T.C.

EGE ÜNİVERSİTESİ

ANDROİD GRAFİKSEL KULLANICI ARAYÜZLERİ İÇİN MODEL TABANLI TEST SENARYOSU ÜRETİMİ

Yüksek Lisans Tezi

Gizem MERCAN

Uluslararası Bilgisayar Anabilim Dalı

İzmir

2019

T.C.
EGE ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

ANDROİD GRAFİKSEL KULLANICI ARAYÜZLERİ İÇİN MODEL TABANLI TEST SENARYOSU ÜRETİMİ

Gizem MERCAN

Danışman : Dr. Öğr. Üyesi Moharram CHALLENGER

Uluslararası Bilgisayar Anabilim Dalı
Bilgi Teknolojileri Yüksek Lisans Programı

İzmir
2019

Gizem Mercan tarafından yüksek lisans tezi olarak sunulan "Android Grafiksel Kullanıcı Arayüzleri için Model Tabanlı Test Senaryosu Üretimi" başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 08/08/2019 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	: Dr.Öğr.Üyesi Muhammed CHALLENGER	
Raportör Üye	: Dr.Öğr.Üyesi İlker Kocabaş	
Üye	: Doç.Dr. Tolga AYAV	

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi olarak sunduğum "Android Grafiksel Kullanıcı Arayüzleri için Model Tabanlı Test Senaryosu Üretimi" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

08/08/ 2019

Gizem MERCAN

ÖZET

ANDROİD GRAFİKSEL KULLANICI (GKA) ARAYÜZLERİ İÇİN MODEL TABANLI TEST SENARYOSU ÜRETİMİ

MERCAN, Gizem

Yüksek Lisans Tezi, Uluslararası Bilgisayar Anabilim Dalı
Tez Danışmanı: Dr. Öğr. Uyesi Moharram CHALLENGER
Ağustos 2019, 80 sayfa

Bu tez çalışmasının amacı bir Android uygulamasının grafiksel kullanıcı arayüzü (GKA) (ing. Graphical User Interface- GUI) testinin farklı model tabanlı yaklaşımlarla gerçekleştirilmesi ve bu testler sırasında kullanılacak olan test senaryolarının üretim yöntemlerinin hata kapsama kriterleri ve test koşum süreleri açısından karşılaştırılmasıdır.

Tez çalışması kapsamında kullanılacak olan model tabanlı test senaryosu üretim yöntemleri ayrıca bütünsel (holistic) ve mutasyon (mutation) test yöntemleri kullanılarak Android uygulamasının GKA testi için yeni bir çerçeve ile genişletilecektir. Böylece elde edilen yaklaşım yazılım testinde iyi bilinen ideal test kriterleri açısından değerlendirilmesi sağlanacaktır.

Ayrıca kullanılan tüm modeller ve yaklaşımlar popüler bir mesajlaşma uygulaması olan Telegram üzerinde test edilecektir. Elde edilen sonuçlara bu tez çalışması kapsamında yer verilecektir.

Anahtar sözcükler: Ideal Test, Android Uygulaması, Test Dizisi Üretimi, Mutasyon Testi, Sonlu Durum Makinası, Düzenli İfade.

ABSTRACT

MODEL BASED TEST SEQUENCE GENERATION FOR ANDROID GRAPHICAL USER INTERFACES (GUI)

MERCAN, Gizem

MSc in International Computer.
Supervisor: Asst. Prof. Dr. Moharram CHALLENGER
August 2019, 80 pages

The purpose of this thesis is to compare the graphical user interface (GUI) of an Android application with different model based approaches and to compare the test methods of the test scenarios to be used during these tests in terms of error coverage criteria and test run times.

The model-based test scenario production methods to be used within the scope of the thesis study will also be expanded with a new framework for the GKA test of the Android application using holistic and mutation test methods. The approach thus obtained will be evaluated in terms of the ideal test criteria well known in the software testing.

Also all the models and approaches used will be tested on Telegram, which is a popular messaging application. The results will be covered by this thesis.

Keywords: Ideal Testing, Model Based Testing, Mobile GUI Testing, Android Ap-plication Testing, Test Sequence Generation, Mutation Testing, Finite State Machine, Regular Expression.

ÖNSÖZ

İnternet ve kablosuz iletişim teknolojilerinin ilerlemesiyle birlikte mobil uygulamalar hayatımızın her alanında yer alır hale gelmiştir. Mobil uygulama sayısının gün geçtikçe artması bu alandaki rekabetin şiddetlenmesine neden olmaktadır. Bu rekabet ortamında kendine yer bulmak isteyen üreticiler kısa süre içerisinde kaliteli ve olabildiğince hatasız uygulamalar üretilmesini hedeflemektedir. Ortaya çıkan uygulamaların kalitesini belirleyen en önemli kriter iyi birer test sürecinden geçmiş olmasıdır. En uygun test senaryorlarının hedeflenmesi ve bu senaryoların başarılı bir şekilde yürütülmesi test etme sürecini olumlu bir şekilde etkilemektedir. Bu noktada donanım programları için sistemin modellenerek uygun test senaryolarının üretilme konusundaki çalışmalar tez konumu belirlememde etken oldu. Tez çalışmamda ise donanım programları için genişletilen ideal test yönteminin, model tabanlı olarak Android uygulaması arayüz testleri için gerçekleştirdim.

İZMİR

08/08/2019

Gizem Mercan

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

ŞEKİLLER DİZİNİ (devam)

Şekil Sayfa

ŞEKİLLER DİZİNİ (devam)

ÇİZELGELER DİZİNİ

Çizelge Sayfa

ÇİZELGELER DİZİNİ (devam)

Çizelge Sayfa

KISALTMALAR DİZİNİ

Kısaltmalar

API	Application Programming Interface.
Dİ	Düzenli İfade.
EFG	Event Flow Graph.
ESG	Event Sequence Graph.
FSM	Finite State Machine.
GKA	Grafiksel Kullanıcı Arayüzü.
GUI	Graphical User Interface.
HFSM	Hierarchical Finite State Machine.
IDE	Domain Specific Modeling Language.
IRE	Indexed Regular Expression.
MBT	Model Based Testing.
OAÇ	Olay Akış Çizgesi.
OSÇ	Olay Sıra Çizgesi.
RE	Regular Expression.
SDM	Sonlu Durum Makinesi.
SUT	System Under Test.

1. GİRİŞ

Mobil uygulamalar sağladığı kullanım kolaylıkları ve hızlıca erişilebilirlikleri sayesinde son zamanlarda hayatımızın her alanında yer alır hale gelmiştir. Üretilen mobil uygulama sayısının giderek artması, mobil uygulama pazarındaki rekabetin gün geçtikçe şiddetlenmesi, kısa sürede uygulama geliştirilmesi ve yayınlanmasına ihtiyaç duyulmaktadır. Bu kadar hızlı bir geliştirme ortamında ortaya çıkan ürünlerin kalitesini belirleyen faktörlerin en başında ürünlerin iyi birer test sürecinden kabul edilebilir bir düzeyde geçmesi gelmektedir.

Test edilen uygulamaların çok fazla özelliğe sahip olması, çok fazla test senaryosu üretimi gerektirmektedir. Fakat her senaryonun ele alınması mümkün olmamaktadır. Dolayısıyla ürünün geneline bakıldığında en uygun test senaryolarının hedeflenmesi ve bu senaryoların başarılı bir şekilde yürütülmesi temel alınarak sistemin bütünü hakkında bir görüşe sahip olmak uygulamanın test süresini azaltarak test sürecini olumlu bir şekilde etkilemektedir.

Daha önce Android uygulamalarının model tabanlı test senaryosu oluşturma yöntemlerinin kıyaslanması sırasında ideal test yöntemine yakınlığı konusunda çalışma yapılmadığından dolayı tez çalışmasında elde edilen sonuçların hata kapsama ve test üretim süresi kriterleri bağlamında kıyaslanması oldukça önemlidir. Elde ettiğimiz modelden yola çıkarak mutant bir sistem yaratılması ve bu sistemin güncel bir Android uygulaması üzerinde test edilmesi farklılık yaratan diğer bir unsurdur. Çalışma sonucunda elde edilen çıktılar, birçok alan (domain) için geliştirilen mobil uygulamaların test aşamalarının daha kolay, daha etkin bir şekilde yapılmasını ve ortaya çıkan ürünlerin kalitesini olumlu yönde etkilemesi açısından özgün bir katkı sağlayacaktır.

Tezin 2. Bölümü'nde altyapı ve alan bilgisi verilecektir. 3. Bölümü'nde ilgili çalışmalar gözden geçirilecek ve kısaca açıklanacaktır. 4. Bölüm'de ise Android GKA testleri için kullanılan yöntemden bahsedilecektir. 5. Bölüm'de üzerinde çalıştığımız örnek durumdan bahsedilecektir. 6. Bölüm'de kullanılan

araçlar anlatılacaktır. Son olarak 7. Bölüm'de önerilen yöntemin beklenilen faydaları ve öngörülen zorlukları özetlenecektir.

2. ALTYAPI VE ALAN BİLGİSİ

Bu bölümde tez çalışması için gerekli olan altyapı bilgileri yer almaktadır. Bu bölümün alt bölümlerinde Android uygulamasının yapısından, uygulama arayüzlerinden, model tabanlı test yönteminden, mutasyon tabanlı test yönteminden ve son olarak ideal tabanlı test yönteminden bahsedilecektir.

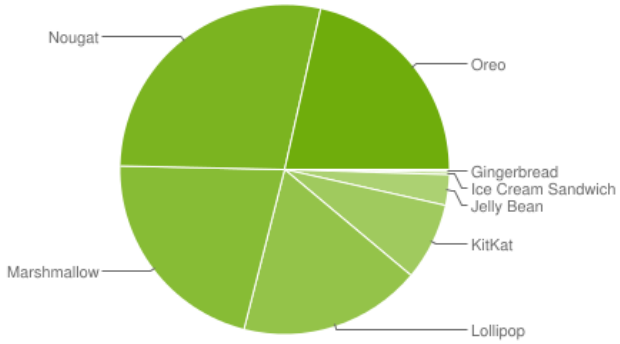
2.1 Android Uygulamalarının Yapısı

Mobil bir işletim sistemi olan Android, Open Handset Alliance (2018), Google ve özgür yazılım topluluğu tarafından geliştirilmiştir. Kullanımı herkese açık olan ücretsiz ve özgür bir sistem sunar. Linux tabanlı bir işletim sistemi içermektedir. Uygulama geliştirmek için kullanılan araçlar ve farklı uygulama programlama arayüzleri (Application Programming Interface (API)) Android SDK (Software Development Kit) ile sağlanır. Aşağıdaki yer alan Tablo 1'de Android cihaz sahiplerinin kullandığı Android platformlarının dağılımı gösterilmektedir.

Tablo 2.1. Android Platform Dağılım Tablosu (Android, 2018)

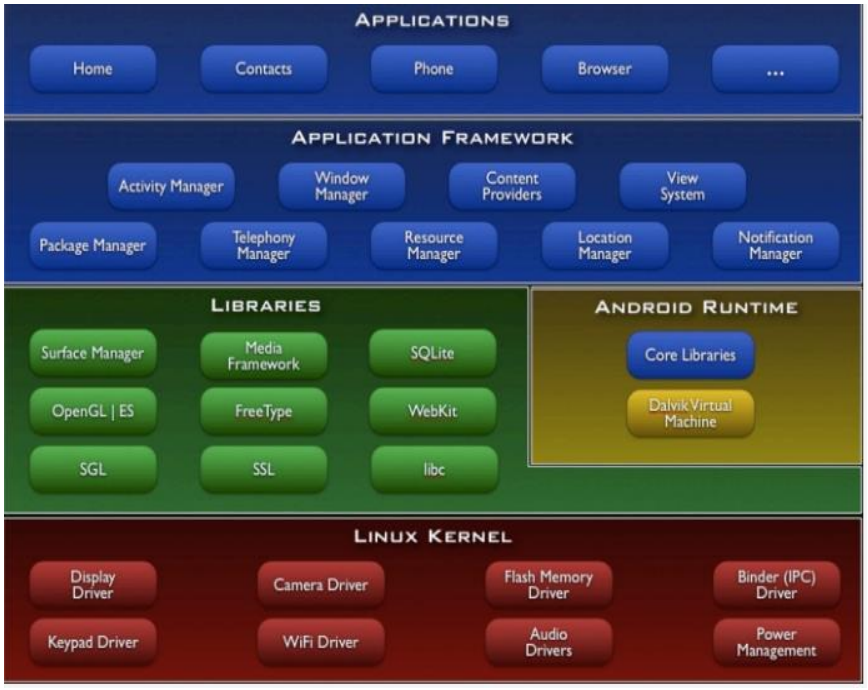
Versiyon	Kod Adı	API	Dağılım
2.3.3 - 2.3.7	Gingerbread	10	0.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.1%
4.2.x		17	1.5%
4.3		18	0.4%
4.4	KitKat	19	7.6%
5.0	Lollipop	21	3.5%
5.1		22	14.4%
6.0	Marshmallow	23	21.3%
7.0	Nougat	24	18.1%
7.1		25	10.1%
8.0	Oreo	26	14.0%
8.1		27	7.5%

2018 yılının Ekim ayının sonuçlarına göre en çok kullanılan işletim sistemleri Marshmallow ve Nougat sürümleri olmaktadır (Android, 2018). Sonuçlar 26 Ekim tarihine kadar geçen 7 günlük dönemden alınmıştır.



