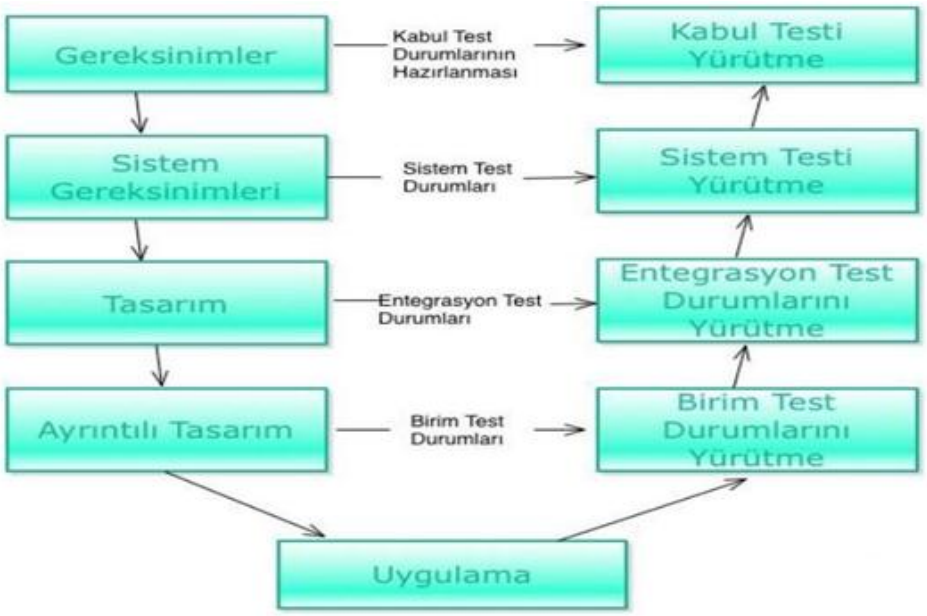
Test: Test aşaması, yazılım kodlanması sürecinin ardından gerçekleştirilen sınama ve doğrulama aşamasıdır. Elde edilen uygulama yazılımının hem belirlenen gereksinimleri sağlayıp sağlamadığı hem de gerçekleştirimin beklentilere uygun olup olmadığını kontrol etmek için statik ve dinamik sınama tekniklerinden yararlanır. Statik teknikler, yazılımın tüm yaşam döngüsü boyunca elde edilen gösterimlerin analizi ve kontrolüyle ilgilenirken, dinamik teknikler sadece gerçekleştirilmiş sistemi içerir. Yazılım üretiminde ilk testler genelde geliştirme sürecinde programcı tarafından yapılır. Bununla birlikte, asıl hata ayıklama ve geribildirim hizmeti test ekipleri tarafından yapılır. Testler ve geribildirim müşteri yazılımı kullandığı sürece devam eder.

Bakım: Hata giderme ve yeni eklentilerin yapıldığı aşamadır. Yazılımın kullanıma başlanmasından sonra yazılımın desteklenmesi sürecini kapsar. Yazılımın eksiklerinin giderilmesi, iyileştirilmesi gibi alt aşamaları içeren aşamadır.

1.4.1.2.V-Modeli

V-modeli şelale modelinin genişletilmiş bir yaklaşımıdır. Bu yaklaşımda daha güvenilir ve doğru yazılımlar geliştirmek için her bir yazılım geliştirme süreç adımının karşılığı

olan test eylemleri model içerisinde gösterilmiştir. Şelale modelden farklı olarak yazılım geliştirmeye başlamadan test planı oluşturulur. Şekil 1.2' de V-model gösterilmektedir.



Şekil 1.2. V Modeli [26].

1.4.1.3.Prototip Geliştirme Modeli [27]

Gereksinim belirsizliğini ortadan kaldırmak ve kullanıcının istediği yazılımı geliştirmek için ortaya konmuş bir modeldir. Gereksinim verileri toplanarak işe başlanır. Kullanıcılar ve geliştiriciler bir araya gelerek yazılım girdi ve çıktılarını belirlerler. Tespit edilen bilgilerle hızlıca bir tasarım prototipi oluşturularak kullanıcıya sunulur ve değerlendirme yapılır. Bu değerlendirmeler ışığında yeni gereksinimler tespit edilmeye çalışılır. Kullanıcın beklentisini karşılayacak şekilde yazılım oluşana kadar bu işleme devam edilir. Gereksinimler yeterince ayrıntılı hale getirildikten sonra istenilen sistem geliştirilir. Prototip geliştirme modeli gereksinimlerin net olarak bilinmediği durumlarda başarılı bir şekilde uygulanabilir.

1.4.1.4. Çevik Modeller [27]

Çevik modeller 1990'ların ortasında zor uygulanan, aşırı kuralcı klasik yazılım süreç modellerine tepki olarak ortaya çıkmıştır. Şelale ve V gibi klasik modeller doğalarında var olan aşırı belgelendirme faaliyetleri ve sıralı yaklaşımları ile daha yüksek maliyetle

daha yavaş yazılım geliştirmeye sebep olarak görülmüştür. Yazılım geliştirme sürecini hızlandırmak amacıyla bu süreci daha etkin kullanmak ve gerektiğinde belgelendirme yapmak temeline dayalı çevik yazılım geliştirme yöntemleri ortaya çıkmıştır. Çevik geliştirme modelleri şu ilkelere dayanır:

- Hızlı, devamlı ve kullanışlı yazılımlar üreterek müşteri memnuniyetini sağlamak amaçlanmalıdır.
- Çalışan yazılım gelişimin en önemli ölçüsüdür.
- Cevap verilemeyen veya geç cevap verilen talepler memnuniyeti azaltır.
- En iyi iletişim karşılıklı görüşmedir.
- Basitlik önemlidir.
- Kendi kendini organize eden takım yapısı gereklidir.

Yazılım geliştirme amacıyla üretilen bu yöntem klasik yazılım geliştirme modellerine göre yazılım geliştirmeyi daha esnek hale getirir. Bu yöntem müşterinin isteklerinin net olmadığı, yazılım geliştirmenin her aşamasında müşteri isteklerinin değişebildiği, hedeflenen sistemin hemen görülmek istendiği durumlarda kullanılabilir.

Başlıca çevik modeller:

- Uç (Sınırsal) Programlama (Extreme Programming)
- Çevik Birleştirilmiş Süreç (Agile Unified Process)
- Scrum
- Test Güdümlü Geliştirme (Test Driven Development)
- Çevik Bilgi Yöntemi (Agile Data Method)
- Özellik Güdümlü Geliştirme (Feature Driven Programming)

Çevik yazılım geliştirme kısa vadeli planlar ve küçük gelişmeler halinde yazılımın geliştirilmesini öngörür. Kısa vadeli planlar yazılımın geliştirilmesinde yineleme getirir.

Her yinelemede yazılım üzerine yeni özellikler eklenir. Çevik yazılım geliştirme değişikliklere uyum sağlamayı kolaylaştırır.

1.4.1.5.Evrimsel Geliştirme Modelleri [27]

Evrimsel geliştirme modellerinde, ilk gerçekleştirimi hızlı yapıp onun üzerinde kullanıcının tepkilerini alıp iyileştirmeyi öngörerek yazılım geliştirilmesi hedeflenir.

Başlıca evrimsel geliştirme modelleri:

. Araştırmacı (explatory) geliştirme

. Atılabilir (throwaway) prototipleme

Araştırmacı geliştirmede yazılım geliştirilmeye daha anlaşılır bir kesim ile başlanır; üstüne yeni özellikler eklenerek devam edilir. Bu yaklaşım yazılımın gereksinimlerinin daha iyi anlaşılmasını hedeflemektedir.

Atılabilir prototiplemede amaç yazılım gereksinimlerini keşfetmektir. Bu nedenle ilk anda tespit edilen gereksinimlerden bir prototip yazılım geliştirilerek müşteriye gösterilir. Buradan hareketle müşterinin kafasındaki hedef yazılım keşfedilmeye çalışılır. Gösterilen prototip daha sonra bir daha kullanılmadan atılabilir.

1.4.1.6. Spiral Model [27]

Spiral modelde yazılım geliştirme süreci geriye dönüşü olan ardışık faaliyetler yerine spiral olarak genişleyen bir yapıda ifade edilir. Spiral model kullanıldığında her bir sarmalın sonunda yazılımın yeni bir sürümü ortaya çıkar. İlk halkanın sonunda artırımda ortaya çıkarılan yazılım uygulama ortamında kullanılanın gerçek bir prototipidir. Geliştirimi sırasında tüm süreç uygulanmıştır. Sonraki yinelemelerde üstüne yeni işlevler yine sürecin tüm adımları kullanarak eklenir. Kalite, verimlilik ve kapasite açısından geliştirilen yazılımın uygun özellikleri taşıması her sarmal döngüde (her yeni sürümde) sağlanarak geliştirme süreci sürdürülür.

Sarmalın her bir döngüsü sırayla tekrarlanan dört etkinlikten oluşur:

- Planlama; yapılacakların eldeki kaynaklar ve zaman çizelgesi gibi konular göz önünde tutularak planlanması

- Risk çözümleme; teknik ve yönetsel risklerin çözümlenmesi
- Geliştirme; yazılımın gerçekleştirimi, çözümleme, tasarım, kodlama
- Değerlendirme; kullanıcı geribildiriminin alınması, yapılanın geçerliliğinin ve gerektiği gibi çalıştığının ortaya konması

Spiral model çok karmaşık büyük projelerde uygulanabilecek gerçekçi bir model olarak görülmektedir.

1.4.1.7.Artımlı Model [27]

Şelale modeline yinelemeli bir özellik katılarak artımlı model oluşturulmuştur. Bu model belirli bir takvime bağlı olarak yazılımı sürümler halinde geliştirip teslim etmeye dayanır. Her bir yeni sürüm, öncekinin üstüne bazı ek işlevlerin eklenmesini öngörür. Bu model gereksinimlerin tamamının belli olduğu durumlarda etkin bir şekilde kullanılabilir. Bu model evrimsel geliştirmeden farkı, çıkan her sürümün son ürünün sahip olduğu tüm işlevleri içermesidir.

2.MATERYAL VE YÖNTEMLER

2.1.YAZILIM TESTİ

Yazılımın özelliklerini değerlendirmek amacıyla incelenmesi ve yazılımda var olan ile istenen durumlar arasındaki farklılıkların değerlendirilme süreci yazılım testi olarak adlandırılır [28]. Test bir yazılımın hata bulma amacıyla çalıştırılmasıdır [29]. Başka bir tanımla test bir sistemin özellik ve yeteneklerinin uygunluğunu ölçmek için hatalarının açığa çıkarılma sürecidir [30].

Bir hata yazılım geliştirme sürecinin herhangi bir evresinde ortaya çıkabilir ve hatanın birçok sebebi olabilir. Boch, yazılım testini, "Korunmak için, akla gelmeyecek kadar gizli-kapalı belirsizlikleri karşılaştırma sürecidir." şeklinde tanımlamıştır. R. Vaderwall'a göre, "Yazılım testi, ürünün davranışlarını öngörme/tahmin etme ve bu tahminlerin gerçek sonuçlarıyla karşılaştırılma sürecidir."

Yazılım testleri hata olmadığını göstermek yerine yazılımda hataların var olduğunu göstermeyi amaçlar. Bu nedenle hataların tespit edilmesi yapılan testlerin başarısını gösterir. Ne kadar çok hata tespit edilip düzeltilmiş ise yazılım o kadar başarılıdır.

Yazılım testinin amaçları şunlardır:

- Yazılımda var olan hataları tespit etmek ve tekrarını önlemek
- Yazılım geliştirme süreci boyunca meydana gelen hataları ortaya çıkarmak
- Geliştirilen yazılımların kalitesini arttırmak
- Geliştirilen yazılımın beklentilere uygunluğunu ve doğruluğunu tespit etmek, sapmaları belirlemek
- Son ürün olarak hatadan arındırılmış ve isterleri karşılayan bir ürün teslim etmek
- Yazılım hatalarının tespit edilmesiyle ortaya çıkabilecek riskleri azaltmak.

Muller ve arkadaşları ise yazılım testinin amaçlarını şu şekilde özetlemektedir [31]:

- Bir kişiye, bir ortama veya bir şirkete zarar verebilecek yazılım hatalarını belirlemek.

- Hataların ana sebepleri ve etkileri arasındaki farkı ayırt edebilmek.

- Elde edilen örnekler yardımıyla testin niçin gerekli olduğunu tayin edebilmek.

- Test etmenin niçin kalite güvencenin bir parçası olduğunu ve elde edilen örneklere bakarak daha yüksek kalite sağlamak için nasıl bir testin olması gerektiğini saptamak.

-Hata(error), kusur(defect), aksaklık (fault), bozukluk(failure), yanlışlık(mistake) ve yanlış(bug) gibi terimleri yeniden hatırlamak.

- Yazılım projesinin başlangıcından bitmesine kadar olan süreçte hataları en aza indirebilmek.

- Projenin teslim edildikten sonraki teknik desteğini sağlayabilmek, yeni gereksinimleri projeye rahatlıkla uygulanabilmesini sağlamak ve kullanım kolaylığı gibi konularda yeterli olmak.

2.1.1.Yazılım Testinin Önemi

Günümüzde yazılımın yer almadığı alan neredeyse yoktur. Finans, Telekomünikasyon, Ar-ge, Savuma Sanayi, Bilgi Sistemleri gibi birçok alanda kullanılan yazılımların kontrolü test edilerek yapılmaktadır.