Şekil 2.1. Android platformunun dağılımı (Android, 2018)

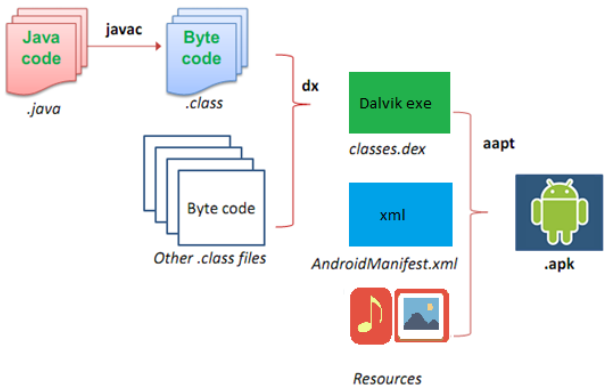
Android işletim sisteminin parçaları Şekil 2.2'de gösterildiği gibi en alt katmanında çekirdek (kernel), bir üst katmanında sistem kütüphaneleri, onun üzerinde uygulama geliştirme çatıları (frameworks) ve en üst katmanda ise yerleşik temel uygulamaları içermektedir. Bu sayede yazılımcılara daha denetimli, test edilebilir ve bakımı kolay uygulamalar geliştirmesine olanak sağlamış olur.



Şekil 2.2 Android katmanları (Smieh, 2012)

Android, temel yapısında Linux çekirdeğini barındırır. Linux çekirdeğine eklenen kodlar ve kütüphaneler Genel Kamu Lisansına sahiptir (GNU General Public License, 1991). Diğer bileşenleri ise üretici firmaların kendi kapalı ROM'larını tasarlamalarına uygun olacak şekilde Apache Yazılım Lisansı (Apache License, 2004) ile dağıtılmaktadır. Bu katmanda güç yönetimi, ses sürücüleri, wifi sürücüleri, klavye sürücüleri, ekran sürücüleri, kamera sürücüleri, flash bellek sürücüleri ve işlemler arası iletişim sürücüleri bulunmaktadır.

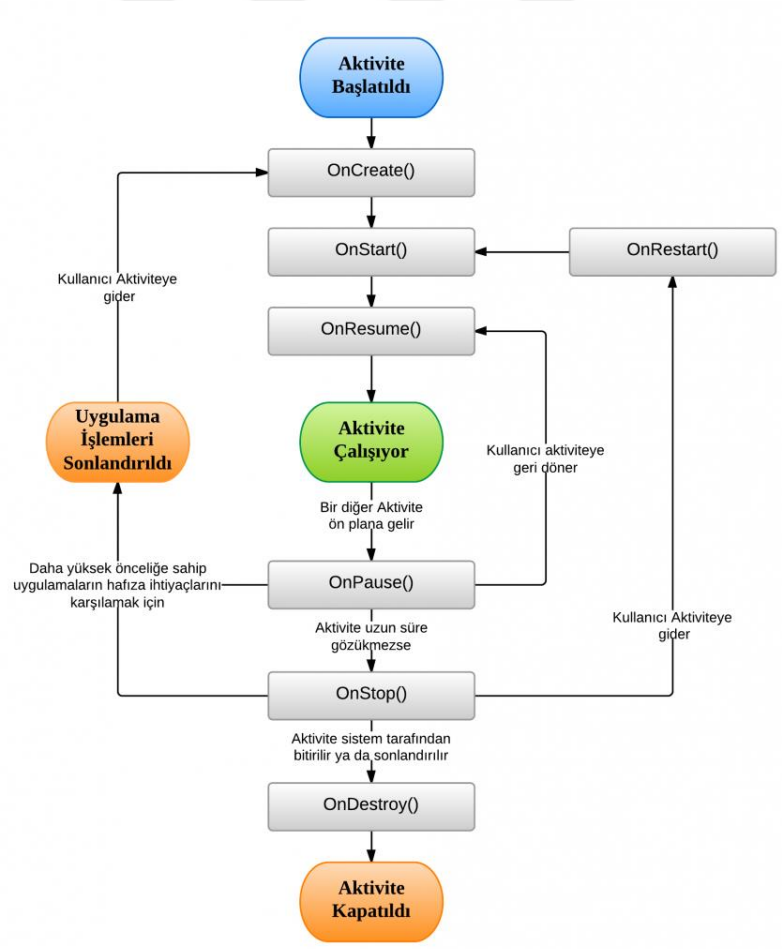
Android mimarisinde linux çekirdek katmanın üstünde C programlama dili ile yazılmış sistem kütüphaneleri, tarayıcıların desteklenmesi için Webkit, grafiksel işlemler için OpenGL, veritabanı işlemleri için SQLite, video ve ses işlemleri için Media Framework bulunmaktadır. Bu kütüphanelere ek olarak, Android Runtime (ART) da bu katmanda bulunmaktadır. Android Runtime, Java kütüphanelerini ve Dalvik Sanal Makinesini (Dalvik Virtual Machine (DVM)) içermektedir. Dalvik Sanal Makinesi, uygulamaların çalışmasını sağlamaktadır. Her bir android uygulaması kendi Dalvik Sanal Makine kopyasında çalışmaktadır. Dalvik Sanal Makinesi'nin yazmaç tabanlı (register based) bir mimarisi vardır. İşlem gücünün ve bellek yapısının kısıtlı olduğu mobil ortamlarda çalışmak için tasarlanmıştır. Android uygulamasının oluşturulması için gerekli işlemler Şekil 2.3'te gösterildiği gibidir. Java ile yazılmış olan kaynak kodlar alınır ve bu kodlar derlenerek bytecode'a çevrilir. Bu dosyalar minimum bellek kullanımının sağlanması için dex dosyalarına dönüştürülür ve çalıştırmak üzere Dalvik Sanal Makinesine verilir.



Şekil 2.3. Android uygulamasının oluşturulması (Codeburst, 2018)

Bir Android uygulamasının temel bileşenleri aktivite, servis, içerik sağlayıcı ve yayın alıcılarıdır.

Aktivite (Activity): Aktivite, uygulamanın GKA sağlayan bileşenidir. Kullanıcı ile uygulama arayüzü arasındaki iletişim bu aktiviteler ile sağlanır. Aktivitilerin hepsi bir yığında (stack) bulunur. Ekranda gösterilecek olan aktivite, yığından çağrılır ve ekranda gösterilir. Her aktivitenin takip etmesi gereken bir yaşam döngüsü (life cycle) vardır. Eğer bir aktivite kapatılırsa, aktivite yönetici (activity manager) tarafından yığından sıradaki aktivite ekrana getirilir. Aşağıda bulunan Şekil 2.4'te bir aktivitenin yaşam döngüsü gösterilmektedir.



Şekil 2.4. Aktivitenin yaşam döngüsü (Activity Life Cycle) (Android Lifecycle, 2019)

Servis (Service): Servis, uzun süreli işlemleri arka planda işleten uygulama bileşenidir. Bu bileşenin bir GKA'sı yoktur. Bir servis diğer uygulama bileşenleri

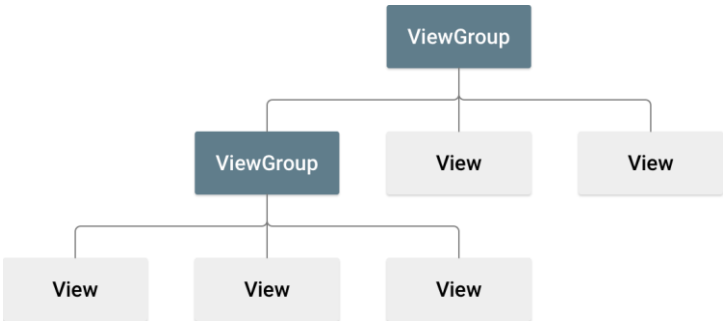
tarafından başlatılabilir, uygulama durdurulmuş veya başka bir uygulamaya geçilmiş olsa bile arka planda çalışmaya devam eder. Bir servis genellikle ağ işlemleri, müzik çalar veya içerik sağlayıcı etkileşimlerini işlemek için kullanılır.

İçerik Sağlayıcılar (Content Provider): İçerik sağlayıcılar, uygulamanın işlemleri arasında yapısal verilerin akışlarını kontrol eden standart arayüzleri (interfaces) oluşturur. Verinin kapsüllenmesi ve veri güvenliği bu bileşen tarafından sağlanır.

Yayın Alıcılar (Broadcast Receiver): Yayın Alıcıları, sistem başlatma, SMS bildirimi veya pil uyarıları gibi sistemden gelen yayın mesajlarını yakalayan bileşenlerdir. Özel bir yayın mesajı aynı zamanda bir uygulama tarafından da tetiklenebilir.

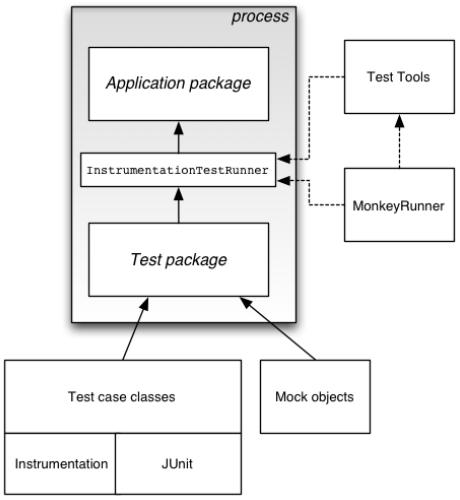
2.2 Android Grafiksel Kullanıcı Arayüzleri

Android işletim sisteminin temel kullanım yeri akıllı telefonlar ve tabletler olduğu için uygulamaların GKA'ları dokunmatik ekranlarda en iyi sonucu alabilmek üzere tasarlanmıştır. Mobil cihazlar arasında çözünürlük farkları olduğu için cihazlara özgü arayüz tasarımlarının da yapılması gerekmektedir. Uygulamalarda yeralan görünümler o ekrana ait olan XML dosyalarına daha önceden tanımlı olan widget'ların eklenmesiyle oluşturulur (EditText, TextView, Button ve RadioButton gibi View nesneleridir). Yerleşim planı (layout) ise ViewGroup objeleri ile sağlanır. Şekil 2.5'te görülen yerleşim planları, alt görünümlerinin (child views) ekran üzerinde nasıl yer alacaklarını control eden kapsayıcılardır. Ekran tasarımlarında kullanılmak üzere önceki widget'ların düzenlenmesiyle özel widget'ların oluşturulması mümkündür.



Şekil 2.5. Android GKA Hiyerarşi (Android UI, 2019)

Android uygulamaları için gerekli olan SDK içerisinde GKA test araçları ile ilgili kütüphaneler mevcuttur. Bu araçlardan ilki, bu tez kapsamında da kullandığımız Android Instrument Framework'tür. Bu framework ile JUnit birim test kütüphanesinin Android uygulamaları için özelleştirilen test sınıfları sunulmaktadır. Test altındaki uygulamanın yapılan testler ile ortak uygulama içeriğine (Common Application Context) erişmesi sayesinde aynı işlemde (process) yer alması sağlanır. Uygulama paketlerinin ve test paketlerinin diğer modüller ile olan ilişkisi Şekil 2.6'da gösterilmiştir.



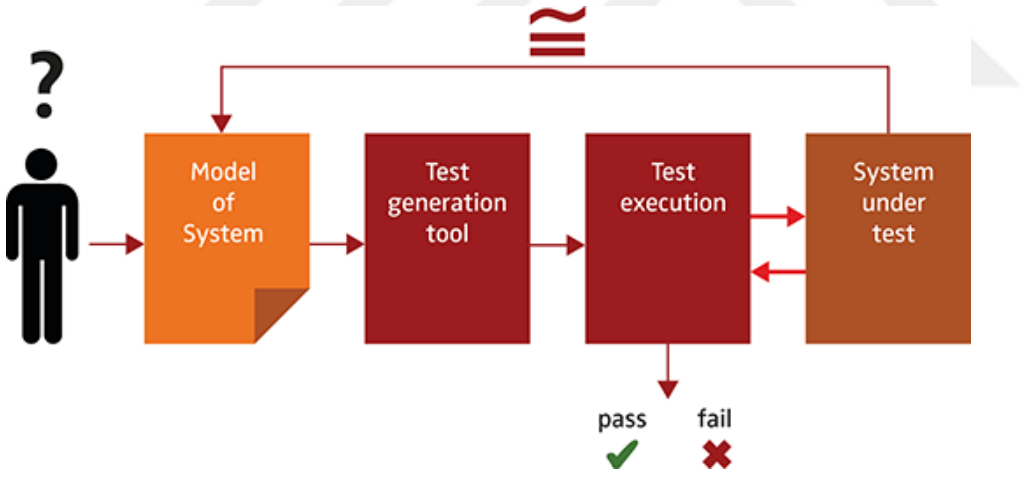
Şekil 2.6. Android Test Araçları (Android Testing Fundamentals, 2019)

GKA testlerinde kullanılan bir diğer araç ise Monkey test aracıdır. Bu araç ile kullanıcı tarafından yapılan; sistem seviyesindeki olaylar, ekrana dokunma, bir butona basma, sürükle-bırak gibi arayüz işlemleri rastegele bir şekilde simüle

edilir. Özellikle, uygulamalarının çökme (crash) ihtimaline karşın stabilite testlerinde kullanışlıdır. Stres testlerinde uygulamanın cevap vermediği ya da beklenmeyen bir hata ile karşılaşma durumlarında test durur ve durum rapor edilir. Test aracı ile her seferinde aynı rastgele sayıdaki olay çalıştırılarak bir etkinlik akışı tekrarlanabilir. Aynı olay sırasına sahip testin ilk seferinde problem görülmezken, uygulamanın aynı olay sırasıyla beşinci defa çalışmasından sonra problemle karşılaşılabilir.

2.3 Model Tabanlı Test (MBT) Yöntemi

Model tabanlı test yöntemi; davranışsal ya da yapısal modeller kullanılarak otomatik bir şekilde test senaryoları oluşturularak, bu senaryolar üzerinden yapılan test yöntemidir. Temelde 4 aşamadan oluşur; modelleme, test üretimi, testlerin çalıştırılma ve çıktılarının alınmasıdır.



Şekil 2.7. Model Tabanlı Test (Embedded Systems Innovation, 2018)

Günümüzde kullanıcılara sunulan GKA'lar çok farklı şekillerde karşımıza çıkmaktadır, bu farklılık bazen karmaşık bir halde almaktadır. Karmaşıklığı azaltmanın yollarından biri GKA'yı modellemektir.

2.4 Modelleme Yöntemleri

Bu bölümde Sonlu Durum Makineleri (Finite State Machines), Düzenli İfadeler (Regular Expressions), Olay Sıra Çizgeleri (Event Sequence Graph) ve Olay Akış Çizgeleri (Even Flow Graph) ile yapılan modelleme yöntemlerinden bahsedilecektir.

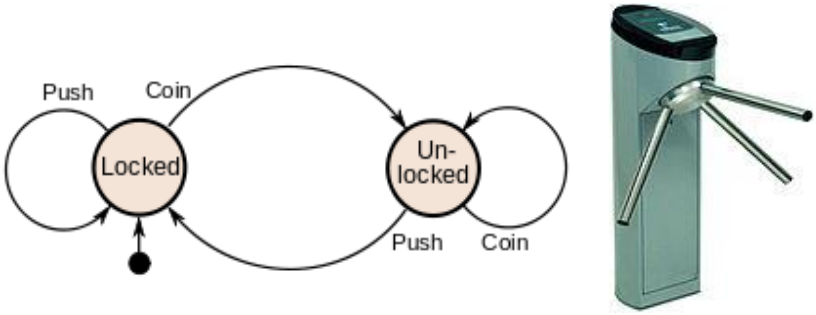
2.4.1 Sonlu Durum Makineleri (SDM)

Sonlu sayıda duruma (state) sahip, verilen bir giriş değerini bir durumdan diğer bir duruma ileten ve geçişler sonucunda bir çıkış üretebilen matematiksel modellere sonlu durum makinesi denir.

Bir SDM, 5 öğeden (S, X, Y, h, s_0) oluşur (Lee and Yannakakis, 1996).

- S, n tane durumu temsil eden sonlu bir kümedir.
- X, boş olmayan sonlu bir girdi sembolleri kümesidir.
- Y, boş olmayan sonlu bir çıktı sembolleri kümesidir.
- s_0, başlangıç durumunu temsil etmektedir.
- h, davranış fonksiyonunu (behaviour function) temsil eder, geçişleri sağlar.

Aşağıda örnek olarak bir turnikenin sonlu durum makinesi verilmektedir. Turnikenin 2 durumu bulunmaktadır; bunlar kilitli (geçişe kapalı) ve kilidi açık (geçişe açık) durumlarıdır. Başlangıç durumu kapalı olan turnike para atıldıktan sonra açılmakta ve geçiş yapıldıktan sonra tekrar kitlenmektedir. Açık iken para atılırsa durumu değişmeyecek, halen geçişe açık olacaktır. Bu durumu SDM'ler ile kolaylıkla modelleyebilir ve sistemin beklenen girdilerini çıkartabiliriz.

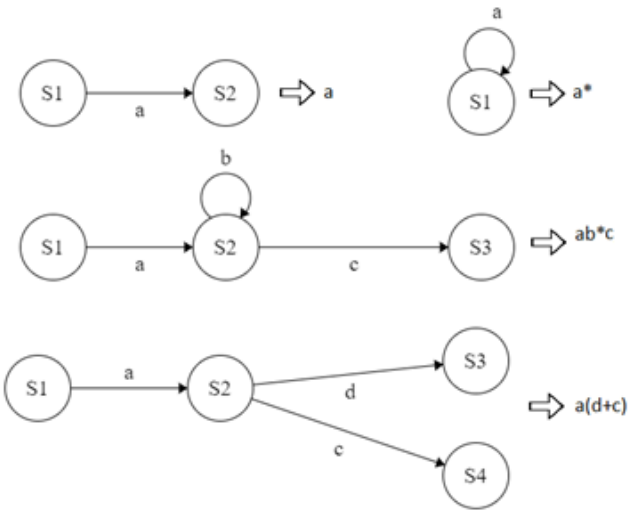


Şekil 2.8. Sonlu Durum Makinesi

2.4.2 Düzenli İfadeler (Dİ)

Düzenli ifadeler (Regular Expressions (RE)), Belirli bir kalıbı tanımlayan karakter dizisidir. Bu diziler şağıda belirtilen operatörler kullanılarak kurulur.

- Bitiştirme; belirli bir sembol ile ifade edilmeyen bir operatördür. Öyle ki ab gibidir ve 'a izlenir b tarafından' anlamındadır.
- Seçim + ile gösterilir. Öyle ki a+b gibidir ve 'a veya b' anlamındadır.
- Yineleme * ile gösterilir. Öyle ki a* gibidir ve 'a isteğe bağlı olarak tekrarlanır. Şekil 2.9'da düzenli ifadeler (Dİ) ve SDM'lerin ilişkisi gösterilmektedir.



Şekil 2.9. Düzenli İfadeler

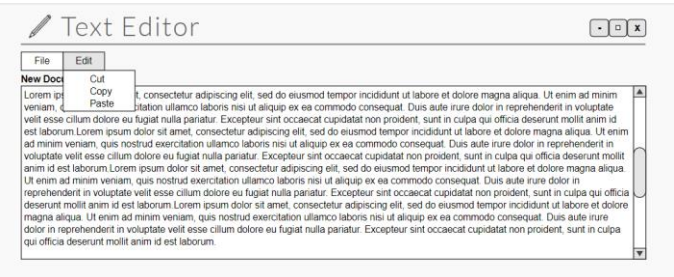
2.4.3 Olay Sıra Çizgeleri (OSÇ)

Olay sıra çizgeleri (Event Sequence Graph (ESG)), test altındaki uygulamayı daha net bir şekilde açıklayan yönlü olay çizgeleridir. Olaylar düğümler ile temsil edilmekte, yönlü oklar ise olayların birbirine olan geçişlerini tanımlamaktadır

Bir olay sıra çizgesi 4 öğeden (N, A, S, F) oluşur (Belli, 2001; Belli et al., 2017).

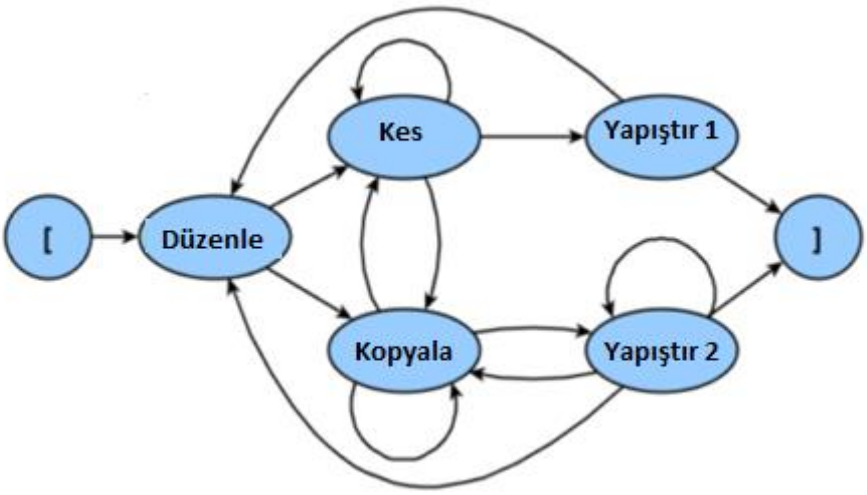
- N olayları temsil eden sonlu bir düğüm kümesidir.
- A ⊆ N x N, olaylar arasındaki ilişkiyi gösteren sonlu bir kümedir. Çizgede yer alan x ve y elamanları arasında, y den x'e giden bir yay varsa bu ilişkileri olduğunu gösterir ve kümenin içerisinde yer alır.
- S ⊆ N x N, ilk veya başlangıç olaylarını temsil eden seçkin bir kümedir.
- F ⊆ N x N, bitiş veya final olaylarını temsil eden seçkin bir kümedir.

Yukarıda bahsedilen tanımlar OSÇ'nin dayandığı model için basit bir yaklaşım uygulandığını göstermektedir. Olay sıra çizgelerinde yer alan her düğüm, uygulamaya özel anlamları varsa, dikkate alınmayan bir olaydır. Çizgede yer alan her bir yay iki olay dizisinin sırasını temsil eder. Şekil 2.10'da çok basit bir metin editörü gösterilmektedir, editör üzerinde Edit (Düzenle) sekmesi altında Cut (Kes), Copy (Kopyala) ve Paste (Yapıştır) olmak üzere 3 tane olay vardır.



Şekil 2.10. Basit bir metin editörü

Metin üzerinde düzenleme işlemi yapılmadan önce belirli bir metnin seçilmiş olması beklenmektedir. Metin seçildikten sonra düzenleme sekmesi altındaki kesme, kopyalama, yaıpştırma işlemlerin yapılması uygun olacaktır. Modeli bu şekilde düşündüğümüzde ilk olarak düzenleme alanı sonrasında ise kesme, kopyalama işlemleri yapılabilir. Şekil 2.11'de ilgili OSÇ'nin çizimi yapılmıştır. Dikkat edilirse metni yapıştırma durumu için iki tane olay söz konusudur. Yapıştırma 1 olayı, kendisine Kes olayıyla geçiş yapılmasını sağlarken, Yapıştırma 2 olayı ise Kopyalama olayının arkasından meydana gelmektedir. OSÇ'lerde başlangıç için "[" işareti ve bitiş için "]" işareti kullanılmaktadır.



Şekil 2.11. Olay Sıra Çizgeleri (Belli, 2001)

2.4.4 Olayı Akış Çizgeleri (OAÇ)

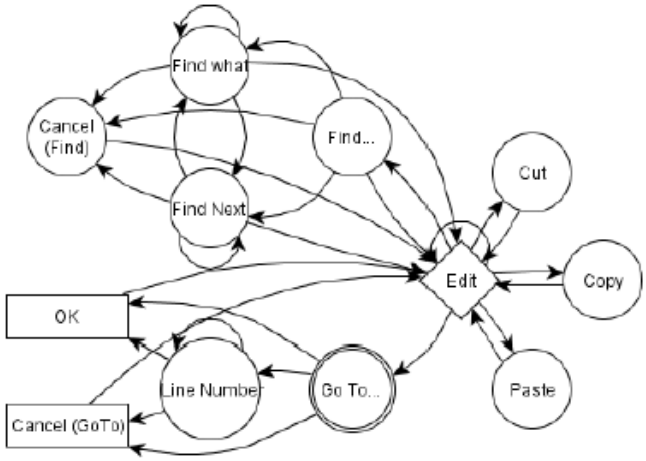
Olay akış çizgeleri (Event Flow Graph (EFG)), GKA'ların bileşen tabanlı yapısını dikkate alır. Örneğin, menü açma (menu-open) olayları, kısıtlanmış odak (restricted-focus) olayları, kısıtlanmamış odak (unrestricted-focus) olayları, sonlardırma (termination) olayları ve sistem ile etkileşim (system-interaction) olayları birbirinden ayrılmış olur (Memon et al.,2003).

Olay sıra çizgeleri üzerinde, uygulamalara ait bilgilerden yola çıkarak her bir düğüm için semantik bir anlam yüklendiğinde olay akış çizgelerinin elde edildiğini düşünebiliriz.

Bir olay akış çizgesinin olması için GKA'ünün bir bileşenine C diyelim. C bileşeni, 4 ögeden (V, E, B, I) oluşur (Belli et. al., 2012).