Yazılım dünyasındaki gelişmeler yazılımın karmaşıklığını ve büyüklüğünü arttırmıştır. Bu durum hata miktarının artması, güvenliğini sağlanamaması gibi bazı sorunları beraberinde getirmiştir. Bu sorunlar yazılımların ilk geliştirilmeye başladığı dönemlerden beri var olmuştur. Var olan yazılım projeleri incelendiğinde çok az bir miktarının başarı ile sonuçlandığı görülmüştür. Başarısız projeler nedeniyle de yılda milyonlarca dolar zarar edilmektedir. Başarısız projeler incelenerek kayıtlar tutulmaya çalışılmış ve bu başarısızlıkların nedenleri tespit edilmeye çalışılmıştır. Başarısızlığa neden olan hatalar iletişim eksikliği, programlama hataları, ihtiyaç değişikliği, zaman baskısı, dokümantasyon eksikliği, geliştirme araçları eksikliği gibi sebepler olarak tespit

edilmiştir. Buna ek olarak başarılı projeler de incelenerek en iyi pratikler tespit edilmiştir [33].

2004 yılında The Standish Group tarafından yayınlanan "The Choas Report" adlı araştırma sonuçlarına göre ABD'de yazılım firmalarının geliştirdiği 30000 yazılım projesi incelendiğinde, bu projelerin sadece %34'ünün başarılı bir şekilde sonuçlandığını, %15'inin başladıktan sonra iptal edildiği; %51'inin ise tartışmalı, yani zamanında bitirilmemiş ya da gereksinimleri karşılamadan sonlanmış projeler olduğunu belirtilmektedir.

Yazılım testinin önem ve gerekliliğini ortaya koyan en önemli yazılım felaketlerinden bazıları aşağıda verilmiştir [18].

Denver Havaalanı Otomatik Bagaj Sistemi: ABD'nin Denver kentindeki uluslararası havaalanındaki uçuşlardaki beklemeleri azaltmak, daha az maliyetle daha hızlı bagaj hizmeti sunmak için geliştirilen otomatik bagaj sistemi yazılımı, yazılım hataları nedeniyle planlanan zamandan yaklaşık 16 ay sonra hizmete girmiş ve bu gecikmenin günlük maliyetinin 1 milyon dolara yakın olduğu hesaplanmıştır. O zamandan beri çeşitli sorunlarla çalışan yazılım için iş görmeyeceği belirlenerek yenilenme kararı alınmıştır.

Ariane 5 Patlaması: Avrupa Uzay Ajansı tarafından geliştirilen 7 milyar Euro'luk uydu taşıma amaçlı füze fırlatıldıktan 37 saniye sonra roket kontrol yazılımındaki hatadan dolayı havada infilak etmiştir. Ariane 5 füzesinde kullanılan kodlar Ariane 4'te kullanılan kodlardan geliştirilmiş modül testleri gerçekleştirilmesine rağmen sistem testleri gerçekleştirilmeden kullanılmıştır. Bir değişkendeki taşma sonucu, kontrol sistemi devre dışı kalarak 500 milyon dolarlık kayba sebep olmuştur.

Hedefi Iskalayan Patriot Füzeleri: Körfez savaşı esnasında Patriot füzelerinden biri hedefi ıskalayarak 28 Amerikan askerinin ölümüne sebep olmuştur. Yapılan incelemede füzelerin zaman hesaplamasında kullanılan 24 bitlik değişkende oluşan hatanın buna sebep olduğu ortaya çıkmıştır.

Therac-25 Tedavi Cihazi: Tümörlerin yok edilmesinde kullanılan Therac-25 cihazının tedavi esnasında düşük güç modunu algılayamayıp yüksek güç modunda çalışmasıyla hastalara normalden 100 kat fazla doz verilip, 6 kişinin ölümüne sebep olmuştur. Bu

durumun yazılımsal sebebinin 255 giriş denemesinden sonra tampon belleğin taşması olarak açıklanmıştır.

Yukarıda bahsi geçen felaketler bir yazılım testine tabi tutulmuş olsaydı belki de önlenmiş olabilirdi. Yazılımlarda ortaya çıkan hatalar zaman kaybı, maddi kayıpların yanı sıra ve can kayıplarına da sebep olabilmektedir. Bu sebeple yazılımların test edilmesi önemlidir ve zarara uğrama ihtimalini azaltmaktadır.

2.1.2.Yazılım Test Prensipleri

Yazılım testinden beklenen, test edilecek yazılımın, kullanıcı gereksinimlerine ve beklentilerine cevap verip veremediğinin test edilmesidir [33].

Günümüz yazılımlarında farklı türde hatalar vardır ve bu hatalar kimi zaman bilgi kayıplarına ve değişmelere sebep olmaktadır. Kalite, piyasaya sürülen yazılımlardaki hataları azaltmak için yazılım testi tarafından değerlendirilir. Geleneksel olarak yazılım testi, yazılımdaki mümkün olabilecek hataları bulmayı aramaktan ziyade, yazılımın karakteristiklerini ortaya çıkarmaya çalışır. Kaliteli bir yazılım ürünü ortaya çıkarabilmek için uyulması gereken bazı prensipler bulunmaktadır. Bu prensipler şunlardır [34]:

- Test işlemi, bağımsız gruplar tarafından yapılmalı,

- Test için en iyi personel seçilmeli,

- Test, maksimum hata sayısı elde etme amacıyla planlanmalı,

- Geçersiz ve beklenmeyen giriş durumları, geçerli durumlar kadar iyi test edilmeli,

- Test esnasında test altındaki yazılımda değişiklik yapılmamalı,

- Test durumları ve test sonuçlarını içeren test raporu hazırlanmalı,

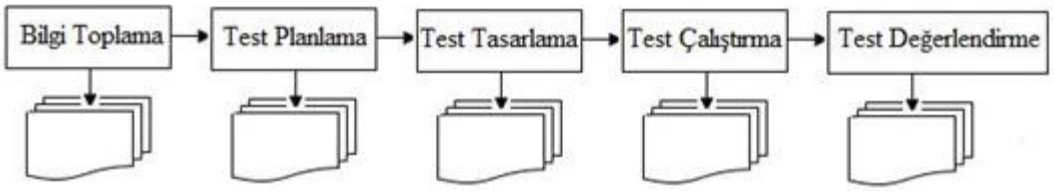
- Mümkünse, beklenen sonuçlar belirlenmeli ve rapora dahil edilmeli,

- Test ilerledikçe planlanmalı ve daha sonra güncellenmeli ve

- Uygun bir test metodu seçilmelidir.

2.1.3.Yazılım Test Süreçleri

Yazılım test süreci planlı bir şekilde gerçekleşen bir dizi eylemden oluşur. Bu süreçte yazılım hatalarına odaklanılır. Aşağıdaki şekilde sadeleştirilmiş yazılım test süreci gösterilmektedir.



Şekil 2.1. Sadeleştirilmiş Yazılım Test Süreci [18].

Bilgi Toplama: Projeyle ilgili bilgilerin elde edildiği ve projeyi anlamaya yönelik çalışmaların yapıldığı süreçtir.

Test Planlama: Testin kapsamının, testin stratejisinin, test ortamının, hangi yazılım parçalarının test edilip edilmeyeceğinin, proje kapsamında amaçlanan test eylemlerinin belirlenip doküman oluşturulduğu aşamadır. Yazılım projesinin kapsamına bağlı olarak birim test planı, yazılım test planı, sistem test planı, kabul test planı gibi birden fazla test planı geliştirilebilir.

Test planlarının geliştirilmesindeki amaç proje kapsamındaki test stratejisinin tanımlanması, kaynak paylaşımının planlanması, sorumlulukların, risklerin ve önceliklerin açıklığa kavuşturulmasıdır [27].

Test planının içeriğinde testin amacı, test stratejisi, test geliştirme ortam özellikleri(donanım, yazılım, ağ alt yapısı, gerekli test araçları vb), test takvimi bulunur.

Test Tasarımı: Test ortamının hazırlandığı süreçtir. Test esnasında kullanılacak yazılım donanım ortamı belirlenir test caseler yazılır ve test yordamları hazırlanır.

Test Çalıştırma: Birim testlerin yapılması ile test çalıştırma süreci başlar. Test koşturmada son adım kullanıcı kabul testleridir. Bu testlerde sistemin kullanıcı gereksinimlerini karşıladığı doğrulanır.

Eğer testler, gereksinimleri test etmezse, o testler geçersiz sayılır. Test verileri, testlerin amacını yansıtmazsa, o testler de geçersizdir [35].

Testler çalıştırılırken test mühendisleri tarafından genel olarak şu adımlar izlenir:

1. Yazılım ekibi tarafından test edilecek yazılım oluşturulur.

2. Test ekibi gelen yazılıma uygulanacak test durumlarına karar verir.

3. Yazılımın test edilebilir olduğuna karar verilir.

4. Yazılım teste kabul edilirse testler başlar.

5. Testlerde bulunan hatalar raporlanarak yazılım ekibine bildirilir.

6. Bulunan hatalar yazılım ekibi tarafından düzeltilir ve yeni sürüm hazırlanır.

7. Test ekibi yeni gelen yük ile yineleme testlerini yapar.

8. Bu adımlar, müşteriye yazılımın son kabul edilebilir versiyonu verilinceye kadar devam eder.

Test Değerlendirme: Gerçekleştirilen testlerin sonuçlarının değerlendirildiği, test esnasında belirlenen yazılım hatalarının incelenip, düzeltildiği aşamadır.

2.1.4.Yazılım Test Düzeyleri

Yazılım testleri temelde 4 ayrı aşamada gerçekleşir. Testlerin ayrıntı düzeyi arttıkça test edilen parçanın büyüklüğü azalır. Bu test aşamaları şunlardır [18]:

- Birim Testleri
- Tümleştirme Testleri
- Sistem Testleri
- Kabul Testleri

2.1.4.1. Birim Testleri

Metot, modül veya prosedür gibi bir kod parçasının kendisinden beklenen işlevselliği doğru olarak yerine getirip, getirmediğini ve hata içermediğini göstermek için gerçekleştirilen testlerdir [36]. Birim testler başarılı ve kaliteli yazılım ürünleri ortaya çıkartmak için kullanılabilecek önemli bir test aşamasıdır. Bir yazılım projesinde birim testler başarılı bir şekilde gerçekleştirilirse, diğer test aşamalarında daha başarılı sonuçlar elde edilmesi muhtemeldir.

Birim testi yapılmasının en kolay yolu bir yazılım test aracı kullanılmasıdır. Eğer bir test yazılım aracı kullanılmayacaksa birim test kodları yazılım geliştirmeye paralel olarak geliştirilmelidir. Bu nedenle birim testler yazılım geliştirmenin bir parçası olarak düşünülmeli ve planlama buna uygun olarak gerçekleştirilmelidir. Gereksinimler veya kod güncellendikçe birim test kodları da güncellenmelidir.

Başarıyla sonuçlanan birim testler, yazılım tümleştirme ve sistem testlerinden önce geliştirilen yazılımın genel olarak istenen gereksinimleri karşıladığının bir göstergesidir. Ancak yazılımın hatasız olduğunu göstermez. Geliştirilen yazılımdan beklenenlerin tam olarak karşılandığını ortaya koymak için entegrasyon ve sistem testleri ile arayüzlerin testlerinin de gerçekleştirilmesi gerekir.

Birim testlerinin amacı, geliştirilen ve derlenebilen en küçük kod parçalarının (metot, prosedür, sınıf vb.) kendilerine ait görevleri doğru şekilde yerine getirdiğinin doğrulanmasıdır. Bu amaçla, birim testler yapılırken test edilecek birim, diğer birimlerden izole edilir. Test edilen birim için gerekli olan girdileri sağlayacak diğer birimlerin yerine o metotların koçanları (stub) kullanılır. Böylece diğer birimdeki olası bir hatanın test edilen birimi etkilemesi engellenir. Test edilecek birimin icra edeceği göreve göre test durumları oluşturulur. Bu durumlar kullanılarak test edilecek birim çalıştırılır ve elde edilen sonuçlar beklenen sonuçlarla karşılaştırılarak testlerin başarılı olup olmadığına karar verilir. Eğer test başarısız olmuşsa birimin kodu tekrar gözden geçirilir, gerekli düzeltmeler yapılır ve tekrar test edilir. Bu işlem, tüm birimler başarılı bir şekilde kendilerine ait birim testleri geçinceye kadar devam eder [18].

2.1.4.2. Tümleştirme Testleri

Yazılım projelerinde birim testlerin başarılı bir şekilde sonuçlandırılmasından sonra tümleştirme (entegrasyon) testleri başlar. Birim testlerde amaç modüllerin ayrı ayrı kendilerinden beklenen işlevleri yerine getirdiğinin doğrulanmasıyken tümleştirme testlerinin amacı bir araya gelerek entegre edilmiş olan bir yazılımın içerisindeki bileşenleri birbirleriyle uyum içerisinde doğru bir şekilde çalıştığının ve bileşenlerin kendilerine ait gereksinimleri yerine getirdiğinin doğrulanmasıdır. Bu testlerin yapılması için kullanılacak test durumları, yazılım gereksinimleri ile arayüz gereksinimlerinden çıkartılır. Tümleştirme testlerinde kara kutu test yöntemi kullanılır. Testlerde hem pozitif hem negatif test durumları uygun parametreler kullanılarak çalıştırılır. Sonuçlar, beklenen değerlerle karşılaştırılarak sistemin kendisinden beklenen davranışları gerçekleştirdiği doğrulanır [37].

Tümleştirme testlerin yapılırken big bang, aşağıdan yukarıya veya yukarıdan aşağıya olmak üzere farklı stratejiler kullanılabilir.