- V, bileşendeki tüm olayları temsil eden noktalar kümesidir. Bu kümenin her bir elemanı C içerisinde yer alan bir olayı temsil eder. Bu olay menü açma, kısıtlanmış odak, sonlandırma ya da sistem etkileşim olayı olabilir.
- E ⊆ N x N, olayların arasındaki ilişkinin yönlü kenarlar ile temsil edildiği kümedir. Olay e_j'nin ancak ve ancak e_i olayından hemen sonra meydana geldiği durumlarda e_j olayını takip ettiğini varsayalım. Kenar (v_x, v_y) ε E, ancak ve ancak v_y olayının v_x olayını takip ettiği durumlarda sağlanır.
- B ⊆ V, bileşenin ilk çağrıldığında kullanıcıya gösterilebilen olayları temsil eden kümedir.
- I ⊆ V, bileşenin kısıtlı odak olayları kümesidir.

Aşağıdaki Şekil 2.12'de örnek bir olay akış çizgesi görülmektedir.



Şekil 2.12. Örnek bir olay akış çizgesi (Belli et. al., 2012)

Bu olay akış çizgesinde, yer alan “Edit” olayı, bir menü açma olayıdır ve bu yüzden kare ile gösterilmiştir. Çift çember ile çizilmiş “Go To..” ise kısıtlı odak olayıdır. Dikdörtgen içerisinde yer alan “Ok” ve “Cancel” ile gösterilen kısımlar ise sonlandırma olaylarıdır. Bu olaylar sayesinde “Go To..” olayından çıkılmış olunur. “Find” penceresinin herhangi bir modu olmadığı için kısıtlanmamış odak olaylarına ve “Find what”, “Cancel (Find)”, “Cut”, “Copy”, “Paste” olayları ise sistem etkileşim olaylarına birer örnektir. Bunlara ek olarak; “Edit” çizgedeki tek başlangıç olayıdır. Diğer olaylar başlangıç olayı gerçekleşmeden gerçekleşemezler. Diğer bir deyişle, “Edit” sekmesine tıklanmadan o sekmenin altındaki “Cut”, “Copy” ve “Paste” gibi menüleri göremeyiz.

2.5 Mutasyon Test Yöntemi

Mutasyon testi, hata tabanlı bir test tekniğidir. Mutasyon yeterlilik skoru (mutation adequacy score) ile test kriteri sağlanır. Bu kriter, hataları bulma konusunda test setinin etkinliğini ölçmekte kullanılır. Bu test genel prensip olarak bir programcının sıklıkla yapabileceği hataları temel alır. Mutantın yeri ve tipi doğru seçilerek istenilen test yeterlilik kriteri simüle edilebilir. Programcının yapması muhtemel olan hataları, kasıtlı olarak orijinal yazılıma eklenir. Hata eklenmiş yazılıma mutant denir. Her bir mutant birbirinden farklı değişiklikler içerir. Aşağıdaki Şekil 2.13’te mutant oluşumuna bir örnek verilmektedir.

Program p	Mutant p'
...	...
if ($a > 0$ && $b > 0$)	if ($a > 0$ || $b > 0$)
return 1;	return 1;
...	...

Şekil 2.13. Mutant Sistemin Oluşumu (Jia and Harman, 2011).

Belirli bir test setinin kalitesini değerlendirebilmek için, o test setiyle beraber mutantlar test edilir. Test seti içerisinde yer alan herhangi bir test

senaryosunun sonucu orijinal programdan elde edilen sonuçlardan farklıysa, mutant dolayısıyla hata tespit edilmiş olur.

Mutasyon testinin sonucunda bir mutasyon skoru hesaplanır. Bu skor giriş setinin kalitesini gösteren bir skordur. Mutasyon skoru s, tespit edilen mutantların sayısı m, toplam mutant sayısına t olarak alınırsa, aşağıdaki şekilde hesaplanır (Jia and Harman, 2011).

$$\text{Mutasyon skoru} = s = \frac{m}{\text{t}}$$

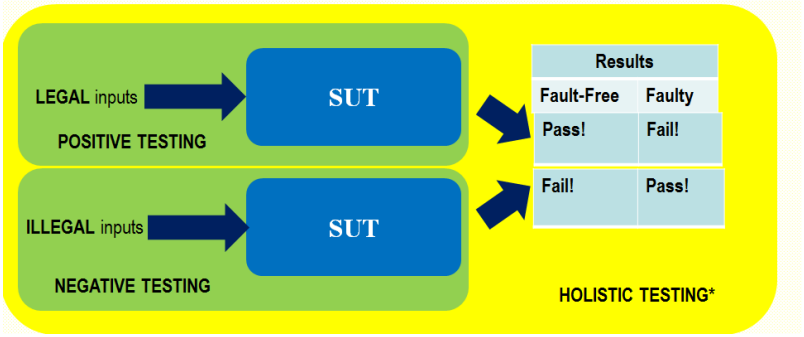
2.6 Bütünsel Test Yöntemi

Bütünsel test yöntemi, geliştirilecek olan sistemin hem istenilen hem de istenilmeyen özelliklerinin modellenmesini kapsayan bütüncül bir yaklaşım benimser (Belli, 2001). Sistemin istenilmeyen özelliklerinin belirlenmesi mutantlar yardımıyla, istenilen özelliklerinin belirlenmesi orijinal yazılımın modellenmesiyle gerçekleştirilebilinir. Bu bütüncül yaklaşımda aşağıdaki test yöntemlerinden bahsedilmektedir.

- Pozitif Test: Test altındaki sistemi olması gereken girdiler ile test etme (legal girdiler ile).
- Negatif Test: Test altındaki sistemi beklenmeyen girdiler ile test etme. (mutant sisteme ait illegal girdiler ile)

Yapılan testler sonucunda ortaya başarılı (successful) ve başarısız (fail) olan test senaryoları çıkmaktadır. Bir örnek ile açıklamak gerekirse, çevrimiçi olarak alışveriş yapılan sitelerin üye girişi sayfalarında kullanıcı adı alanına e-mail adresi girilmesi beklenmektedir. Üyelerin başarılı bir şekilde giriş yapabilmeleri için e-mail adreslerini girmeleri pozitif testi oluştururken, e-mail adresi için gerekli özellikleri taşımayan (sadece rakamlardan oluşan ya da @ içermeyen adres) bir adresin girilmesi ise negatif testi oluşturmaktadır. Üye girişi sayfasının pozitif ve

negatif test senaryoları ile test ediliyor olması ise bütünsel test kavramı ile sağlanabilir.



Şekil 2.14. Bütünsel Test (Belli, 2001)

3. İLGİLİ ÇALIŞMALAR

Mobil uygulamaların grafiksel kullanıcı arayüz testleri ile ilgili araştırma yapıldığında özellikle iki yöntem karşımıza çıkmaktadır. Bunlar rastgele (random) ve model tabanlı (model based) test yöntemleridir. Rastgele test etme yönteminde, test edilmek istenen uygulamaya özgü bir tanımlama yapılmaz. Genelde yapılacak olan testin süresi ya da toplam test sayısı belirtilir. Test dizisi üretim yöntemi olarak da rastgele test dizisi üretimi sağlanır. Model tabanlı yöntemleri incelediğimizde test altındaki sistem (System Under Test (SUT)) bir model ile ifade edilir. Modelleme aşamasında bu işlem elle (manual) ya da otomatik bir şekilde yapılabilir.

Rastgele test etme yönteminde uygulamaya dair bir bilgi bilinmemektedir. Uygulamalar üzerinde yapılan rastgele test ile aslında kara kutu testi (black box testing) sağlanmaktadır. Testi yapan kişi kara kutu testi sırasında uygulamanın kaynak koduna ihtiyaç duymamaktadır. Rastgele test yöntemlerinin başında Monkey (Monkey, 2018) ve Dynodroid (Machiry et al., 2013) araçları yer almaktadır. Monkey aracı ile test yapan kişinin belirlediği sayıda test dizisi üretimi sağlanır. Test yapılan uygulamalar üzerinde özellikle stres testleri için tercih edilmektedir. Diğer bir rastgele test yapılmasına olanak sağlayan Dynodroid aracı da benzer bir rastgele test dizisi üretme yöntemi kullanmaktadır. İki aracın

karşılaştırılmasıyla ilgili yapılan deneysel çalışmalara göre Dynodroid, kod kapsama ve hata yakalama açısından Monkey'den daha iyi sonuçlar vermektedir. Fakat dezavantajı olarak, Monkey'e oranla 5 kat daha yavaş çalışmaktadır.

Test dizisi üretimi açısından rastgele test dizisi üretim yöntemleri daha hızlı ve kolay çözümler olarak görülürken aslında fazlalık test dizilerini içermesi açısından kullanışsız test dizisi üretilmesine neden olmaktadırlar (Choudhary et al., 2015). Bu bağlamda model tabanlı test etme yöntemleri etkili test dizilerinin üretilmesi açısından önem kazanmaktadır.

Yazılım testi alanında, model tabanlı test konusunda Chow'un (1978) çalışmaları önemli bir yer tutmaktadır. Chow yaptığı çalışmalarda, test edilecek olan yazılımı sonlu durum makineleri (finite state machine (FSM)) ile modellemiş ve modellerinden test dizilerinin üretilebilmesi için W-Method isminde yeni bir yöntem ileri sürmüştür.

Olay sıra çizgesi (OSÇ) ve olay akış çizgesi (OAÇ) kavramları, sonlu durum makinelerinden (SDM) farklı olarak sırasıyla Belli (2001), Menon ve diğerleri (2001) tarafından ortaya sürülmüştür.

Model tabanlı test (MBT) (Belli et al., 2015), sistemi temsil etmek ve otomatik olarak test senaryolarını bu modelden türetmek için model adı verilen bir soyutlamanın oluşturulmasını içerir. MBT, genellikle uygulamada test performansını etkileyen belirli varsayımları veya gereksinimleri kullanan çeşitli yaklaşımlar kullanılarak gerçekleştirilebilir.

Model tabanlı test yönteminin mobil uygulamalar üzerinde uygulanması güncel olan bir çalışma alanıdır (Baek and Bae, 2016; Muangsiri and Takada, 2017; Su et al., 2017). Bu konuyla ilgili birçok araç ve yöntem ileri sürülmüştür. MobiGUITAR (Mobile GUI Testing Framework) aracı bu konuda geliştiren araçlar arasındadır (Amalfitano et al., 2015). Bu araç ile ileri sürülen yöntem mobil kullanıcı arayüzünü ripleme (gui ripping), üretim (generation) ve koşum (execution) aşamalarından oluşmaktadır. Üretim aşamasında, ripleme yöntemi

aracılığıyla model üzerinden test dizileri otomatik olarak üretilmekte ve daha sonra elde edilen test dizileri koşulmaktadır.

Model tabanlı test ile ilgili Android GUI üzerinde yapılan bir diğer çalışmada ise Baek ve Bae tarafından, çok seviyeli GUI karşılaştırma kriter seti önerilmiştir (GUICC). Bu set, GUI modeli oluşturma seviyeleri arasında birden fazla soyutlama seviyesinin seçimini sağlamıştır. Aynı zamanda bu çok seviyeli karşılaştırma kriterlerinin testin etkinliği üzerinde etkisini incelemişlerdir. Elde ettikleri sonuçlarda kendi yöntemlerinin aktivite tabanlı GUI modelinden daha etkin sonuçlar verdiklerini görmüşlerdir (Baek and Bae, 2016).

Diğer bir çalışmada ise Muangsiri ve Takada tarafından davranış modeline dayalı otomatikleştirilmiş GUI testinin en verimli test yaklaşımlarından biri olduğu öne sürülmüştür. Kullanıcılarının kullanımına göre oluşturulan test senaryolarının Markov zinciri gibi istatistiksel modellere dayalı olarak da oluşturulabileceklerini fakat bununla birlikte öncesinde statik analiz yapılması gerektiğini, bunun da çok fazla ön şart ve zaman gerektirdiğini vurgulamışlardır. Bu zorlukları çözmek için, mobil uygulamalar tarafında daha hızlı ve daha fazla test senaryosu kapsanılmasını sağlayan, davranış temelli bir GUI test yaklaşımı önermişlerdir (Muangsiri and Takada, 2017).

Bir başka yaklaşımda ise olasılıksal model tabanlı test yöntemi olan “stoat” tanıtılmıştır. Stoat 2 aşamadan oluşmaktadır. 1. aşamada uygulamanın kendisi girdi olarak verilmektedir ve uygulamanın GUI etkileşim noktalarında tersine mühendislik yapılabilmesi için, ağırlıklı kullanıcı arayüzü arama stratejisini ve statik analizle zenginleştirilmiş dinamik analizleri kullanmaktadır. 2. Aşamada ise Gibbs örneklemini tekrar tekrar mutasyona uğratacak şekilde adapte etmektedir ve mutasyona uğrayan modelden daha fazla oranda kodu ve modeli kapsayan testler üretilmektedir. Bütün testler sırasında testin etkinliğini daha da arttırmak adına bu mutasyonlar rastgele enjekte edilmiştir. Stoat, açık kaynak olan 93 uygulama üzerinde değerlendirilmiştir. Araştırmalar neticesinde; Stoat’ın var olan modelleme araçlarından %17-31 arasında daha fazla kaynak kodu kapsadığı ve

Monkey, Sapienz gibi test araçlarından 3 kat daha fazla benzersiz uygulama çökmelerini tespit ettiği gösterilmiştir (Su et al., 2017).

Keio Üniversitesi tarafından yapılan bir çalışmada ise çok sayıda test otomasyon tekniği üzerinde çalışıldığına ancak yapılan çalışmaların hepsinde yazılımın doğası gereği sürekli değişen uygulamalar üzerinde kullanıcı davranışlarının göz ardı edildiğine dikkat çekilmiştir. Etkili bir biçimde test senaryolarını oluşturup test için gerekli olan eforu azaltabilmek için; GUI modellenmesinin, uygulamanın kullanım analizine bakılarak oluşturulan kullanım modellenmesinin ve kullanım analizinin en yeni uygulamaya göre güncellenmesinin gerekli olduğunu belirtmişlerdir (Miguel and Takada, 2015).

Belli (2001) tarafından ortaya atılan bütünsel test (holistic test) yöntemi ilk olarak GKA yazılım testinin model tabanlı testi için kullanılmıştır. Bu yaklaşımda pozitif test ve negatif test olmak üzere iki temel aşama yer almaktadır.

Demillo ve diğerleri (1978) tarafından mutasyon testi yöntemi yazılım testlerinde kullanılmak için öne sürülmüştür. Bu yöntem test altındaki yazılımı değişime uğratarak mutantların elde edilmesine dayanır. Mutantlar en az bir hatayı içerecek şekilde üretilirler, test aşaması ve üretim aşaması birbirinden ayrıdır. Hata tabanlı bir yaklaşımı benimsemektedir. Oluşturulan mutantlar hatayı yakalama kabiliyetleri göz önünde bulundurularak ölü (killed) ya da canlı (lived) olarak ayrılırlar. Testin sonunda yapılan testin kalitesini ölçebilmek için mutasyon skoru (mutation score) hesaplanır. Bu testin amacı temel olarak oluşturulan test dizilerinin hata bulma yeteneklerinin değerlendirilmesi ve test edilmesidir. Kod tabanlı mutasyon (code-based mutation) ve model tabanlı mutasyon (model-based mutation) olarak mutasyon testi ikiye bölünebilir. Bu tez çalışması sırasında mutasyonlar üretilirken iki yöntem de kullanılmıştır. Model tabanlı yöntem ile mutasyonlar üretilmiştir. Kod tabanlı mutasyonlar ise model tabanlı mutantlara karşılık geldiği için test dizilerinin çalıştırılması sırasında kullanılmıştır.

Mutasyona örnek vermek istersek, iki düğüm arasında var olmayan bir kenarın orijinal modele eklenmesi olarak söylenebilir. Burada sadece bir işlem

kullanıldığı için birinci derece mutasyona örnek teşkil etmektedir. Sadece tek bir operatör (kenar ekleme) kullanılmıştır. Daha fazla operatör kullanılarak oluşturulan mutasyonlar ikinci veya yüksek dereceden mutasyonları oluşturmaktadır. Kullanılan operatörler model tabanlı mutasyon için çeşitlilik göstermektedir (Belli et al., 2016). Android uygulamaları testi ve donanım sistemleri testi için mutasyon testinin kullanımı bulunmaktadır (Deng et al., 2017; Kilincceker et al., 2018).

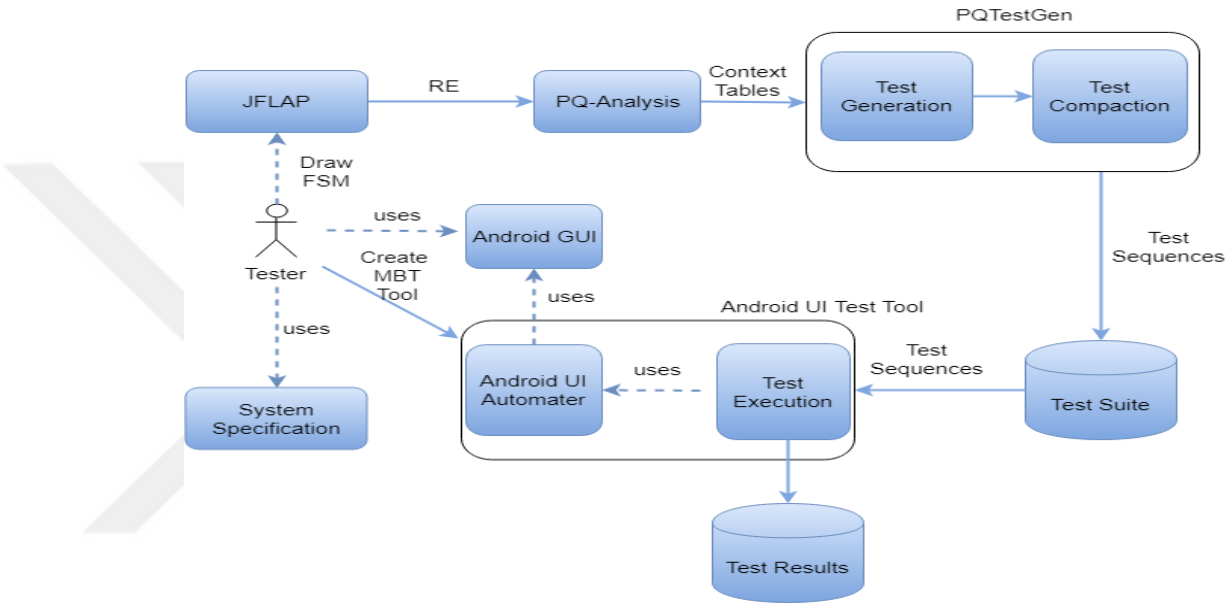
İdeal test kavramıyla ilgili olarak, teorik altyapısı ilk defa Goodenough ve Gerthart (Goodenough and Gerthart, 1975) tarafından yazılım programları testi için öne sürülmüştür. Bu çalışmada test senaryolarının uygunluğunu ölçmek için kullanılan kapsama kriterlerinin güvenilir (reliable) ve uygun (valid) olması gerektiği vurgulanmıştır. Tanımlanan test kriterlerine göre üretilen test dizilerinin tutarlılığı güvenirlik ile vurgulanırken, test dizilerinin hata yakalama yetenekleri ise geçerlilik ile vurgulanmaktadır.

Howden (1976) tarafından ise ideal test yöntemi için yol analizi yöntemini ileri sürmüş ve bu yöntemin geçerlilik gereksinimini sağladığı gösterilmiştir. Bouge (1985) ise teorik çerçeveye eğilim (bias) ve kabul edilebilirlik (acceptability) özelliklerini ideal test yöntemine ekleyerek bu yöntemi genişletmiştir. Ayrıca Kılınççeker ve diğerleri tarafından bu çalışmaya benzer bir ideal test önerisi donanım tanımlama dilleri (hardware description language) testi için ileri sürülmüştür (Kilincceker et al., 2018). Detaylı olarak yapılan literatür taraması sonuçlarında ideal test yönteminin Android uygulaması GKA testi için kullanımına rastlanılmamıştır.

Bu çalışmada ise donanım programları için genişletilen ideal test yönteminin, model tabanlı olarak Android uygulaması arayüz testleri için gerçekleştirilecektir.

4. YÖNTEM

Bu tez çalışması kapsamında, Android uygulaması GKA'sı kullanılarak ideal test yöntemi için bütünsel test ve mutasyon testi kullanılarak yeni bir yöntem öne sürülecektir. Öne sürülen bu melez yönteme ait test sürecinin genel görünümü aşağıda gösterilmektedir.



Şekil 4.1. Öne sürülen yöntem genel görünüm

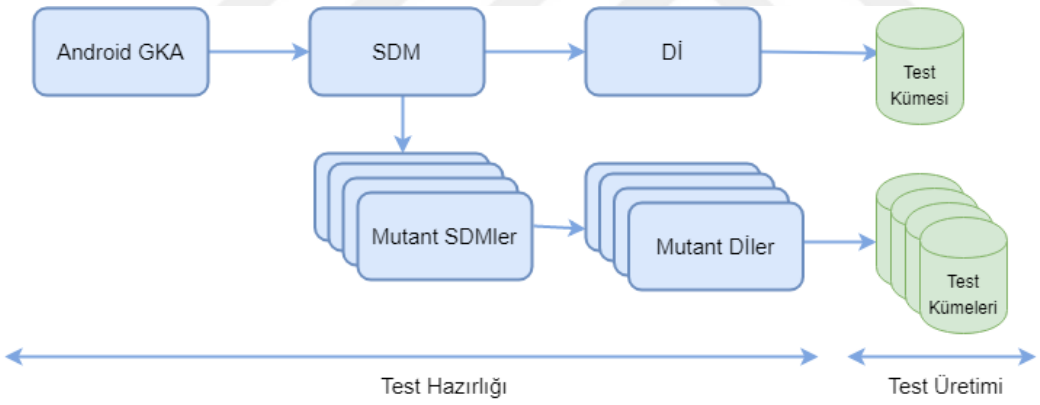
Yukarıda bahsedilen aşamalardan ilki test altında olan Android uygulamasına ait SDM modelinin testi uygulayacak olan kişi tarafından çizilmesini göstermektedir. Uygulama davranışının SDM'ler ile modellenmesinden sonra bu modeller kullanılarak Düzenli İfadeler (Dİ) üretilmektedir. PQ-Analysis aracı (PQ-Analysis, 2018), Dİ'leri kullanılarak bağlam tabloları oluşturmaktadır. PQTestGen (Kilincceker et. al., 2018) aracına girdi olarak verilen bağlam tabloları gerekli test dizilerinin üretilmesini sağlamaktadır. İşlemler sonucunda elde edilen test kümesi, test koşumu yapılabilmesi ve sonuçlarının alınabilmesi için modeli kullanarak oluşturduğumuz test aracına verilir. Her bir test senaryosunun koşumu yapılırken başarılı ve başarısız olan test dizileri için sonuçlar toplanır. Ortaya çıkan sonuçlar üzerinden başarısız olan senaryoların hata durumları incelenir ve gerekli değerlendirmeler

yapılır. Aynı zamanda olay sıra çizgeleriyle (ESG) ve GraphWalker ile oluşturulan modellerin ve test senaryolarının da koşumu sağlanıp elde edilen diğer sonuçlarla kıyaslanacaktır.

Bahsedilen melez yöntem 2 temel süreçten oluşmaktadır. Bunların birincisi test hazırlığı ve test üretimi, ikincisi ise test etme ve test seçimi süreçleridir.

4.1 Test Hazırlığı ve Test Üretimi

Bu süreç test edilen Android uygulamasına ait SDM modellemesini, oluşturulan model kullanılarak mutant modellerin üretilmesini, son olarak orijinal ve mutant SDM modellerinin Dİ'lere dönüştürülmesi aşamalarını kapsamaktadır. Bu aşamalar aşağıdaki şekilde özetlenmektedir.



Şekil 4.2. Test Hazırlığı ve Test Üretimi

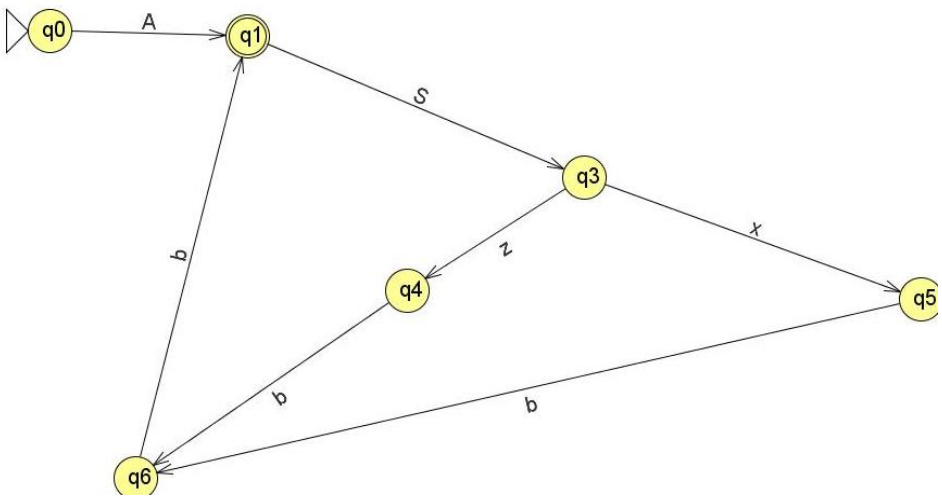
İlk olarak JFLAP (JFLAP, 2018) programı kullanılarak Android uygulamasına ait olacak şekilde bir SDM çizilir. Çizilen modelde her bir düğüm kullanıcının geçiş yapabileceği sayfaları, her bir kenar ise bu sayfa geçişlerini sağlayan arayüz bileşenlerini göstermektedir. Tasarlanan modele ait SDM'nin çok büyük olmaması modelin ölçeklenebilirliği açısından daha iyi olacaktır.

Test altındaki uygulamanın orijinal davranışlarına ait SDM modelleri tamamlandıktan sonra ise mutant modellere ait SDM'ler çizilir. Sistemin

beklemediği (hatalı) test dizilerinin üretilebilmesi için mutant bir modele ihtiyaç duyulmaktadır. Bu mutant modelin yaratımı sırasında test edilecek durumlar (states) için beklenmedik durum geçişleri (transition) gereklidir.