Big bang stratejisine göre geliştirilen tüm üst ve alt düzey modüller birbirleriyle entegre edildikten sonra testler başlar. Bu stratejide sistem bir bütün olarak ele alınarak testler yapıldığından testlerin gerçekleştirilmesi hızlıdır ve zamandan tasarruf edilir. Ama bir hata çıktığında bu hatanın hangi seviye katmanda oluştuğu veya hangi modülden kaynakladığının tespiti zordur.

Aşağıdan-yukarıya tümleştirme test stratejisi öncelikle birim testlerin yapılmasıyla başlar. Birim testleri başarıyla tamamlanan alt düzey modüller birbirleriyle entegre edilir. Bu entegrasyondan sonra alt düzel modüller ile ilgili entegrasyon testleri yapılır. Bu testlerin başarılı şekilde sonlandırılmasından sonra bir üst katman modülleri sistemi entegre edilir ve gerekli entegrasyon testleri yapılır. Bu entegrasyon tüm sistem inşa edilinceye ve entegrasyon testleri başarıyla sonuçlanıncaya kadar devam eder. Bu yaklaşımda temel şart, entegre edilecek modüllerin birim testlerinin başarıyla tamamlanmış olmasıdır.

Yukarıdan-aşağıya tümleştirme test stratejisi, öncelikle sistem için en üst modülün (en dış modül, genellikle kullanıcı grafik arayüz modülü) test edilmesiyle başlar. Bu modülün ihtiyaç duyduğu alt modüllerin koçanları kullanılır. Bu yaklaşımda birçok koçan yazılması gerekir. En üst modül başarılı şekilde test edildikten sonra bir alt düzey

modüller sistemle bütünleştirilir. Bu entegrasyondan sonra gerekli alt düzey tümleştirmesi testleri yapılır. Bu durum en alt düzey modüller entegre edilinceye ve entegrasyon testleri başarıyla sonuçlanıncaya kadar devam eder. Bu yaklaşımda entegrasyon testleri kara kutu testleri olarak başlar ve gittikçe saydam kutu testlerine döner.

2.1.4.3. Sistem Testleri

Sistem testleri, geliştirilen yazılımın performans, güvenilirlik, işlevsellik gibi özelliklerini değerlendiren testlerdir. Birim ve entegrasyon testlerinde geliştirilen yazılımın tasarıma uygun olarak geliştirildiği doğrulanır. Sistem testleriyse müşterinin sistemden istediklerini doğrulamayı amaçlar. Bu nedenle sistem test durumlarında sistem gereksinimleri temel alınarak sistemin çalışacağı, gerçek ortamda karşılaşılabilecek olan senaryolar tanımlanır.

Sistem testleri, kullanıcı kabul testlerinden bir önceki adım olarak yazılım ve donanım entegrasyonundan sonra "sistem test planına" göre yapılır. Sistemin işlevsel, işlevsel olmayan gereksinimlerinin doğrulanması hedeflenir.

Sistem testlerinin ilk adımı olarak işlevsel gereksinimlerin doğrulanması gerçekleştirilir. Bu doğrulama sonucunda sistemin işlevsel olarak çalıştığı gözlemlenir. İşlevselliği yönünden doğrulanan sistem üzerinde bir sonraki adımda, işlevsel olmayan gereksinimlerin karşılandığı gösterilmek amacıyla işlevsel olmayan testler yapılır. Bu testlerden bazıları:

Stres Testleri: Sisteme girdi oranı sistem tasarım oranını aştığı zaman sistemin davranışını gözlemlemek üzere gerçekleştirilen testlerdir.

Performans Testleri: Sistem çıktılarının belirlenen ve kabul edilebilecek olan zaman dilimi içerisinde üretebildiğinin değerlendirilmesinin yapılabilmesi için gerçekleştirilen testlerdir.

Konfigürasyon ve Uyumluluk Testleri: Geliştirilen sistemin farklı platformlarda ve donanımlarda nasıl davrandığının değerlendirilmesi için yapılan testlerdir.

Güvenlik Testleri: Sistemin izinsiz kullanım teşebbüslerindeki davranışlarının değerlendirilmesi için yapılan testlerdir.

Kullanılabilirlik Testleri: Kullanıcı-sistem etkileşimini ve ergonomisini değerlendirmek üzere yapılan testlerdir.

Geri Alma Testleri: Bir hata durumunda sistemin otomatik veya elle yeniden normal duruma dönmesini değerlendirmek için yapılan testlerdir.

Kullanıcı Arayüzü Testleri: Kullanıcının ve yazılımın grafik gösterimi olarak nasıl bir etkileşim içerisinde olacağını, kullanıcının klavye, ekran veya fare ile sisteme vereceği girdilerin sistem tarafından nasıl işleneceğini değerlendirmek için yapılan testlerdir.

İşlevsel olmayan testleri tamamlanan sistem artık doğrulanmış olarak kullanıcı kabul testlerine hazır demektir. Sistem testleri sırasında ortaya çıkan hatalar proje hata yönetimi sürecine göre raporlanır ve gerekli düzeltme işlemleri yapılır. Gerekli düzeltmelerden sonra, düzeltmelerden sistemin geri kalanının etkilenmediğinin değerlendirilmesi için sistem üzerinde yineleme testleri yapılır.

2.1.4.4. Kabul Testleri

Proje kabul testleri müşteri gereksinimlerinin doğrulanması ile gerçekleştirilir. Bunun için her bir kullanıcı gereksinimini doğrulayacak olan test durumları ve test senaryoları oluşturulur. Geliştirilen sistem, tanımlanan bu test durumları ve test senaryoları müşterinin de katılımıyla "kabul test planına" uygun olarak koşturulur. Sistemin tamamının bu testlerden geçmesi ile sistem, müşteri tarafından kabul edilmiş olur.

Kabul testleri için şu adımlar bir süreç olarak işletilir [18]:

1. Müşteri gereksinimleri belirlenir ve belgelendirilir.

2. Kabul test planı hazırlanır.

3. Müşteri gereksinimlerini doğrulayacak test durumu ve senaryoları hazırlanır.

4. Test hazırlık gözden geçirme toplantısında hazırlanan test durumu ve senaryoları onaylatılır.

5. Kabul testleri için gerekli test ortamı hazırlanır.

6. Sistem testlerinin başarıyla sonuçlanmasından sonra kabul testleri yapılır.

7. Kabul test planında belirtilen kabul kriterlerine ulaşılıncaya kadar testlerin yapılmasına devam edilir.

8. Test sonuçları belgelendirilir ve müşteriye onaylatılır.

Kabul testleri, bir yazılım projesinin başarısında en kritik adımdır. İyi planlanmış ve belgelendirilmiş kabul testleri, projenin başarısına büyük katkı sağlar, müşteri memnuniyetini arttırır.

2.1.5. Yazılım Test Teknikleri

Yazılım testleri yapılırken test edilecek yazılıma bakış açısına göre üç test tekniği vardır. Bunlar kara kutu, saydam kutu ve gri kutu test teknikleridir.

2.1.5.1. Kara Kutu Testi

Test edilecek yazılımın iç işleyişine bakılmaksızın yapılan testlere kara kutu testleri denilir. Bu yaklaşımda test edilecek yazılımın, sonucu bilinen bir davranışını doğrulamak için davranışın gerektirdiği girdi değerleriyle çalıştırılarak sınanır. Daha sonra yazılımın bu girdi karşısında elde edilen çıktısı, beklenen sonuçla karşılaştırılır. Bu yaklaşımın kara kutu olarak adlandırılmasının nedeni, uygulamayı test edecek kişinin yazılımın içyapısıyla ilgilenmemesidir. Bu nedenle bu testler yazılım geliştiricilerden çok, test ekipleri tarafından gerçekleştirilir.

Kara kutu testinde bir yazılım parçası test ediliyorsa, test mühendisi bu yazılım parçasının girdisini ve buna karşılık sistem çıktısını bilir. Fakat bu çıktıya nasıl ulaşıldığıyla ilgilenmez. Kara kutu testlerinin amacı, verilen girdilerle istenilen çıktının elde edilmesidir. Bu nedenle yazılımın iç işleyişiyle ilgili diğer bilgilerle ilgilenilmez. Kara kutu test tekniğinde diğer önemli bir nokta da geliştirilen yazılımın tasarımından veya kodlanmasından kaynaklanabilecek hataların bulunmasıdır. Bu amaçla kara kutu testlerinin yapılması ve istenilen amaca ulaşılabilmesi için yazılım geliştiricilerle test ekibi birbirinden ayrı çalışmalıdır. Böylece test ekibi, tasarım ayrıntılarını bilerek yazılıma karşı ön yargı taşımaz.

Kara kutu test tekniğiyle geliştirilen yazılım içerisinde şu hata türleri tespit edilmeye çalışılır:

1. Doğru olmayan veya hiç olmayan işlevlerin tespiti

2. Arayüz hataları

3. Performans hataları

4. Veri tabanlarına ulaşma hataları veya veri yapılarındaki hatalar

5. Başlatma veya sonlandırma hataları

6. Sınır değer hataları [18].

Kara kutu test tekniği basit olarak Şekil 2.2.'de gösterilmektedir.

Şekil 2.2. Kara Kutu Test Yaklaşımı.

Kara kutu testinin avantajları/dezavantajları [38]:

Avantajlar:

- Yazılımlarda hataların bulunması için etkin ve hızlı bir tekniktir.
- Test durumları yazılırken gereksinimlerden hareket edildiği için gereksinimlerdeki tutarsızlıkların ve belirsizliklerin belirlenmesinde önemlidir.
- Testi gerçekleştirecek kişinin yazılımın ayrıntılarını bilmesine gerek yoktur.
- Test ekibi ve kod geliştiriciler birbirinden bağımsız çalışabilirler.
- Testçiler gereksinimleri doğrulamak ve gereken testleri gerçekleştirmek için yazılıma kullanıcı gözüyle bakarlar. Bu da kod geliştiriciler tarafından fark edilemeyen pek çok olası hatanın ve eksiğin bulunmasına yardımcı olur.
- Testleri yapacak kişilerin sistem hakkında teknik ayrıntı bilmesine gerek yoktur.

Dezavantajlar:

- Kara kutu testleri yazılımın belirli parçasını hedeflemez. Bu nedenle birçok hata tespit edilmeden kalabilir, bunlar için başka testler gerekir.
- Sadece belirli sayıda girdi değeriyle testler yapılır. Tüm girdiler ile testlerin yapılması sonsuza kadar sürer.
- Yazılım içerisinde bazı kod parçalarında birden fazla test yapılırken bazı kod parçaları hiç test edilmeden kalabilir.
- Açık ve yalın olmayan gereksinimlerin test durumlarını tasarlamak ve testlerini yapmak kara kutu test tekniğinde zordur.

Kara Kutu Test Stratejisi

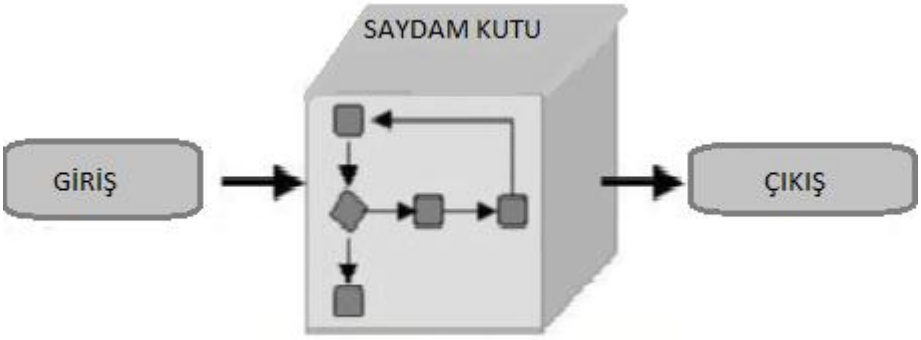
Kara kutu test tekniği tümleştirme, sistem ve kabul test aşamalarında kullanılır. Bu testlerde en verimli sonuçlara ulaşmak için şu kara kutu test stratejisi izlenebilir:

- Kara kutu testleri rasgele belirlenmiş girdilerle gerçekleştirilmelidir.
- Yazılımın sağlamlığının kontrolü için belirtilen aralığın dışındaki değerlerin de testi yapılmalıdır.
- Sınır değerler mutlaka test edilmelidir.
- Değer artışlarında artış miktarı ayrıca test edilerek doğrulanmalıdır. Artış miktarı dışı artımlarda yazılımın tanımlı davranışına göre hareket ettiği doğrulanmalıdır.
- Sayısal girişlerde sıfır değeri mutlaka girdi olarak sınanmalıdır.
- Özellikle gerçek zamanlı sistemlerde stres testi yapılmalıdır. Programın aşırı yüklenme altında nasıl çalıştığı test edilmelidir.
- Kara kutu testlerinde diğer bir amaç gereksinimlerin doğrulanması olduğundan her bir gereksinim için en az bir test durumu yazılmalı ve bu şekilde gereksinim kapsama gerçekleştirilmelidir [18].

2.1.5.2. Saydam Kutu Testi

Saydam kutu testleri yazılımın içyapısı bilinerek tasarlanır. Bu nedenle saydam kutu testlerini gerçekleştirenler genellikle sistemin içyapısını bilen yazılımcılardır. Saydam kutu testiyle programın içyapısındaki birimlerin içindeki hatalar araştırılır. Kaynak kod, saydam kutu testlerinin en önemli girdisi olduğundan, koda ulaşım olmadan saydam kutu testleri yapılamaz. Bunun yanında risk analizleri ve tasarım kısıtları da saydam kutu testlerinin planlanmasında, test stratejisinin belirlenmesinde, test araçlarının seçilmesinde ve test verilerinin oluşturulmasında kullanılır.

Saydam kutu testleri veri, kontrol ve bilgi akışlarının, kodlama standartlarının, hata yakalama ve ayıklama yapısının analizlerini içerir. Beyaz (saydam) kutu test tekniği basitçe Şekil 2.3' de gösterilmektedir.



Şekil 2.3. Saydam Kutu Test Şeması [27].

Saydam kutu test yaklaşımı kullanılarak yapılan testler şunlardır:

Birim Testler: Saydam kutu testinin en iyi ve yaygın kullanımı birim testlerdir. Birim testler, kod yazanların belirli bir kod parçasının görevini doğru şekilde yerine getirip getirmediğini anlamak için yapılan testlerdir.

Statik ve Dinamik Analizler: Statik analiz, kod içerisindeki muhtemel hataları bulmak için yapılan kod üzerindeki incelemeleri içerir. Dinamik analizlerse kodun çalıştırılmasını ve çıkan sonucun analiz edilmesini içerir. Bu nedenle saydam kutu testleri kaynak koda ulaşım hakkı getirir.

Deyim Kapsama (Statement Coverage): Bu tür testte kod çalıştırılarak kod içerisinde yer alan her deyimin en az bir kez çalıştırılması hedeflenir. Böylece her bir deyimin bir yan etki göstermeden çalıştığı doğrulanır. Kod içerisinde çalıştırılmayan deyim olmadığı da doğrulanır.

Dal Kapsama (Branch Coverage): Hiçbir kod düz bir akışla yazılmaz. Kod içerisinde karar noktaları bulunur ve bu noktalardan kod yan dallara ayrılır. Dal kapsama ile program içerisinde yer alan tüm dalların kendilerinden beklenildiği şekilde çalıştığı doğrulanır.

Yol Kapsama (Path Coverage): Kod içerisindeki tüm yolların test edilmesidir.

Saydam kutu testiyle hata ve yanlışlar en erken safhada ve en hızlı şekilde bulunur. Böylece entegrasyon ve sistem testleri daha hızlı yapılabilir ve bulunan hata sayısında önemli bir azalma gözlemlenebilir.

Saydam kutu testinin avantajları/dezavantajları [38]:

Saydam kutu test tekniğiyle sınanan birimin veya modülün belirlenen girdiyle beklenen çıktıyı vermesi hedeflenir. Ancak bu çıktıyı nasıl verdiği ve kod içinde hangi yollardan geçildiği de incelenir.

Avantajlar:

- Kod içerisinde gizli kalmış mantıksal hatalar bulunur.
- Saydam kutu testleriyle yazılan kodun optimizasyonuna katkıda bulunulur.
- Kod içindeki fazla satırlar ayıklanarak ölü kod parçaları bulunur.
- Kaynak kodun analiz edilmesi ve bu analize göre testlerin gerçekleştirilmesiyle yazılım içerisindeki hatalar daha erken aşamada ve daha hızlı bulunur.
- Yazılımın geliştirilmesi için belirlenmiş olan kodlama rehberine uyumluluk, tasarım kararlarına kodlama içerisinde uyulup uyulmadığı saydam kutu testleriyle net olarak görülür.

- Saydam kutu testleriyle yazılımcıların kod geliştirme yetenekleri desteklenir ve güçlendirilir.

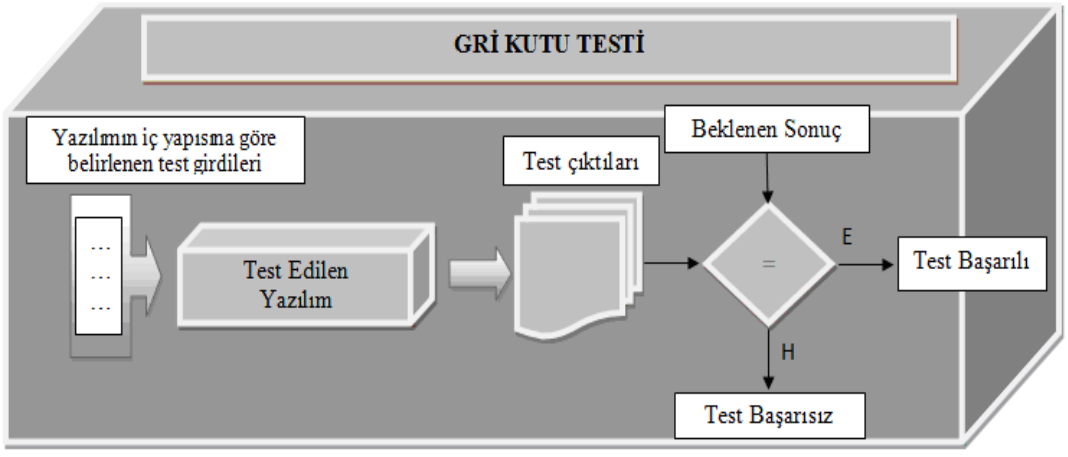
Dezavantajlar:

- Eğer birim tümleştirme testleri test ekibi tarafından yapılacaksa, bu iş için kodun iç yapısının bilinmesi gerekir. Bu da maliyeti arttırır.
- Saydam kutu testleriyle sadece modül ve birimlerin iç işleyişleri test edildiğinden, tümleştirmeden sonra ortaya çıkabilecek olan hatalar tespit edilemez.

2.1.5.3. Gri Kutu Testleri

Gri kutu testleri, saydam kutu ve kara kutu test tekniklerinin birlikte kullanılmasıdır. Bu testlerde gereksinimleri doğrulayacak test durumları kodun içyapısı esas alınarak yazılır. Böylece gereksinimler doğrulanıp, yazılımın içyapısı sınanmış olur.

Gri kutu test yaklaşımı; testin tasarımıyla ilgilenir. Testleri kara kutu testi gibi uygulanır. Ancak test tasarlanırken işlevin içsel veri ve algoritma yapısı da göz önünde bulundurulmaktadır. Gri kutu testlerinde test ekibi yazılımın içyapısını bildiğinden, yazılımın tasarımına ve kod yapısına karşı şartlanma olabileceğinden bazı hataları ortaya çıkartabilecek testler yapılmadan kalabilir. Şekil 2.4' de Gri kutu test şeması gösterilmektedir.



Şekil 2.4. Gri Kutu Test Şeması [18].

3. BULGULAR VE TARTIŞMA

3.1. MOBİL UYGULAMALAR VE TEST ZORLUKLARI

Mobil uygulama akıllı telefon tablet bilgisayar gibi bir ortamda çalışmak üzere tasarlanmış yazılım uygulamalarıdır [39]. Bir uygulama elektronik cihaz (kamera, mp3 okuyucu, akıllı telefon) üzerinde taşınabilip çalışıyorsa mobil olarak kabul edilir.

Mobil uygulamalar mobil cihazlar aracılığıyla dünya genelindeki insanlarla işbirliği ve dosya paylaşımı için uzak yerlerdeki uygulamalara erişim sağlar. Son yıllarda mobil cihazlarda görülen donanımsal ilerlemeler ve networklerin gelişimi mobil uygulamaların başarısına önemli bir katkı sağlamıştır. İlk mobil uygulama yaklaşık on yıl önce geliştirilmişken mobil uygulamaların gelişimi kısa zamanda hızlı bir ilerleme göstermiştir ve mobil uygulamaların popülaritesi gün geçtikçe artmaktadır. Giderek karmaşık hale gelen uygulamaların büyüme ve kalite isteğini karşılayabilmek için yazılım geliştirme süreci büyük önem taşımaktadır. Bu sebeple yazılımlar geliştirilirken geliştiricilerin büyük çaba ve özen göstermesi gerekmektedir. Geliştirme sonrasında yazılımdan beklenen en önemli ölçüt kalitedir ve yazılımların doğruluğunu ve geçerliliğini kontrol etmenin en iyi yolu testtir. Ancak mobil uygulamalar test edilirken bazı zorluklarla karşılaşılır.

Ballard'a göre mobil uygulamaların testi ve gelişimi için 3 faktörü göz önüne almak gerekir. Mobil içerik, mobil kullanıcı ve mobil uygulamalar. Mobil içerik faktörü farklı ağ ara yüzleri, düşük batarya, cihazın ekran ve klavye büyüklüğü gibi ortam kısıtlarıyla ilgilidir. Mobil kullanıcı faktörü uygulama ile etkileşim halindeki kullanıcının hareketini göz önüne alır. Mobil uygulama faktörü ise uygulamanın üzerinde çalıştığı cihaz ile karakterize edilir [40].

Mobil uygulamaların çoğu tipine bağlı olmaksızın genellikle küçük ekipler halinde geliştirilir. Bu ekip yazılımı en kısa sürede ürünü markete sunma baskısı altında oluşturur ve test için ayrılan zaman kısadır.

Mobil cihazlar farklı tip ve özellikte donanımlarla donatılarak üretilmektedir. Bir mobil cihaz genellikle şu donanım bileşenlerini içerir: dokunmatik ekran, tuş takımı, hücresel bağlantı, GPS alıcısı, çoklu giriş aygıtları, Wi-Fi ve Bluetooth. Bu bileşenlerin birbirleri ile ve işletim sistemiyle farklı şekillerde etkileşir ve test edilirken tüm bu ihtimalleri göz önüne almak, oluşabilecek uyumluluk ve performans sorunlarını test etmek büyük bir çaba gerektirir. Bu durum mobil uygulamaların çok platformlu yani birden fazla işletim sistemini destekleyecek şekilde geliştirilmesini ve test edilmesini gerektirmektedir. Aynı uygulama kodu farklı cihazlarda ekran boyutu ve çözünürlüğe bağlı olarak farklı şekillerde görünebilir. Farklı mobil cihazların farklı yazılım özellikleri ve farklı donanım bileşenlerinin varlığından dolayı test etmek zorlaşır. Bu özellikle Android işletim sistemi için doğrudur çünkü sayısız cihaz kombinasyonu mevcuttur.

Ayrıca mobil işletim sistemlerinin çeşitliliği hedefi tutarlı bir kullanıcı sağlamak olan mühendislik ekipleri için dev bir sorun teşkil etmektedir. Örneğin aynı uygulamada aynı ortamlarda çalışan farklı iki telefonun sensörleri aynı girdiye karşılık farklı çıkışlar hesaplayabilir [41].

Mobil cihaz sınırlandırmaları sebebiyle mobil cihazlar üzerinde test yapmak pratik değildir. Uygulamalar ilk olarak emülatör kullanılarak test edilip daha sonra hedef cihaz üzerinde testleri gerçekleştirmek gerekmektedir. Manuel test gerçekleştirmek için tüm hedef cihazlarda deneme yapmak testin her aşamasında karmaşık ve pahalıdır.

Donanım platformundaki kaynak kısıtlılığı mobil uygulamaların gelişimi için başka bir zorluktur. Teknolojik gelişmelere rağmen limitli kullanıcı ara yüzü ve işlemci kapasitesi, bellek kısıtlamaları ve enerji gereksinimi ihtiyacı mobil uygulamaların testi için birer zorluk olarak görülür. Mobil uygulama kaynak kullanımı sistemin yanlış işleyişinden performans bozulmalarından etkilenmemesi için sürekli izlenmesi gereken önemli bir bileşendir. Bu kaynaklardaki kısıtlamalar hatayı ortaya çıkarabilmek için özel test faaliyetleri gerektirir. Örnek olarak bir mobil uygulama test edilirken aşırı kaynak kullanımından ortaya çıkabilecek hataları gidermek amacıyla uygulama düşük bellek seviyesi ve yüksek hafıza kullanımı şartları altında test edilmelidir.

Mobil uygulamaların bağlantı değişkenliği, bant genişliği kısıtı ve kablosuz ağ kullanımı da mobil uygulama kısıtları arasında yer alır. Mobil cihazlar hız ve güvenliğin

değişken olduğu mobil ağa giriş yapılarak kullanıldığı için mobil uygulamalar daima onlinedır. Düşük bant genişliği, yavaş ve güvenilir olmayan ağlar mobil uygulamalar için bir engeldir.

Mobil platformun açık kaynak kodlu yapısı kullanıcılara uygulamaları kolayca indirip yüklemesine izin veren yapısından dolayı bir mobil uygulama aynı ortamda çalışan uygulamalar için kolayca hedef haline gelmektedir ve mobil uygulamalara karşı güvenlik saldırı ihtimalini arttırmaktadır. Bu sebeple mobil uygulama testi olası güvenlik açıklarını ortaya çıkarmalıdır.

Mobil uygulamalar mobil test GUI uygulamalarının kullanıcı etkileşimine odaklanmış olduğundan özel zorluklar doğurmaktadır. GUI uygulamaları olay güdümlüdür ve kararsız bir yapıya sahiptir. Bu durum mobil uygulamaların testini zorlaştırır.

Android testleri de tüm bu zorluklara ek olarak kendine özgü zorluklar içerir. Android uygulamaları etkili test tekniklerine, stratejilerine ve araçlarına ihtiyaç duyar. Test esnasında Android yapısına ait özel problemler, açık konular ve çeşitli sorunlar ortaya çıkar. Bunların önemli bir sebebi geliştiricilerin acemiliğidir. Tüm mobil uygulamalarda olduğu gibi Android uygulamalarının test stratejileri ve gelişimi taşınabilir cihazlar üzerindeki kısıtlamalardan etkilenir. Bu faktörlerin tümü Android mobil uygulamaların testi için yeni zorluklar ortaya çıkarır.