Beklenmedik durum geçişlerinin yapılabilmesi için orijinal uygulama üzerine hatalı kod enjekte işlemi yapılır ve mutasyona uğramış bir uygulama (mutant) elde edilir. Mutant uygulamaların üretilme aşamasında UI Automator (UI Automator, 2018) aracı kullanılmaktadır. Bu araç temelde mobil uygulamanın sahip olduğu GKA bileşenleri hakkında fikir sahibi olunmasını sağlamaktadır. Bu araç sayesinde tersine mühendislik (reverse engineering) yapılarak mobil uygulama üzerinde yer alan hangi bileşenlerin mutasyona uğratabileceği planlanır. Planlanan mutasyonlar kullanılarak mutant uygulamalar elde edilir. Test etme süreçlerinde gösterileceği gibi elde edilen test dizilerinin bazıları da mutant uygulamalar üzerinde koşulacaktır.

Test hazırlığının diğer bir aşaması ise elde edilen orijinal ve mutant uygulamalara ait SDM'lerin, Dİ'lere dönüştürülmesidir. Dİ'lerin oluşturulması işlemi de JFLAP üzerinden yapılabilmektedir. Aşağıda JFLAP üzerinde çizilen örnek bir SDM gösterilmektedir. Bu SDM için üretilen düzenli ifade A((Sxb+Szb)b)* şeklindedir.



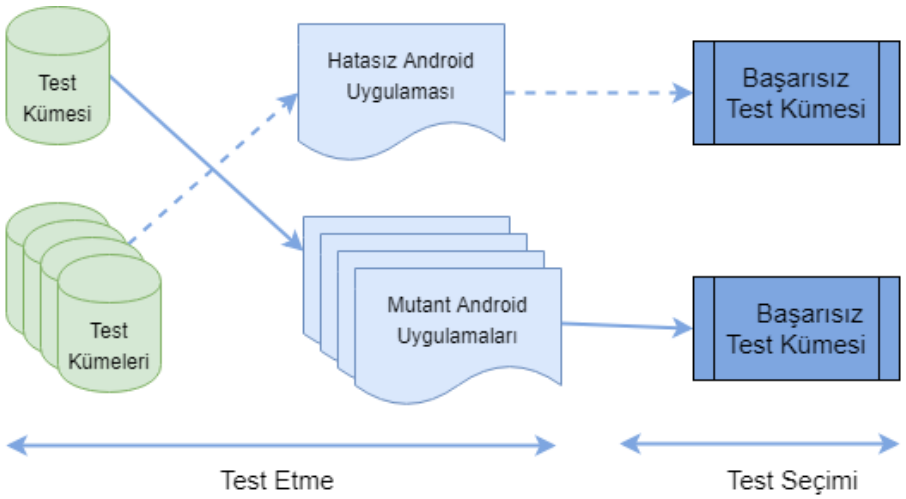
Şekil 4.3. Örnek SDM

Modellere ait Dİ'ler, PQ-Analysis aracı üzerinde kullanılarak bağlam tabloları elde edilir. PQ-Analysis aracının çıktısı olan bağlam tabloları da PQ-TestGen aracına girdi olarak verilir. Bu tablolar içerisinden test edilmek istenilen test senaryoları elde edilmiş olunur. Bu işlem ile birlikte test hazırlığı aşaması IRE (Indexed Regular Expression) için tamamlanmış olur.

Test dizilerinin üretilme aşamasında son olarak; olay sıra çizgeleriyle (OSÇ) hazırlanan modellerden ve GraphWalker aracıyla çizilen modellerden de üretim yapılacaktır. Bu sayede SDM'lerden üretilen IRE test dizileriyle; OSÇ'ler ile üretilen test dizilerinin ve rastgele üretilen test dizilerinin kıyaslaması yapılabilecektir.

4.2 Test Etme ve Test Seçimi

Test hazırlığı sırasında elde edilen test kümeleri çapraz bir şekilde orijinal uygulama ve mutant uygulamalar üzerinde koşulur. Mutant uygulamalar daha önce bahsedilen orijinal uygulamaya yapılan kod tabanlı mutasyonlar (hatalar) ile oluşturulan uygulamalardır. Yapılan çapraz testler sonucunda elde edilen sonuçlar başarılı ve başarısız olan test kümelerini oluşturmaktadır. Aşağıda yer alan şekilde test etme ve test seçimi aşamaları özetlenmektedir.



Şekil 4.4. Test Etme ve Test Seçimi

Çıkan sonuçlar kullanılarak ideal test kümesinin pozitif ve negatif test durumları için ideal test kümesi elde edilir. Koşulan test dizilerinin bazıları başarılı olurken bazıları başarısız olmaktadır. Sürecin sonunda ideal test kümesi elde etmek için orijinal modelden elde edilen test kümesi mutant Android uygulamaları üzerinde koşulur ve bu durumda başarısız olan test dizileri seçilir. Bu sayede pozitif test için ideal test kümesi oluşturulmuş olur. Aynı yöntem kullanılarak negatif test aşamasında mutant (hatalı) modellerden üretilen test kümeleri ise orijinal Android uygulaması üzerinde koşulur ve bu durumda her bir mutant için başarısız olan test dizileri ideal test kümelerinin oluşturulmasına imkân sağlar. Bu sayede Goodenough ve Gerhart'ın (1975) ileri sürdüğü İdeal Test'in güvenilirlik (reliability) ve geçerlilik (validity) gereksinimleri, oluşturulan pozitif ve negatif test kümeleri ile karşılanmaktadır.

5. ÖRNEK DURUM

Bu kısımda bahsedilen yöntemin yapılabilirliğini gösterebilmek için yapılan örnek durum çalışmaları gösterilecektir. Örnek durumun ele alınacağı ve model tabanlı testin uygulanacağı Android uygulaması olarak Telegram (Telegram, 2018) kullanılacaktır. Telegram, açık kaynaklı, hız ve güvenlik odaklı popüler bir mesajlaşma uygulamasıdır. Uygulamaya ait kaynak kod F-Droid (F-Droid, 2018) üzerinde bulunmaktadır. Mutant uygulamaların üretilebilmesi açısından örnek durum çalışmasında kullanılacak olan uygulamanın açık kaynak kodlu bir yazılım olması oldukça önemlidir.

Uygulamaya karar verildikten sonra, modelleme sonucu elde edilen test dizilerinin koşulabilmesi için bir test aracı hazırlanmıştır. Test aracının hazırlanması aşamasında Android uygulamaları üzerinde test yapılabilmesini sağlayan Android Testing Support (Android Testing Support, 2018) kütüphanesi kullanılmıştır. Android Studio IDE'sinde ilgili yazılım geliştirmeleri yapılmış olunup, programlama dili olarak Java kullanılmıştır.

5.1 Test Hazırlığı

5.1.1 Modelleme

Modellemenin ilk aşamasında seçtiğimiz Android uygulamasına ait bir SDM, JFLAP aracı üzerinde çizilmiştir. SDM üzerinde yer alan her bir düğüm, kullanıcının ziyaret edebileceği uygulama sayfalarını gösterirken her bir kenar ise sayfaların arasındaki geçişleri sağlayan arayüz bileşenlerini temsil etmektedir.

Modelleme sırasında SDM'nin çok fazla durum (state) içermesinden dolayı ölçeklenme problemiyle karşılaşılmıştır (Bkz. Ek 1). Bu problem çözümü olarak çok katmanlı bir SDM (HFSM) yapısıyla modellemeye devam edilmiştir. Çok katmanlı SDM yapısıyla yapılan modellemenin katmanlara bölünmesi sırasında bir arayüzün üzerinde yer alan fonksiyonaliteler ile o işlem sonrasında açılan alt sayfalar göz önünde bulundurulmuştur. Uygulamanın ana ekranında yer alan fonksiyonaliteler genel olarak üst katmanı, bu fonksiyonların gerçekleştirilmesi sırasında çıkan sayfalar ise alt katmanlar olarak modellendirilmiştir.

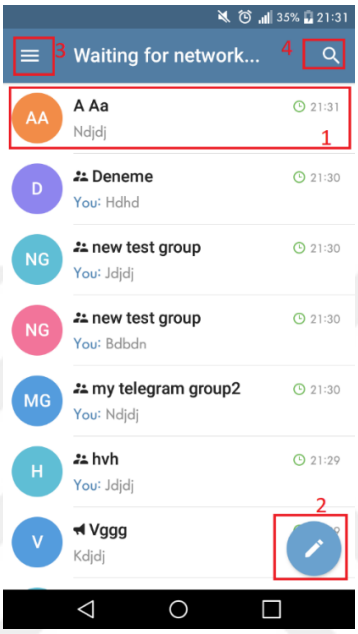
OSÇ'ler ile yapılan modelleme hali hazırda katmanlı bir yapı sunmaktadır. OSÇ ile ilgili modellemeler TSD aracı kullanılarak yapılmıştır. Bu modelleme sırasında birbirini takip eden olaylar gösterilmekte, iki olay arasına çizilen yaylarda herhangi bir geçiş girdisi verilmemektedir.

5.1.1.1 Ana Sayfanın Modellenmesi

Bu tez kapsamında Telegram uygulamasında yer alan özelliklerden mesajlaşma, yeni bir konuşma yaratma, rehberde isim arama ve ana ekranda yer alan mesajlar içerisinde arama kısımları modellenmiştir. Şekil 5.1'de ana ekrandan (üst katman) bu özelliklerin kullanımlarına geçiş sağlayan arayüz bileşenleri (alt katmanlar) gösterilmiştir. Telegram uygulamasının arayüzünde;

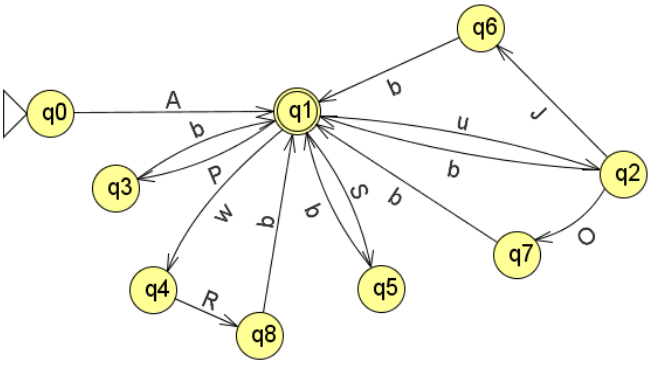
- 1 numaralı kısım mesajlaşma,

- 2 numaralı kısım yeni bir konuşma başlatma,
- 3 numaralı kısım hamburger menüde yer alan rehberde arama yapma
- 4 numaralı kısım ise ana ekranda yer alan mesajlar içerisinde arama yapma işlemlerine geçişleri göstermektedir.



Şekil 5.1. Telegram ana ekran arayüzü

İlk olarak uygulamanın ana ekranına ait kullanıcı arayüzü üst katman olarak SDM ile modellenmiştir. Şekil 5.2'te gösterilen SDM üzerinde yer alan kısımlar, uygulamanın ikonuna tıklanarak açılmasını ve sonrasında yapılan mesajlaşma (1), yeni konuşma başlatma (2), rehberde arama yapma (3), mesajlarda arama yapma (4) özelliklerini kapsamaktadır. Ana ekran modellemesinde q0 uygulamanın başlangıç durumunu (initial state), q1 ise son durumunu (final state) temsil etmektedir. Yani uygulamanın ana ekranı son durum ile temsil edilmektedir.



Şekil 5.2. Telegram ana ekranına ait SDM (üst katman)

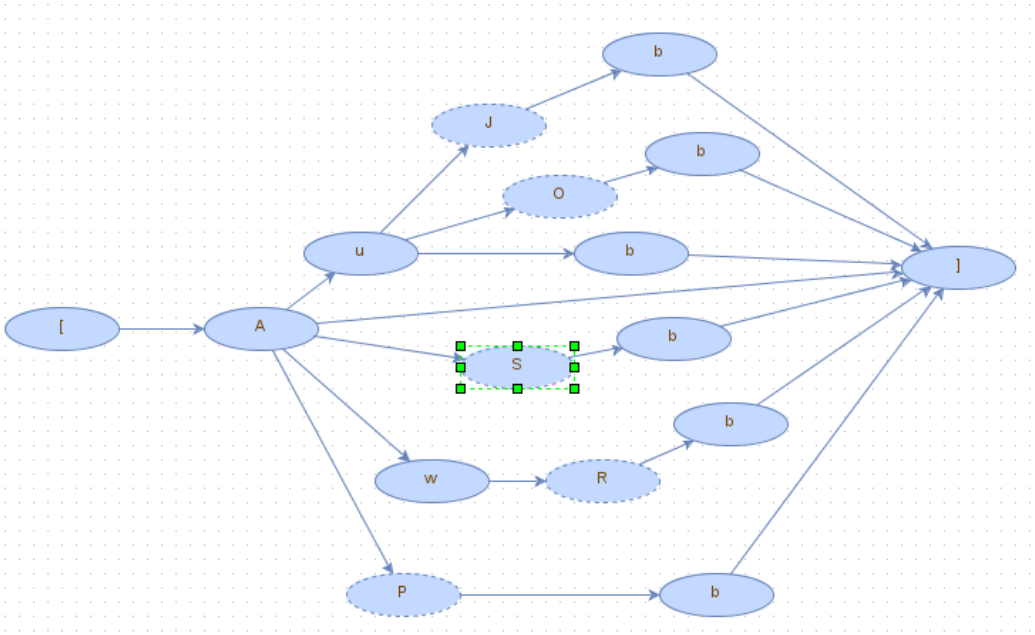
Büyük harfle ile gösterilen geçişler katmanlı (layered) bir SDM yapısı düşünüldüğünde üst katmanı (upper layer) temsil ederken küçük harfler ise alt katmanı (sub layer) temsil etmektedir. Aşağıdaki Tablo 5.2'de üst katman SDM'ye ait harflerin anlamı yer almaktadır (Modelin tamamı için alfabedeki tüm harfler kullanıldığı için, küçük harfle gösterilmesi gereken Şekil 5.3'te gösterilen uygulama ikonuna tıklayıp uygulamayı açma olayı büyük "A" harfiyle gösterilmek durumunda kalınmıştır).