3.2. ANDROİD TEST ÇERÇEVESİ [42]

Android test çerçevesi her düzeyde uygulamanın test edilebileceği güçlü araçlar ve mimari sağlayan geliştirme ortamının ayrılmaz bir parçasıdır. Test çerçevesinin en önemli özellikleri şunlardır:

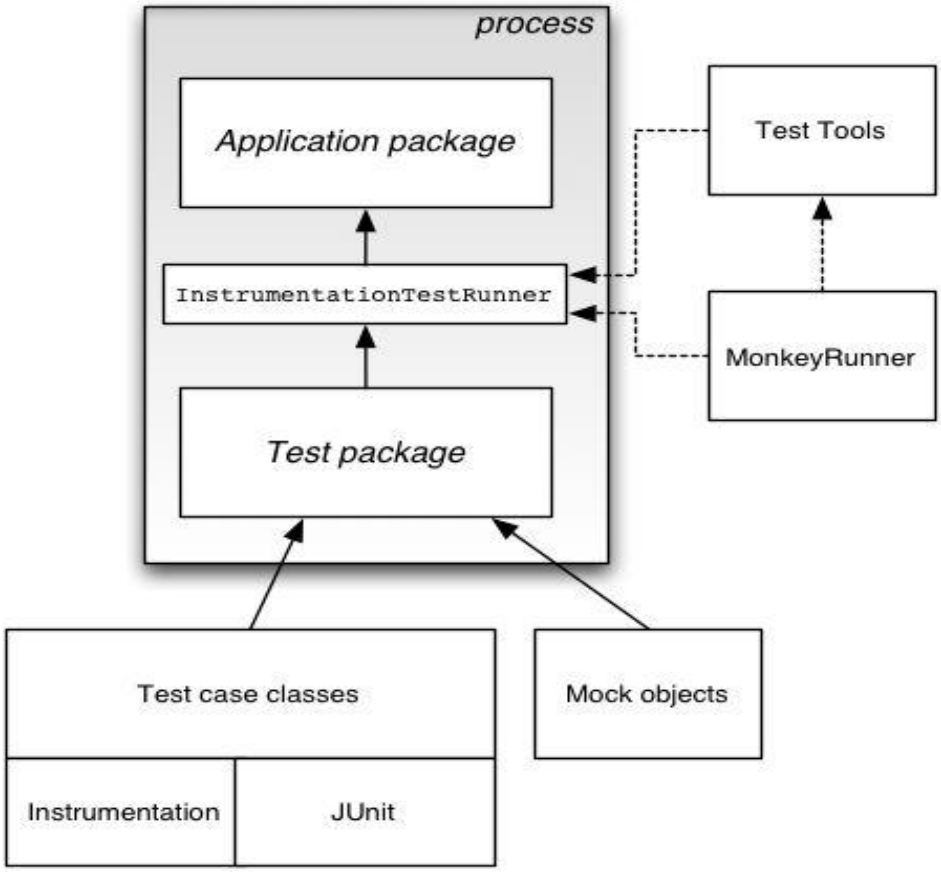
- Unit test, yazılımdaki en temel test çeşitidir. Yazılan fonksiyonların çalışıp çalışmadığının testlerini yapmak için kullanılır. JUnit ise Java Unit testlerine verilen isimdir. Android testleri JUnit'e dayanmaktadır. Bu sebeple Android bileşenlerini test etmek için JUnit test classı kullanılabilir.

- Android JUnit uzantıları bileşene özgü test case sınıfları sağlar. Bu sınıflar bir bileşenin yaşam döngüsünü kontrol etmeye yardımcı olan methodlar ve sahte nesneler(mock object) üretmek için yardımcı yöntemler sağlar.

- Testler ana uygulama paketleriyle benzer oldukları için yeni teknikler ve araçlar kullanmak gerekmez.
- Komut satırı ya da ADT yardımıyla test altındaki proje hakkında bilgi alınabilir.

Android Test ortamı şu temel özellikleri içerir:

- Android sistem nesnelerine erişimi sağlayan JUnit
- Uygulamaları inceleyen ve testleri kontrol eden bir çerçeve
- Yaygın olarak kullanılan Android sisteminin nesnelerinin Mock sürümleri
- Instrumentationlu ya da instrumentationsuz test suiteleri ya da testleri tek tek çalıştırmak için araçlar
- Eclipse için ADT Plugin ve komut satırında testler ve test projeleri yönetmek için destek



Şekil 3.1. Test Çerçevesinin Özet Yapısı [42].

Şekil 3.1. Android test çerçevesinin özet yapısını göstermektedir ve aşağıda listelenen bileşenlerden oluşmaktadır.

Test Yapısı: Androidde Test projeleri, test paketleri, test case sınıfları standart bir yapı halinde düzenlenmiştir. Android testleri JUnit'e dayanır. Genelde JUnit test, test altındaki bir uygulamanın belli bir bölümünü test eden methoddur.

Test Projeleri: Testler Android uygulamaları gibi projeler halinde düzenlenmişlerdir. Bir test projesi de test paketi ve diğer dosyalar için kaynak kodu olan ve izin dosyası oluşturulabilen bir Eclipse projesidir. Android SDK test projelerini güncellemek ve oluşturmak için komut satırı ve ADT içerir.

Instrumentation: Android sistemindeki kontrol metodlarının bir setidir. Bu methodlar Android bileşenlerin bağımsızlığını ve Android uygulamaların nasıl yükleneceğini

kontrol eder. Instrumentation test altındaki uygulamayı kontrol eder ve uygulamayı çalıştırmak için gerekli mock objelerinin kullanımına izin verir. Çevre ile uygulama etkileşimi bu yaklaşım kullanılarak kontrol edilebilir.

Mock Nesneleri: Mock nesneleri test birimlerinin izolasyonunu sağlamak için gerçek objeleri çağırmak yerine taklit(sahte) objeler kullanır. Kullanılan yöntemlerin doğru olduğundan emin olmak için yapılır ve testleri tekrar tekrar ve bağımsız olarak çalıştırmaya olanak sağlar.

Test Caseler: Test caseler gereksinimlere göre hazırlanan girdi, olay ya da aksiyonların sonucu oluşması beklenen sonuçların belirtildiği dokümanlardır. Test caseler yazılımın temellerini oluşturan gereksinimlerin dizayndaki problemlerin ve eksikliklerin ortaya çıkarılmasını sağlar.

3.3. MOBİL UYGULAMALARDAKİ TEST ZORLUKLARINA OTOMASYON ÇÖZÜMÜ İLE YAKLAŞIM

Mobil uygulamaları test etmenin zorlukları ve karmaşıklığıyla başa çıkabilmek için test otomasyon araçları kullanılabilir. Bir uygulama yazılımındaki hataların belirlenmesi karmaşık bir süreçtir. Bu karmaşıklığı azaltmak için testlerin otomatik bir şekilde yapılması gerekmektedir. Geleneksel olarak, yazılım testleri az sayıda tekrarla ve manuel olarak yapılır. Ancak geliştirilen yazılımlar için sistematik bir yaklaşım gereklidir; örneğin özellikle güvenlik ile ilgili yazılım geliştirmesi yapıldığında sistematik ve izlenebilir bir yaklaşım esastır. Bu amaçla mevcut uygulamada, test durumlarını düzenlemek ve yürütmek için otomatikleştirilmiş araçlar kullanılmaktadır. Test otomasyonu testlerin yürütülmesini kontrol eder ve beklenen sonuçla gerçek sonuçları yazılım kullanılarak karşılaştırmaya yardımcı olur. Harcanan zamanı azaltarak verimliliği arttırır. Sürekli aynı testleri yapmak zamanla bu konudaki dikkati azaltır ve hatayı görme ihtimalini azaltır. Otomasyon sayesinde hata yapma oranı azaltılabilir. Otomasyon olmadan yapılan testlerde performans ve kalite önemli bir sorun olarak ortaya çıkar. Otomasyonsuz yapılan testler testi yapan kişinin yeteneğine bağlı olarak başarılı olurken otomasyonlu testler bir sınırlamaya dahil olmadan tüm platform ve versiyonlarda çalışır. Böylece sonuçlar hızlı bir şekilde elde edilir.

Mobil uygulamalar için genel algı geleneksel uygulamalara göre daha az maliyetli ve ucuz olmaları gerektiğidir. Mobil uygulamaların güvenilir ve doğru olması gereklidir. Maliyeti düşük tutmak güvenilirliği yeterli düzeyde sağlamak için otomasyon önemli bir araçtır.

Yazılım testlerinde meydana gelen hatalar uygulamaların uygulama ara yüzlerinin işletim sistemi ve donanım katmanlarının birlikte çalışabilirlik sorunları yüzünden meydana gelir. Bu durum uygulama seviyesindeki hataları uygulama çerçevesi ya da işletim sistemi hatalarından ayıran farklı katmanlar arasındaki test otomasyonu ihtiyacını vurgular.

Yazılım testinde otomasyonun avantajlarını şöyle sıralayabiliriz:

- İnsan kaynaklı hataları minimize eder
- Daha fazla hatayı daha kısa sürede tespit eder
- Testlerin yeniden kullanımını kolaylaştırır
- Testlerle kapsanan kod yüzdesini artırır
- Testler 7x24 çalışabilir
- Maliyeti düşürür
- Güvenliği sağlamada yardımcı olur.

3.4. ANDROİD TEST OTOMASYON ARAÇLARI

Mobil cihazların test edilme zorluğuna bir çözüm olarak görülen Android uygulamaların test edilmesinde kullanılan açık kaynak kodlu beli başlı test otomasyon araçları listelenmiş ve bu tez çalışmasında kulanımı tercih edilen test aracı olan Robotium hakkında detaylı bilgi verilmiştir.

-Monkey

Stres testi için ideal bir otomasyon aracıdır. Uygulamaya rastgele girdiler göndererek yazılımı test eder. Emülatör ya da mobil cihaz üzerinde çalışabilir. Kullanımı kolaydır ve kısa sürede sonuç verir. Ancak kapsamlı bir test için yeterli değildir.

-MonkeyRunner

Emülatör ya da cihaza dışardan müdahaleyi mümkün kılan bir araçtır ancak uygulamaya müdahale imkânı sağlamaz. Testler Jython dili ile yazıldığından ek bir çaba gerektirir ve testlerin yazılması zordur. Kara kutu testleri için uygundur. Kaynak koduna ihtiyaç duymadan çalışır.

-Robolectric

Robotium ile benzer bir yapıya sahiptir. Robotium testleri için cihaz ya da emülatör gerekirken roboelectric'de testler DVM (Dalvik Virtual Machine) üzerinde çalışabilir bu sebeple daha hızlıdır. Kod yapısı Androide benzerdir.

-Selendroid

Emülatör ve gerçek cihazlarda kullanılabilir. Test esnasında aynı anda birden fazla cihaz ile etkileşime girebilir. Kullanıcı etkileşimini destekler.

-Robotium

- ✓ Otomatik kara kutu testlerinde kullanılan bir çerçevedir.
- ✓ UI test caselerini otomatikleştirmek için kullanılır.
- ✓ Java ve JUnit 3 üzerine inşa edilmiştir.
- ✓ Ücretsiz olarak edinilebilir ve kullanıcı etkileşimlerini otomatik hale getirebilen bir araçtır.
- ✓ Açık kaynak kodlu bir proje olduğundan herkes geliştirme amacıyla katkıda bulunabilir.
- ✓ Android üzerinde Activity, Dialog, Toast, Menu ve Context Menu bileşenlerini destekler.
- ✓ JUnit test çerçevesini kullanır.
- ✓ Geliştirme ortamları: Eclipse, ADT(Android Development Kit), Android SDK(Software Development Kit) ve JDK(Java Development Kit) dir.
- ✓ Robotium desteğiyle, test case geliştiricileri birden fazla Android aktivitesini kapsayan, fonksiyon, sistem ve kullanıcı kabul test senaryoları yazabilir.
- ✓ Ekran görüntüsü alınabilir.
- ✓ Kaynak koda gerek yoktur.

- ✓ Android platformunda bir değişiklik yapmadan robotium bu platformda kullanılabilir.
- ✓ Uygulamayı test etmek için minimal bilgi gerektirir bu da zaman kazandırır.
- ✓ Test caseleri hızlı bir şekilde yürütülür.
- ✓ Uygulama olmaksızın kaynak kodu ile çalışabilir.
- ✓ Uygulama projesi ve test projesi aynı JVM üzerinde yani Dalvik Virtual Machine (DVM) de çalışır.
- ✓ Kod kapsama raporu alınabilir.
- ✓ Komut satırı ile çalışılabilmektedir.
- ✓ Robotium Cobertura ve Emma gibi kod kapsama ölçüm araçlarıyla bir arada çalışabilir.
- ✓ Robotium kullanılarak yazılmış test caseleri Android emülatör ya da gerçek bir Android cihazda çalıştırılabilir.
- ✓ Solo kütüphanesini kullanır.

Örnek olarak aşağıdaki kod parçacığında solo.enterText metodu ile test aracı 20 değerini uygulamaya girmektedir. Daha sonra 30 değerini girmektedir. solo.clickOnButton metodu ile Çarpma isimli butona basılmaktadır. assertTrue metodu ile de çarpma işlemi sonucu için ekranda yazılması gereken 600 değerini aramaktadır. Test aracı ile bu kodu çalıştırıldığında test kodu veri girme işlemlerini kullanıcı yerine girmektedir.

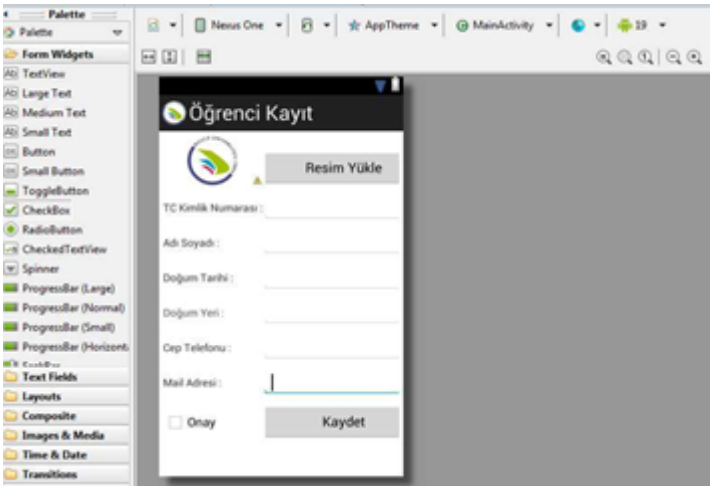
```
public void test () {

//20 değerinin girilmesi

solo.enterText(0, "20");

//30 değerinin girilmesi

solo.enterText(1, "30");

//Çarp butonuna tıklanması

solo.clickOnButton("ÇARP");
```

```
//Çarpma sonucunun doğrulanması için 600 değerinin aranması

assertTrue(solo.searchText("600"));

}
```

Robotium test aracı kullanılarak geliştirilen kod incelendiğinde birçok işlemi kolaylaştırdığı görülmüştür. Uygulama kodu ile test kodu karşılaştırıldığında ise test kodunun uygulama koduna göre oldukça kısa ve kolay kodlanabilir olduğu görülmektedir.

3.5. ROBOTİUM OTOMASYON ARACI İLE GELİŞTİRİLEN UYGULAMANIN TEST EDİLMESİ

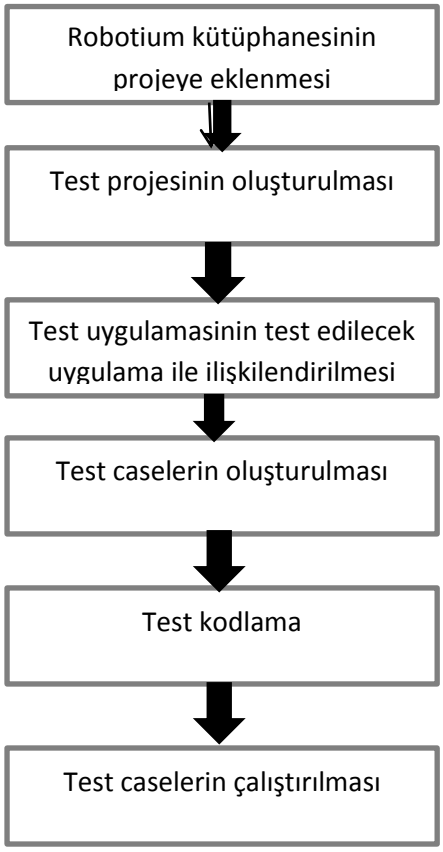
Bu tez çalışmasında testleri gerçekleştirmek amacıyla "Öğrenci Kayıt" isimli bir Android uygulaması geliştirilmiştir. Geliştirilen bu uygulama öğrenci kaydı esnasında gerekli bilgilerin girişi için bir arayüz sağlamaktadır. Öğrenci kayıt uygulaması: Resim yükleme, Tc Kimlik numarası, ad soyadı gibi çeşitli kişisel bilgilerin veri tabanına kaydedilmesi için geliştirilmiş bir uygulamadır. Uygulama geliştirilirken testin ne kadar etkili olduğunu daha iyi anlayabilmek için uygulama kodlandıktan sonra düzeltmeler yapılmadan teste geçilmiş, yazılım hataları test edilerek ortaya çıkarılmıştır. Şekil 3.2 "Öğrenci Kayıt" uygulamasının ekran tasarımını göstermektedir.



Şekil 3.2. Öğrenci Kayıt Uygulamasının Ekran Tasarımı.

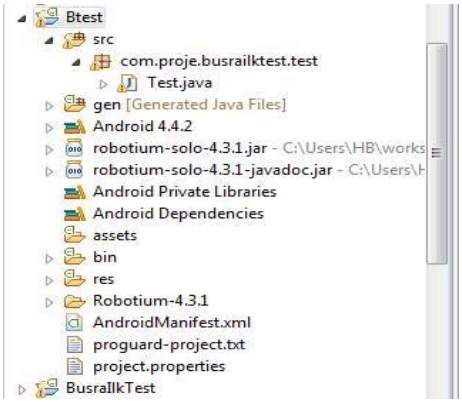
3.5.1. Robotium ile Android Test Projesi Oluşturma ve Çalıştırma

Robotium test aracı kullanarak uygulamaları test edebilmek için işlem adımları Şekil 3.3` de gösterilmiştir.



Şekil 3.3. Robotium Test Adımları.

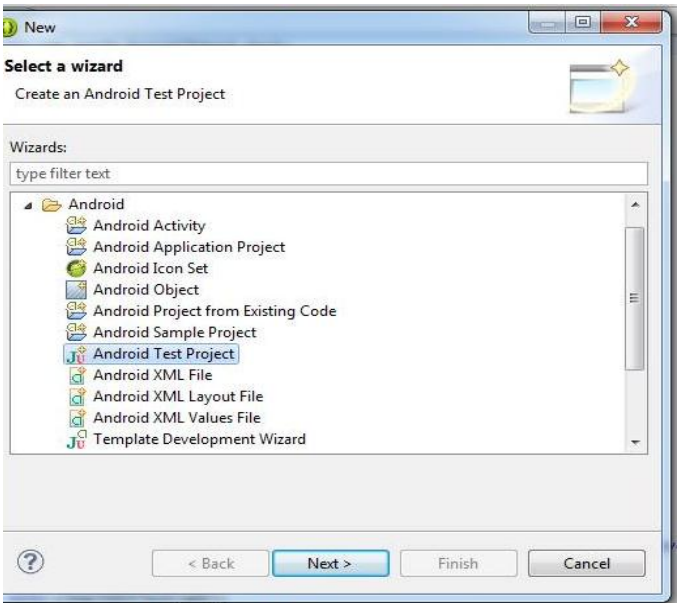
Adım 1. Robotium kütüphanesinin projeye dahil edilmesi. Şekil 3.3 Test projesine eklenmiş Robotium kütüphanesini göstermektedir.



Şekil 3.4. Robotium Kütüphanesinin Projeye Eklenmesi.

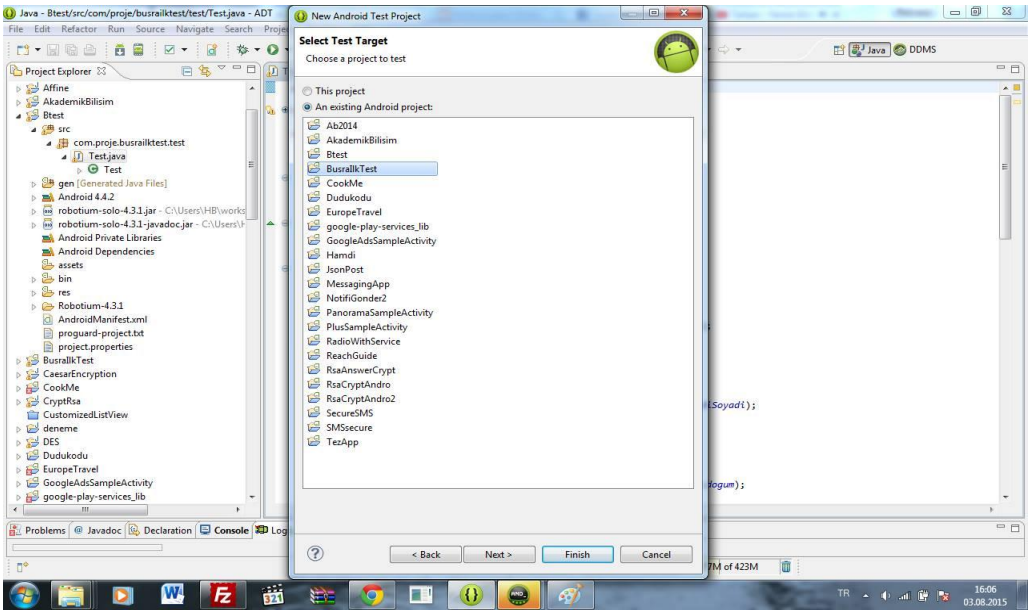
Adım 2. Test projesinin oluşturulması.

Test için yeni bir Android proje dosyası oluşturulur. Bu dosya Android test projesi olarak seçilir. Bu sayede geliştirme ekibinden bağımsız olarak testler yürütülebilir. Şekil 3.4. Test projesinin oluşturulma aşamasını göstermektedir.



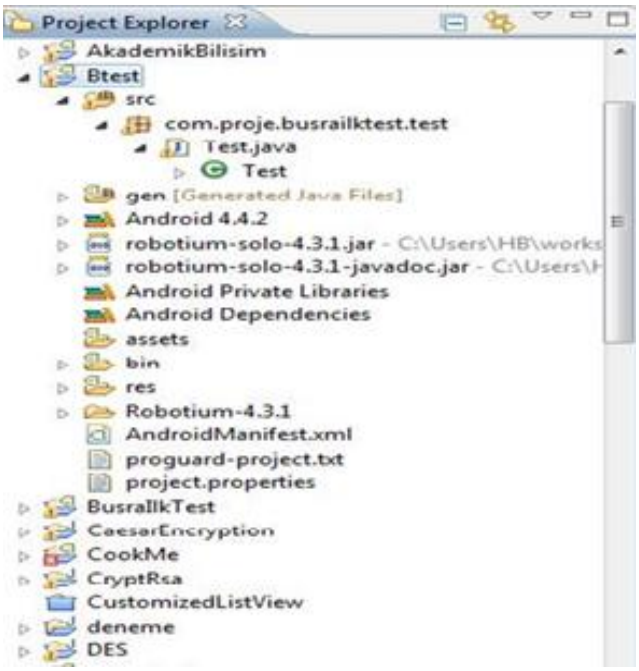
Şekil 3.5. Test Projesinin Oluşturulması.

Adım 3. Test edilecek proje dosyasının test uygulamasıyla ilişkilendirilmesi. Şekil 3.5 test edilecek proje dosyasının seçim ekranını göstermektedir.



Şekil 3.6. Test Projesinin Test Edeceği Proje Dosyasının Seçilmesi.

Adım 4. Test projesinde yeni bir Junit Test Case oluşturulması. "Btest" isimli dosya Öğrenci Kayıt uygulamasının yer aldığı "BusraIlkTest" isimli uygulamayı test etmek için kodların yazılacağı proje dosyasıdır.



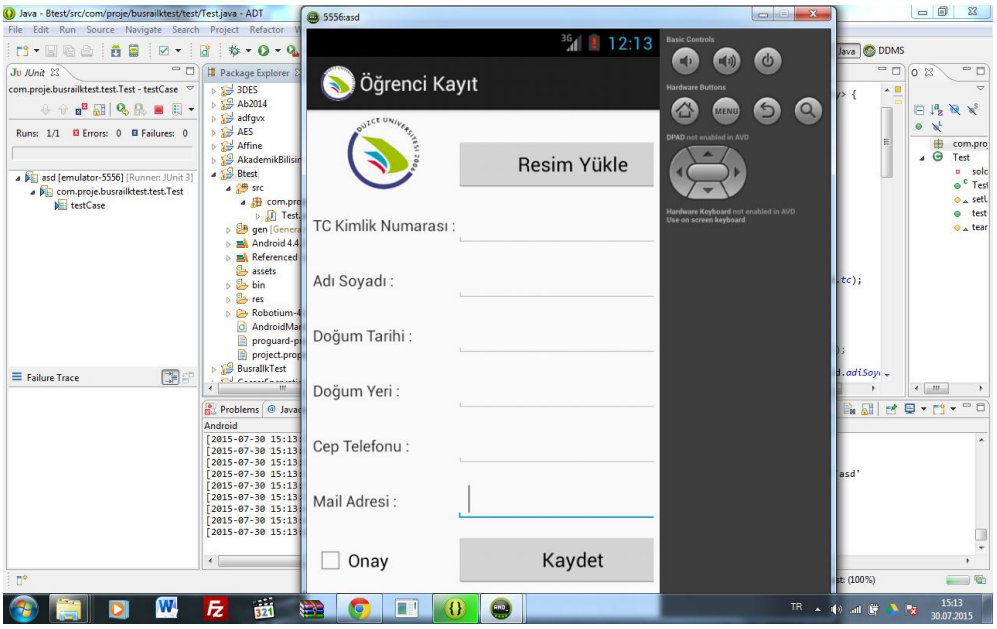
Şekil 3.7. Test Projesinin Görünümü.

Adım 5. Test kodlarının yazılması.

Robotium test otomasyonu solo nesnesinin özellikleri kullanılır ve kodlamada bu kütüphaneden yararlanılarak test kodları oluşturulur.

Adım 6. Testin çalıştırılması

Geliştirilen kodlar Android JUnit Test olarak çalıştırılarak test işlemi yürütülmeye başlanır. Şekil 3.3 testin çalışması esnasında alınmış bir ekran görüntüsünü göstermektedir.



Şekil 3.8. Testin Çalışma Esnasında Alınan Ekran Görüntüsü.

Testin Görevi: Öğrenci Kayıt sayfasına girilen verilerin ve uygulamanın çalışabilirliğinin kontrolü

Adım 1: "Resim Yükle" butonuna basıldığında resim yükleme işleminin gerçekleşip gerçekleşmediğinin test edilmesi.