Şekil 5.3. Telegram ikonuna tıklama (üst katman)

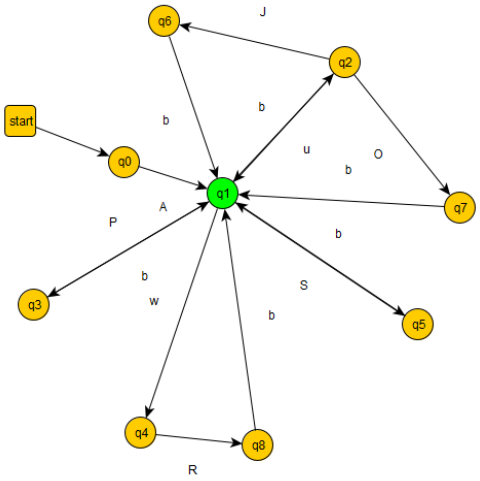
Tablo 5.2. Uygulama Anasayfası Harf Gösterim ve Anlam Tablosu

Harf Gösterimi	Anlamı
A	Uygulama ikonuna tıklama
U	Konuşma listesinde ilk kullanıcıyı seçme
b	Geriye tıklama
J	Metin yazma alanına tıklama
O	Mesajlaşma alanında seçeneklere tıklama
E	Mesajı gönderme
w	Hamburger menüye tıklama
R	Rehbere tıklama
S	Mesajlaşmalar içerisinde ara ikonuna tıklama

Aynı modeli olay sıra çizgeleri (OSÇ) ile çizdiğimiz zaman, SDM üzerinde geçişleri temsil eden girdilerimiz bu sefer çizge üzerindeki düğümleri oluşturur. Düğümler arası geçişler birbirini takip eden olayları belirtir ve girdileri çizim üzerinde gösterilmez. OSÇ ile çizim yaptığımız TSD aracı bize katmanlı bir yapı sağlamaktadır. Çevresi kesik çizgili olan düğümler, alt katmanlara sahip olduğunu göstermektedir. Aşağıdaki Şekil 5.4'te Telegram uygulamasının ana sayfasına ait olan OSÇ yer almaktadır.



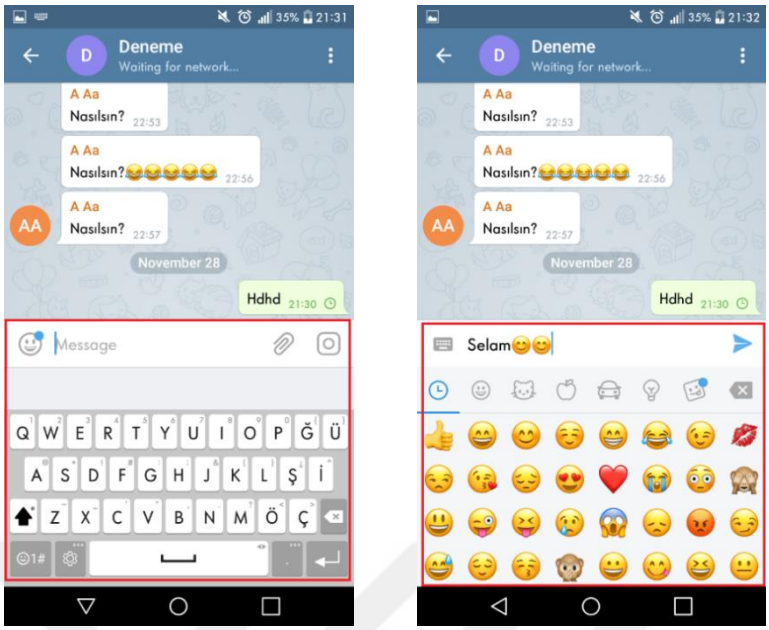
Şekil 5.4. Telegram ana sayfasına ait OSÇ



Şekil 5.5. Telegram ana sayfasına ait GraphWalker modeli

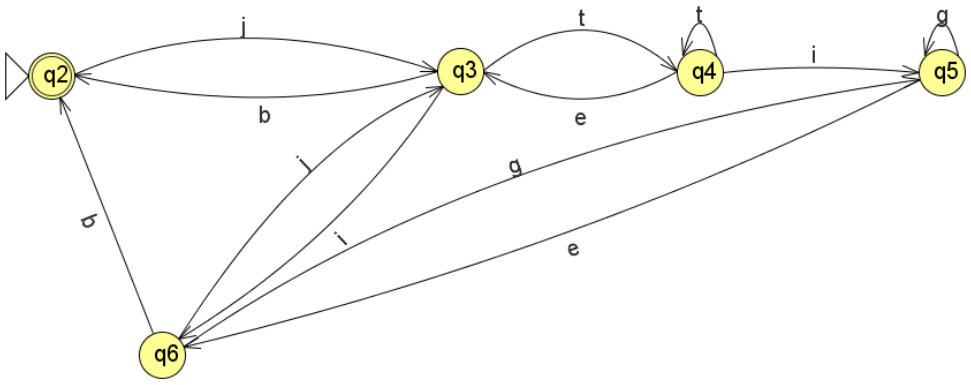
5.1.1.2 Mesajlaşma Sayfasının Modellenmesi

Telegram uygulamasının ana sayfasına ait SDM üzerinde mesajlaşma alanına (üst katmanda J ile gösterilen) gelinebilmesi için öncesinde uygulamanın açılması ve mesajlaşma listesinde yer alan bir kullanıcının seçilmiş olması gerekmektedir. Dolayısıyla mesajlaşma alanına ait SDM öncesinde üst katmanda bu işlemlerin yapılmış olduğunu varsayılır. Daha sonra kullanıcı Tablo 5.2'de gösterilen mesajlaşma işlemlerini bu model üzerinde gerçekleştirebilir. Mesajlaşma sayfasına ait arayüz Şekil 5.6'daki gibidir.



Şekil 5.6. 1B: Mesajlaşma sayfasına ait metin yazma ve emoji seçme arayüzleri (alt katman)

Bu modelin SDM'si 1B ile hatalı modellere ait SDM'ler 1FBX ile gösterilecektir. (Aşağıda yer alan SDM'de mesajlaşmalar ekranında ilk kullanıcıyı seçecek şekilde varsayım yapılarak çizilmiştir.)



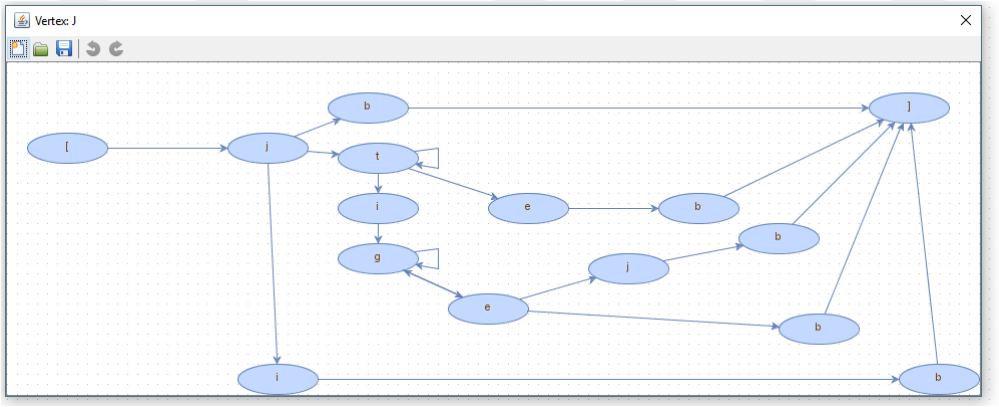
Şekil 5.7. 1B: Mesajlaşma sayfasına ait hatasız SDM (alt katman)

Bu SDM üzerinde kullanılan küçük harfler alt katmanda yapılabilecek olan geçişleri göstermektedir. Aşağıdaki Tablo 5.3'te SDM'ye ait harflerin anlamı yer almaktadır.

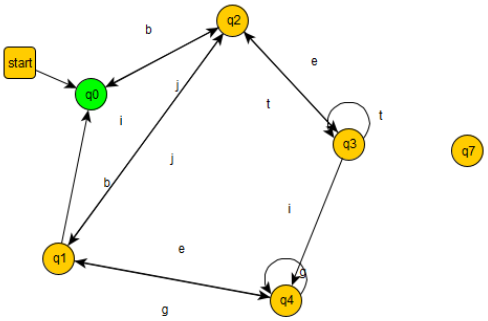
Tablo 5.3. 1B Modeli Harf Gösterim ve Anlam Tablosu

Harf Gösterimi	Anlamı
b	Geriye tıklama
j	Metin yazma alanına tıklama
t	Metin yazma
e	Mesajı gönderme
i	Emojiler ikonuna tıklama (Yüz ifadesi ikonu)
g	Bir tane emoji seçme

Aşağıdaki Şekil 5.8'de Telegram uygulamasının 1 B modeline ait olan OSÇ çizimi yer almaktadır.

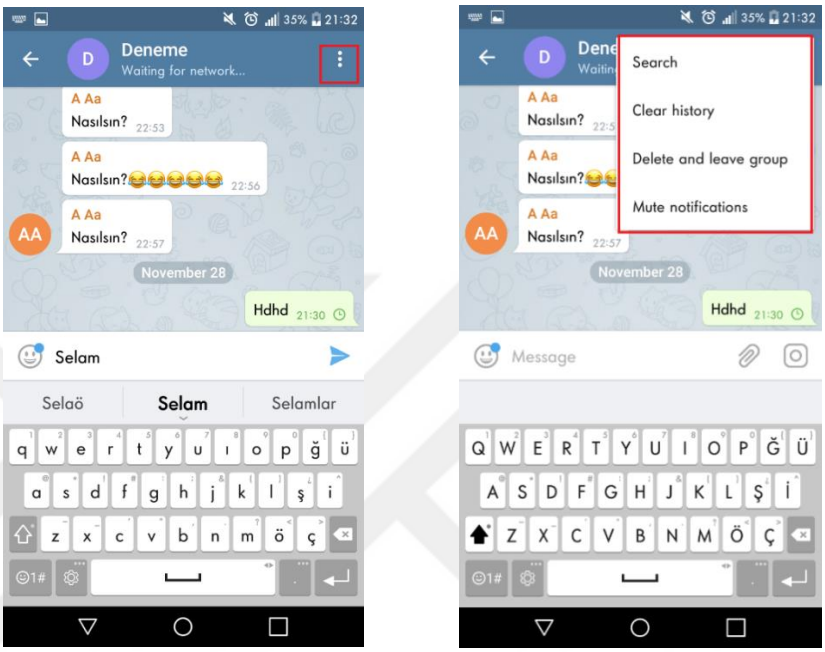


Şekil 5.8. 1B: Mesajlaşma sayfasına ait hatasız OSÇ



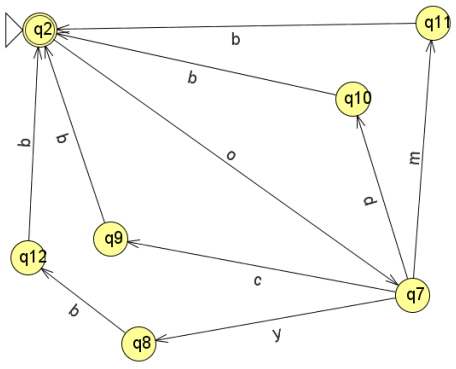
Şekil 5.9. 1B: Mesajlaşma sayfasına ait hatasız GraphWalker modeli

Mesajlaşma sayfasında yer alan diğer bir özellik ise Şekil 5.10'da gösterilen seçenekler kısmıdır. Bu kısımda kullanıcıya içinde bulunduğu mesajlaşma için arama, geçmişi temizleme, mesajı silme ve gruptan çıkma gibi seçenekler sunulmaktadır.



Şekil 5.10. 1A: Mesajlaşma sayfası seçenekler arayüzü

Mesajlaşma sayfasında yer alan seçeneklere ait SDM aşağıdaki gibi olup 1A ile gösterilecektir. Hatalı modellere ait SDM'ler ise 1FA ile gösterilecektir. (Aşağıda yer alan SDM'de mesajlaşmalar ekranında ilk grup konuşmasını seçecek şekilde varsayım yapılarak çizilmiştir.)