Açıklama: Öğrenci Kayıt ekranında "Resim Yükle" butonunu kayıt olacak öğrenciye kendisine ait resmi yükleyebilmesi için vardır.

Beklenen Sonuç: "Resim Yükle" butonuna tıklandığında kayıt olacak kişiye ait resmin seçilerek yükleme işleminin gerçekleşmesi.

Test Sonucu: "Resim yükle" butonu doğru şekilde çalışıp, işlevini yerine getirerek resim yükleme işlemi tamamlanmıştır.

Adım 2: **"**Tc Kimlik Numarası" alanına kullanıcının girebileceği değerlerin test edilmesi. **Açıklama: "**Tc Kimlik Numarası" alanı öğrenci kimlik numarasının sisteme kaydedilmesi için ayrılmış bir alandır**.**

Beklenen Sonuç: **"**Tc Kimlik Numarası" için ayrılan alana kullanıcı tarafından yalnızca sayısal değerler girilebilmelidir ve kimlik numarası girişi için 11 haneden fazla sayıda girişe izin verilmemesi gerekmektedir.

Test Sonucu: Yalnızca sayı girişi olması gereken alana kullanıcı sayı dışında değerler girebilmiştir ve girilen karakterlerin uzunluğu kontrol edilmemektedir.

Hata 1: **”**Tc Kimlik Numarası” alanındaki kontrolsüz girişler meydana gelmektedir.

Adım 3: “Adı Soyadı” alanına yapılabilecek girişlerin test edilmesi.

Açıklama: “Adı Soyadı” alanı öğrencinin ad ve soyadı bilgisinin sisteme kaydedilmesi için ayrılmış bir alandır.

Beklenen Sonuç: Bu alanda kullanıcı isim ve soy isim olmak üzere iki ayrı metin girişi yapmalı ve bu değerlerin karakter olup olmadığı kontrol edilmelidir.

Test Sonucu: Bu alana sayısal değerler girilebilmektedir ve iki ayrı metinin varlığı kontrol edilmemektedir.

Hata 2: “Adı Soyadı” alanında kontrolsüz girişler meydana gelmektedir.

Adım 4: “Doğum Tarihi” alanına yapılabilecek girişlerin test edilmesi.

Açıklama: “Doğum Tarihi” alanı öğrencinin doğum tarihi bilgisinin sisteme kaydedilmesi için ayrılmış bir alandır.

Beklenen Sonuç: Bu alana girilecek veriler sayısal değer olmalı ve ay gün yıl olarak birbirinden ayrılmış bir şekilde veri girişi yapılabiliyor olmalıdır.

Test Sonucu: Doğum tarihi için ayrılmış alana doğum yeri bilgisi girilmiştir ve sistem herhangi bir uyarı vermemiştir.

Hata 3: “Doğum Tarihi” alanında kontrolsüz girişler meydana gelmektedir.

Adım 5: “Cep Telefonu” alanına yapılabilecek girişlerin test edilmesi.

Açıklama: “Cep Telefonu” alanı öğrencinin cep telefonu bilgisinin sisteme kaydedilmesi için ayrılmış bir alandır.

Beklenen Sonuç: Bu alana girilecek veriler sayısal değer olmalı ve veri girişi 11 hane ile sınırlandırılmış olmalıdır.

Test Sonucu: Bu alana string veri girişi yapılarak Erzurum değeri girilmiştir ve bu veri girişi sistem tarafından kabul edilmiştir. Daha sonra girilen değer silinerek cep telefonu numarası girilmiştir.

Hata 4: "Cep Telefonu" alanında kontrolsüz girişler meydana gelmektedir.

Adım 6: "Mail Adresi" alanına yapılabilecek girişlerin test edilmesi.

Açıklama: "Mail Adresi" alanı öğrencinin mail adresi bilgisinin sisteme kaydedilmesi için ayrılmış bir alandır.

Beklenen Sonuç: Mail adresi için ayrılmış alana girilecek verilerin .com, .edu vb. uzantılardan birini içerip içermediği kontrol edilmeli ve bu alanda "@" simgesi ile ayrılmış stringlerin kontrolü yapılmalıdır.

Test Sonucu: Bu alana mail adresi bilgisi içermeyen girişler yapılmıştır ve sistemin bir uyarı vermediği gözlemlenmiştir. Daha sonra girilen bu değerler değiştirilerek doğru veri girişi yapılmıştır.

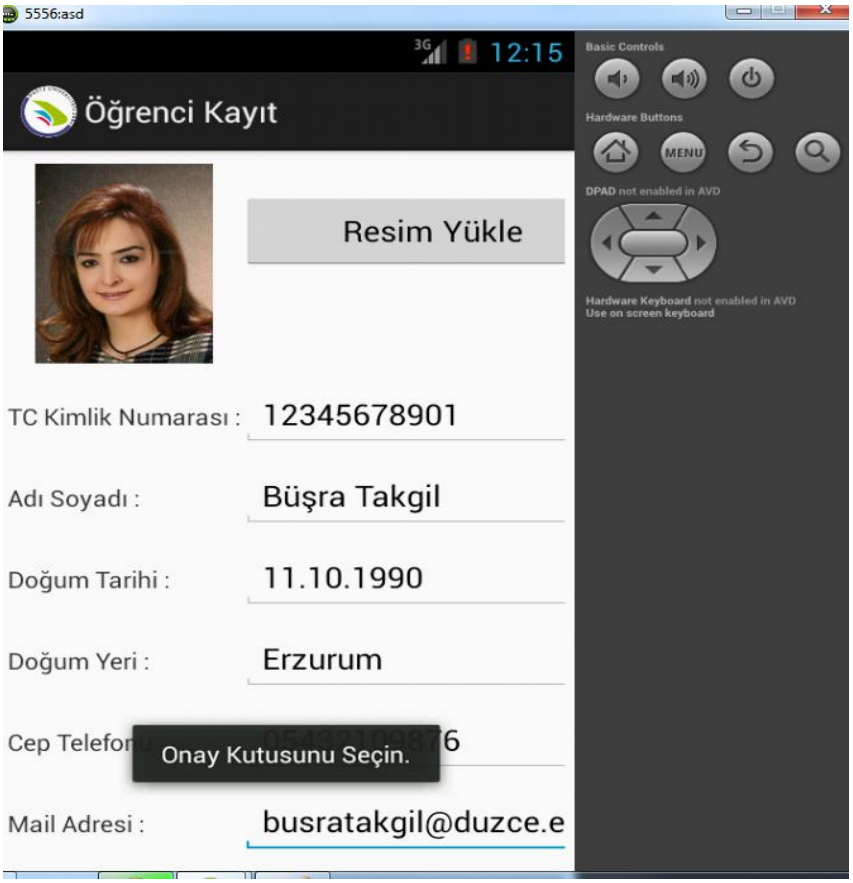
Hata 5: "Mail Adresi" alanında kontrolsüz girişler meydana gelmektedir.

Adım 7: "Onay" kutusunun işlevinin test edilmesi.

Açıklama: "Onay" kutusu bilgi girişi tamamlandıktan sonra girilen verilerin kontrol edilmesi amacıyla eklenen bir checkboxdır.

Beklenen Sonuç: "Onay" kutusunun seçili olmadığı durumda "Kaydet" butonunun işlevini yerine getirmemesi ve onay kutusunun seçili olması gerektiği hakkında uyarı verilmesi gerekmektedir.

Test Sonucu: Onay kutusu seçili olmadan "Kaydet" butonuna basıldığında sistem uyarı vermiştir daha sonra onay kutusu seçilerek işlem doğru şekilde tamamlanmıştır.



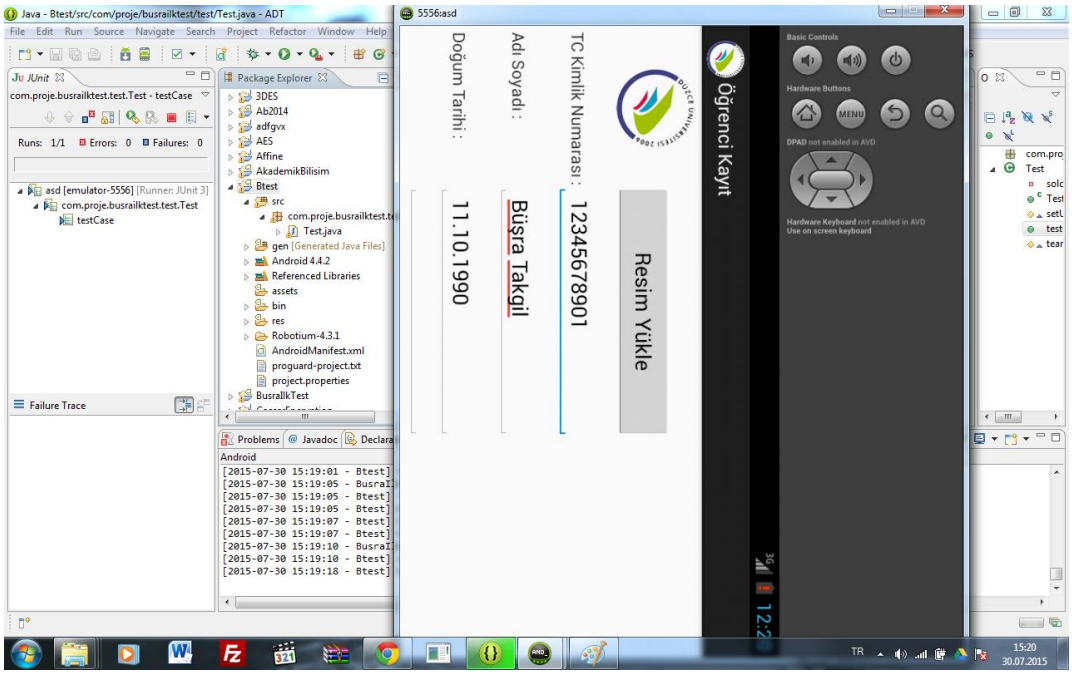
Şekil 3.9. Adım 7 Test Ekran Görüntüsü

Adım 8: Uygulama arayüzünün ekran döndürmelerine karşı test edilmesi.

Açıklama: Kullanılan telefona, ekran büyüklüğüne bağlı olarak uygulama ekranının daha iyi görünmesi ya da kolay kullanım açısından kullanıcı ekran döndürme işlemi yapabilir.

Beklenen Sonuç: Kullanıcı ekran döndürme işlemini gerçekleştirdiğinde uygulama arayüzü bunu destekleyecek şekilde tasarlanmış olmalı, yazı kaymaları ve çözünürlükle ilgili tedbirler alınmış olmalıdır.

Test Sonucu: Ekran döndürülüp daha sonra eski konumuna getirilerek uygulamanın döndürme için uygun bir tasarıma sahip olduğu görülmüştür.



Şekil 3.10. Adım 8 Test Ekran Görüntüsü.

Adım 9: Uygulama çalışırken ekran görüntüsü alınıp alınamayacağının test edilmesi.

Açıklama: Kullanıcı uygulamayı kullanırken girdiğin verileri daha sonra kontrol etmek amacıyla ya da herhangi bir sebeple ekran görüntüsü almak isteyebilir.

Beklenen Sonuç: Uygulama kullanılırken ekran görüntüsü alınıp kaydedilebilir olmalıdır.

Test Sonucu: Ekran görüntüsü başarılı bir şekilde alınıp, kaydedilmiştir.

3.5.2. Test Değerlendirmesi

Robotium test aracı kullanılarak uygulamanın test edilmesi sonucunda çeşitli hatalar tespit edilmiştir. Genel bir değerlendirme yapıldığında uygulama kullanıcı kullanımı açısından yetersiz görülmüştür ve yanlış veri girilmesine olanak sağlamaktadır. Bu hatalar genellikle yazılımsal hatalar olup veri girişlerinin kontrolünün iyi bir şekilde yapılmamasından kaynaklanmaktadır. Bu problemleri gidermek amacıyla veri girişi yapılan bazı alanlarda uyarıcı bilgilendirmeler eklenebilir ya da veri girişi için seçilebilir menüler oluşturulabilir. Örneğin doğum yeri bilgisi için il listesi açılır pencere şeklinde kullanıcının seçimine sunulabilir. Böylece doğum yeri bilgisine il dışında veri girişi engellenmiş olabilir. Bir diğer çözüm önerisi ise kullanıcının veri

girişi yapacağı alanlarda bilgilendirici mesajlar kullanılabilir örneğin bu alana sadece sayısal değerler girilmelidir.

Uygulamalar kullanıcılar için geliştirilir ve uygulamanın kullanım kolaylığı, hızı, görsel tasarımı önem taşır. Bu sebeple uygulamalar bu durumlar göz önüne alınarak geliştirilmelidir.

Yapılan testler sonucunda hataların ortaya çıkarılmasıyla, ürün kullanıcıya sunulmadan karşılaşılabilecek problemler tespit edilmiş ve uygulamalar için testin önemi ve gerekliliği ortaya konmuştur.

4. SONUÇLAR VE ÖNERİLER

Mobil uygulamalara gösterilen talep arttıkça, mobil uygulama yazılımları giderek karmaşık bir hale gelmiştir. Geliştirilen uygulamaların büyüme ve kalite isteğini karşılayabilmek için yazılım geliştirme süreci büyük önem taşımaktadır. Bu kaliteyi sağlamanın en iyi yolu ise uygulamaları test etmektir.

Mobil uygulamaların testi geleneksel uygulama testleriyle benzer özelliklere sahip olsa da mobil uygulamalar için bazı ek gereksinimlere ihtiyaç duyulur. Mobil uygulamalar test edilirken bazı zorluklarla karşılaşılır.

Mobil uygulamalar test edilirken karşılaşılan zorluklara şunlar dâhil edilebilir: Mobil cihazların yazılımsal ve donanımsal olarak farklı özelliklerle donatılması, mobil cihaz sınırlandırmaları, donanım platformundaki kaynak kısıtlılığı, mobil uygulamaların bağlantı değişkenliği, mobil platformun açık kaynak kodlu yapısı, diğer uygulamalarla etkileşim.

Android uygulamalarının testi ve gelişimi de taşınabilir cihazlar üzerindeki kısıtlamalardan etkilenir. Android testleri mobil test zorluklarına ek olarak kendine özgü zorluklar içerir. Bu faktörler Android mobil uygulamaların test süreci ve kalite güvencesi için yeni zorluklar ortaya çıkarır. Test zorlukları ve karmaşıklığıyla başa çıkabilmek için test otomasyon araçları etkili bir çözüm yöntemi olarak ortaya çıkar. Test otomasyonu testlerin yürütülmesini kontrol eder ve beklenen sonuçla gerçek sonuçları yazılım kullanılarak karşılaştırmaya yardımcı olur. Harcanan zamanı azaltarak verimliliği arttırır. Otomasyon araçları kullanılarak hata yapma oranı azaltılabilir. Böylece mobil uygulamaların test zorluklarını otomasyon çözümüyle bakış açısı sağlanmıştır.

Bu tez çalışmasında Robotium test aracı kullanılarak tez için geliştirilmiş bir Android uygulama test edilmiş, test sonucunda çeşitli hatalar tespit edilmiştir. Mobil uygulama test edilirken mobil otomasyon test aracı olan Robotium` un kullanılması test işlemlerini kolaylaştırmış ve zaman kazandırmıştır.

Uygulamalar kullanıcılar için geliştirildiği için uygulamanın kullanım kolaylığı, hızı, görsel tasarımı büyük önem taşır. Bu sebeple uygulamalar bu durumlar göz önüne alınarak geliştirilmesi gerekmektedir. Uygulama değerlendirildiğinde kullanıcı kullanımı açısından yetersiz görülmüştür ve yanlış veri girilmesine olanak sağladığı tespit edilmiştir. Test aracılığıyla ortaya çıkarılan hatalar genellikle yazılımsal hatalar olup veri girişlerinin kontrolünün iyi bir şekilde yapılmamasından kaynaklandığı gözlemlenmiştir. Bu problemleri gidermek amacıyla çözüm önerileri sunulmuştur. Yapılan testler sonucunda hataların ortaya çıkarılmasıyla, ürün kullanıcıya sunulmadan karşılaşılabilecek problemler tespit edilmiş ve uygulamalar için testin önemi ve gerekliliği ortaya konmuştur.

5. KAYNAKLAR

[1] KNYCH, Thomas W.; BALIGA, Ashwin. Android application development and testability. In: *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*. ACM, (**2014)** p. 37-40.

[2] DANTAS, Valéria Lelli Leitão, et al. Testing requirements for mobile applications. In: *Computer and Information Sciences, 2009. ISCIS 2009. 24th International Symposium on*. IEEE, (**2009)** p. 555-560.

[3] FRANKE, Dominik; WEISE, Carsten. Providing a software quality framework for testing of mobile applications. In: *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*. IEEE, (**2011)** p. 431-434.

[4] FRANKE, Dominik; KOWALEWSKI, Stefan; WEISE, Carsten. A mobile software quality model. In: *Quality Software (QSIC), 2012 12th International Conference on*. IEEE, **(2012)** p. 154-157.

[5] KIRUBAKARAN, B.; KARTHIKEYANI, V. Mobile application testing—Challenges and solution approach through automation. In: *Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on*. IEEE, **(2013)** p. 79-84.

[6] KAASILA, Jouko, et al. Testdroid: automated remote UI testing on Android. In:*Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*. ACM, **(2012)** p. 28.

[7] SATOH, Ichiro. A testing framework for mobile computing software. *Software Engineering, IEEE Transactions on*, **(2003).**

[8] LIU, Yanmei; LU, Yueming; LI, Yonghua. An Android-based approach for automatic unit test. In: Cyberspace Technology (CCT 2014), *International Conference on. IET*, **(2014)**p. 1-4.

[9] SHIN, Won; KIM, Tae-Wan; CHANG, Chun-Hyon. Experiment study of Android Software Test using Moment Invariants Algorithm.

[10] GEOGY, Mrs Manju; DHARANI, Andhe. Customising Android Automated Testing Framework To Enable Native Hardware And Software Support. In:*International Journal of Engineering Research and Technology*. ESRSA Publications, **(2013).**

[11] MIRZAEI, Nariman, et al. Testing android apps through symbolic execution.*ACM SIGSOFT Software Engineering Notes*, **(2012).**

[12] GUO, Chenkai, et al. An automated testing approach for inter-application security in Android. In: *Proceedings of the 9th International Workshop on Automation of Software Test*. ACM, **(2014)** p. 8-14.

[13] DELAMARO, Márcio Eduardo; VINCENZI, Auri Marcelo Rizzo; MALDONADO, José Carlos. A strategy to perform coverage testing of mobile applications. In:*Proceedings of the 2006 international workshop on Automation of software test*. ACM, **(2006)** p. 118-124.

[14] FETAJI, Majlinda; DIKA, Zamir. Usability testing and evaluation of a mobile software solution: a case study. In: *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*. IEEE, **(2008)** p. 501-506.

[15] KOVACEVIC, Marko, et al. System for automatic testing of Android based digital TV receivers.

[16] SHAHRIAR, Hossain; NORTH, Steve; MAWANGI, Edward. Testing of Memory Leak in Android Applications. In: High-Assurance Systems Engineering (HASE), 2014 IEEE 15th International Symposium on. IEEE, **(2014)** p. 176-183.

[17] P. Naur ve B. Randell, (Editörler), "Software Engineering", Report on a conference as sponsored by NATO SCIENCE COMMITTEE, Germany, **(1968).**

[18] Gürbüz, A., "Yazılım Test Mühendisliği", Papatya Yayıncılık Eğitim, İstanbul, **(2010).**

[19] Güler, Z., Baykara, M. ve Türkoğlu, İ., "Teknoloji Mühendisliği: Yazılım Mühendisliği Eğitimi", *Elektrik-Elektronik Bilgisayar Sempozyumu* (FEEB 2011), 186-192, Elazığ,**(2011).**

[20] http://ezinearticles.com/?Why-Do-We-Need-SoftwareEngineering?&id=402532 , (Erişim Tarihi: 30 Haziran **2015**).

[21] B. Douglas, "Software Engineering for Students A programming Approach", fourth edition, AddisonWeslwy, ISBN: 0321261275, **(2005).**

[22] http://tr.wikipedia.org/wiki/Yazılım (Erişim tarihi: 11 Mart **2014**)

[23] Kartın, Esma, "Güvenli Yazılım Geliştirme", Beykent Üniversitesi Computer Based Business Application CEN 404, *Seminer Notları*, İstanbul, **(2008).**

[24] http://en.wikipedia.org/wiki/Waterfall_model (Erişim tarihi: 15 Aralık **2014**).

[25] https://tr.wikipedia.org/wiki/Waterfall_model#/media/File:Waterfall.png (Erişim tarihi: 24 Aralık **2014**)

[26] http://salyangoz.com.tr/yazilim-gelistirme-modenelerdir (Erişim tarihi: 24 Ocak **2015**)

[27] Kuday G., Yazılım Mühendisliği Yöntemleriyle Yazılım Test Süreci, *Yüksek Lisans Tezi*, Haliç Üniversitesi, **(2015).**

[28] Myers G. J., "The Art of Software Testing". John Wiley & Sons, Inc., New York, NY, USA, **(1979).**

[29] Hetzel, W. C., "The Complete Guide to Software Testing", 2nd ed. Wellesley, Mass. : QED Information Sciences, ISBN: 0894352423, **(1988).**

[30] Sykes M.G., "A Practical Guide to Testing Object-Oriented Software", Addison-Wesley, March 15, **(2001).**

[31] MULLER, T., GRAHAM, D., FRIEDENBERG, D. ve VEENDENDAL, E., International Software Testing Qualifications Board (ISTQB), Foundation Level Syllabus, USA, **(2007).**

[32] TİFTİK, N., ÖZTARAK, H., ERCEK, G. ve ÖZGÜN, S., Sistem/yazılım geliştirme sürecinde doğrulama faaliyetleri, *III.Ulusal Yazılım Mühendisliği Sempozyumu*, Ankara, 1-2, **(2007).**

[33] http://www.onestoptesting.com , (Erişim tarihi: 5 Mayıs **2015**).

[34] MUSTAFA, K. ve KHAN, R.A., Software Testing: Concept and Practices, India, Lucknow, 5-21, 227-228, **(2007).**

[35] Mosley, D.J. ve Posey B.A., "Just Enough Software Test Automation", Prentice Hall PTR, USA, **(2002).**

[36] Myers, Glenford J., Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, **(2011).**

[37] http://www.softwaretestinghelp.com/ (Erişim tarihi: 4 Haziran **2014**).

[38] http://msdn.microsoft.com (Erişim tarihi:7 Temmuz **2015**).

[39] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research," Hanover, NH, USA, Tech. Rep., **(2000).**

[40] Ballard, B. , Designing the Mobile User Experience. Little Springs Design, Inc., USA: Wiley, **(2007).**

[41] KIRUBAKARAN, B.; KARTHIKEYANI, V. Mobile application testing—Challenges and solution approach through automation. In: *Pattern Recognition, Informatics and Mobile Engineering (PRIME), 2013 International Conference on*. IEEE, **(2013)** p. 79-84.

[42] http://developer.android.com/tools/testing/testing_android.html (Erişim tarihi: 3 Haziran **2015**).

6.EKLER

EK-1. TEST KODLARI

```
package com.proje.busrailktest.test;


import com.jayway.android.robotium.solo.Solo;

import com.proje.busrailktest.MainActivity;

import android.test.ActivityInstrumentationTestCase2;

import android.widget.Button;

import android.widget.EditText;

import android.widget.TextView;

import junit.framework.TestCase;


public class Test extends ActivityInstrumentationTestCase2<MainActivity> {


	private Solo solo;

	public Test() {

		super(com.proje.busrailktest.MainActivity.class);

	}


	protected void setUp() throws Exception {

		solo = new Solo(getInstrumentation(), getActivity());
```

```
}

public void testCase() throws Exception {

solo.clickOnButton("Resim Yükle");
 EditText tc = (EditText) solo.getView(com.proje.busrailktest.R.id.tc);
solo.clearEditText(tc);
solo.enterText(tc,"Büşra Takgil");
solo.clearEditText(tc);
solo.enterText(tc,"12345678901");

EditText adi = (EditText) solo.getView(com.proje.busrailktest.R.id.adiSoyadi);
solo.clearEditText(adi);
solo.enterText(adi,"12345678901");
solo.clearEditText(adi);
solo.enterText(adi,"Büşra Takgil");
solo.waitForText("Büşra Takgil", 1, 2000);

EditText dogum = (EditText) solo.getView(com.proje.busrailktest.R.id.dogum);
solo.clearEditText(dogum);
solo.enterText(dogum,"Erzurum");
```

```
solo.waitForText("Erzurum", 1, 2000);

solo.clearEditText(dogum);

solo.enterText(dogum,"11.10.1990");


 EditText adres = (EditText) solo.getView(com.proje.busrailktest.R.id.adres);

solo.clearEditText(adres);

solo.enterText(adres,"05432109876");

solo.waitForText("05432109876", 1, 2000);

solo.clearEditText(adres);

solo.enterText(adres,"Erzurum");


EditText cep = (EditText) solo.getView(com.proje.busrailktest.R.id.cepTel);

solo.clearEditText(cep);

solo.enterText(cep,"Erzurum");

solo.waitForText("Erzurum", 1, 2000);

solo.clearEditText(cep);

solo.enterText(cep,"05432109876");


 EditText mail = (EditText) solo.getView(com.proje.busrailktest.R.id.mail);

solo.enterText(mail,"05432109876");

solo.waitForText("05432109876", 1, 2000);
```

```
solo.clearEditText(mail);

solo.enterText(mail,"busratakgil@duzce.edu.tr");


solo.clickOnButton("Kaydet");

 solo.waitForActivity("com.proje.busrailktest.MainActivity",10000);

              solo.takeScreenshot();

          solo.setActivityOrientation(Solo.LANDSCAPE);

          solo.wait(3000);

            solo.setActivityOrientation(Solo.PORTRAIT);

          solo.wait(3000);

  }
```

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı : TAKGİL Büşra

Uyruğu : T.C.

Doğum tarihi ve yeri : 02.01.1990 / Erzurum

Telefon : 0 (380) 542 10 36 / 4732

Faks : 0 (380) 542 10 37

E-posta : busratakgil@duzce.edu.tr

Eğitim

Derece	**Eğitim Birimi**	**Mezuniyet tarihi**
Lisans	Selçuk Üniversitesi / Bilgisayar Müh.	2012
Lise	İbrahim Hakkı Fen Lisesi	2007

İş Deneyimi

Yıl	**Yer**	**Görev**
20014-(devam ediyor)	Düzce Üniversitesi	Araştırma Görevlisi

Yabancı Dil

İngilizce (YDS: 71.25)

Yayınlar

1. Takgil B., Kara R., Android Mobil Uygulamalar İçin Yazılım Testi, *UMAS 2015,* **(2015)**.