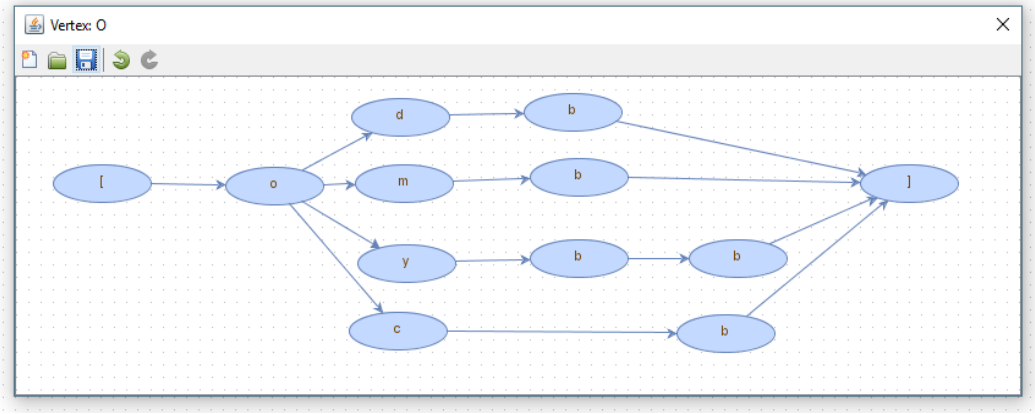
Şekil 5.11. 1A: Mesajlaşma sayfası seçeneklere ait hatasız SDM (alt katman)

Bu SDM üzerinde kullanılan küçük harfler alt bu katmanda yapılabilecek olan geçişleri göstermektedir. Aşağıdaki Tablo 5.4'te SDM'ye ait harflerin anlamı yer almaktadır.

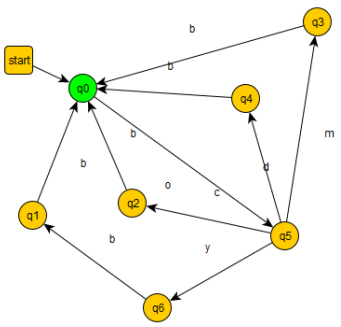
Tablo 5.4. 1A Modeli Harf Gösterim ve Anlam Tablosu

Harf Gösterimi	Anlamı
b	Geriye tıklama
o	Seçeneklere tıklama
c	Mesajlaşma geçmişini silme (Clear history)
d	Mesajlaşmayı sil ve gruptan çık (Delete and leave group)
y	Aramaya tıklama (Search)
m	Bildirimleri sessize alma (Mute notifications)

Aşağıdaki Şekil 5.12'te Telegram uygulamasının 1 A modeline ait olan OSÇ çizimi yer almaktadır.



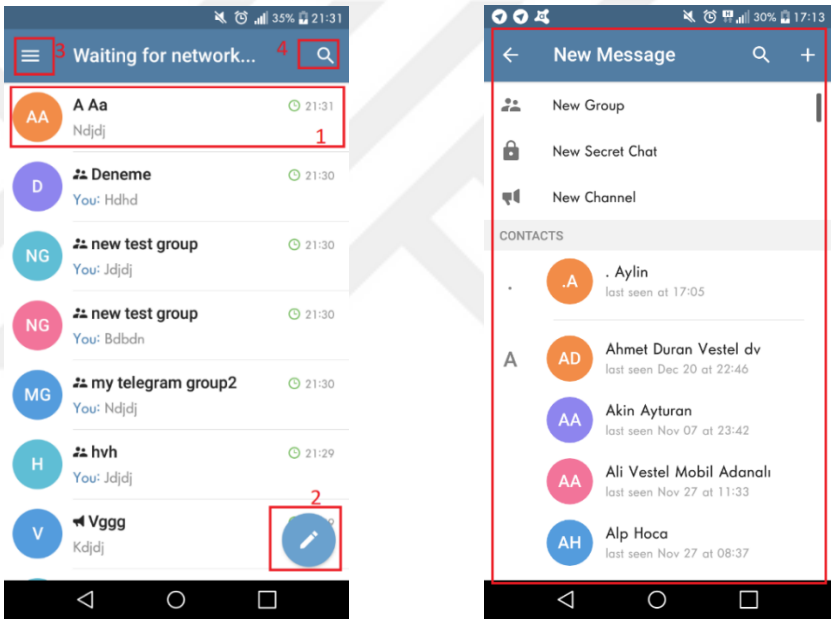
Şekil 5.12. 1A: Mesajlaşma sayfasındaki seçeneklere ait hatasız OSÇ



Şekil 5.13. 1A: Mesajlaşma sayfasındaki seçeneklere ait hatasız GraphWalker modeli

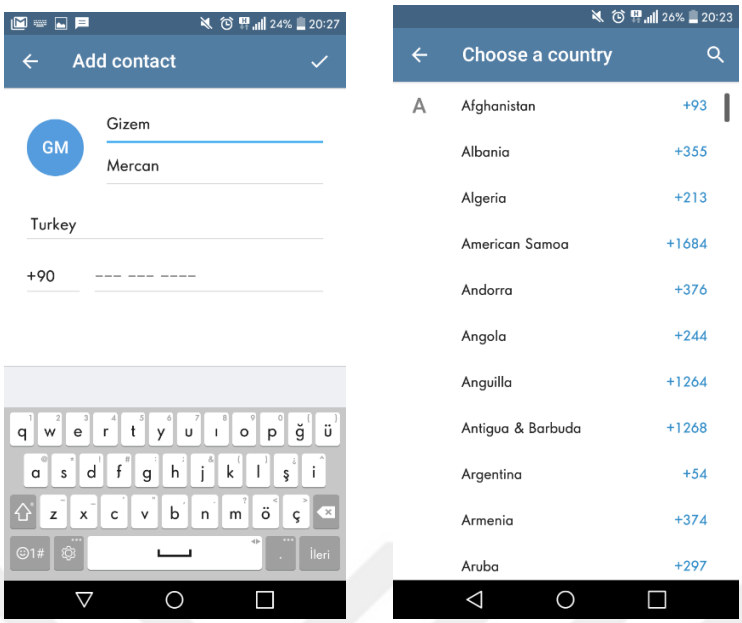
5.1.1.3 Yeni bir Konuşma Oluşturma Sayfasının Modellenmesi

Telegram uygulamasının yeni bir konuşma oluşturmak için anasayfada bulunan kalem ikonuna (üst katmanda P ile gösterilen) tıklanması gerekmektedir. Yeni bir konuşma oluşturma sayfası bu aksiyon alındıktan sonra Şekil 5.14'te görüldüğü gibi ekrana gelmektedir. Bu kısımda kullanıcıya arama yapma, yeni bir grup yaratma, yeni bir kanal kurma, yeni bir kişi ekleme gibi seçenekler sunulmaktadır. Tez kapsamında bu kısmın modellemesini kişi arama ve yeni bir kişi eklenirken seçilen ülke bilgisiyle sınırlandırılmıştır.

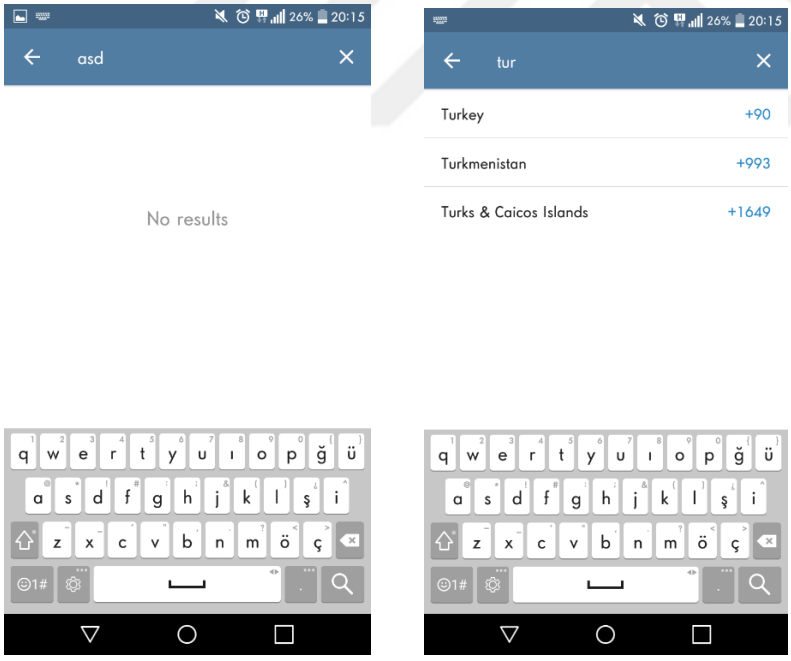


Şekil 5.14. 2: Yeni bir konuşma oluşturma arayüzü

Konuşma oluşturma arayüzünden "+" ikonuna basılarak yeni bir kullanıcı eklenebilir. Kullanıcı ekleme ekranında ülke seçilirken listede olmayan bir ülke girilirse sonucun bulunamaması, listeden bir ülke girilirse sonuçlarda bulunması beklenmektedir. Şekil 5.15 ve Şekil 5.16'da bu sayfalara ait arayüzler görülmektedir.

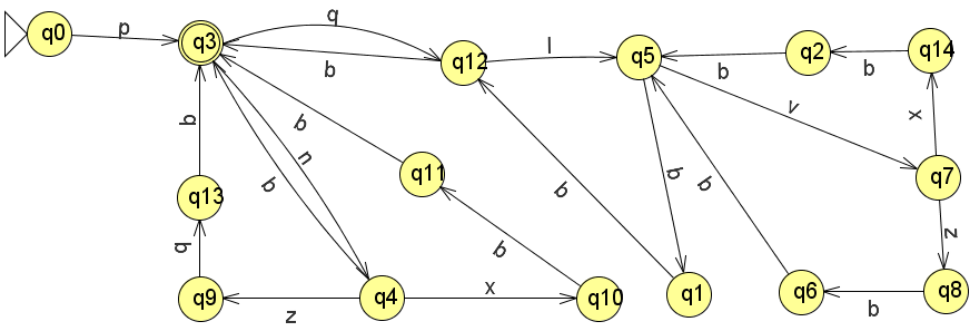


Şekil 5.15. 1A: Rehbere kişi eklerken ülke bilgisinin girilme arayüzü



Şekil 5.16. 1A: Ülke bilgisinin girilme arayüzü

Yeni bir konuşma yaratma sayfasına ait SDM aşağıdaki gibi olup uygulamaya ait anasayfada 2 numara ile gösterilmiştir.



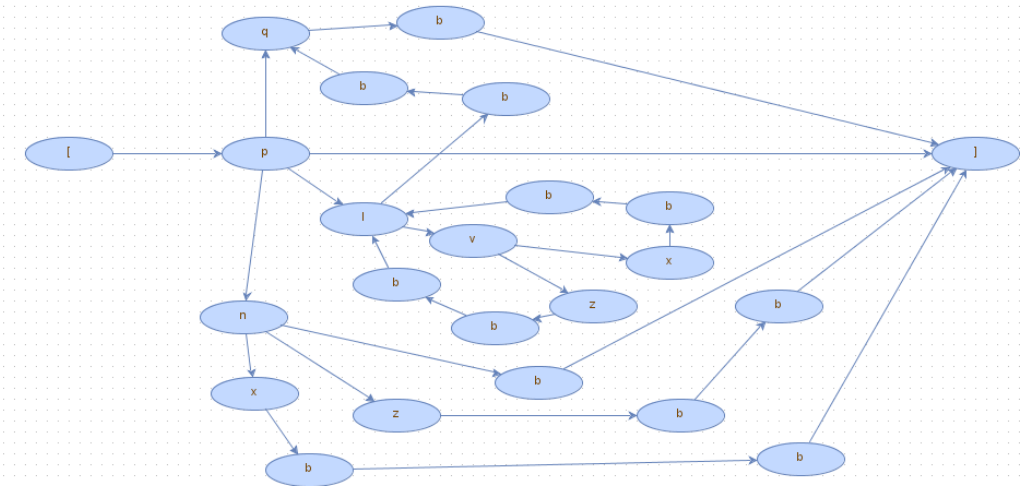
Şekil 5.17. 2: Yeni bir konuşma yaratma SDM (alt katman)

Bu SDM üzerinde kullanılan küçük harfler bu katmanda yapılabilecek olan geçişleri göstermektedir. Aşağıdaki Tablo 5.5'te SDM'ye ait harflerin anlamı yer almaktadır.

Tablo 5.5. 2 Numaralı Modelin Harf Gösterim ve Anlam Tablosu

Harf Gösterimi	Anlamı
B	Geriye tıklama
P	Yeni bir konuşma yaratma
Q	Yeni bir konuşma sayfasında yeni bir kişi ekleme
L	Yeni konuşma sayfasında, yeni kişi eklerken ülkeleri listeleme
V	Yeni konuşma sayfasında, yeni kişi eklerken ülke arama
X	Listede olmayan bir metni arama
Z	Listede olan bir metni arama
N	Yeni bir konuşma sayfasında aramaya tıklama (Search)

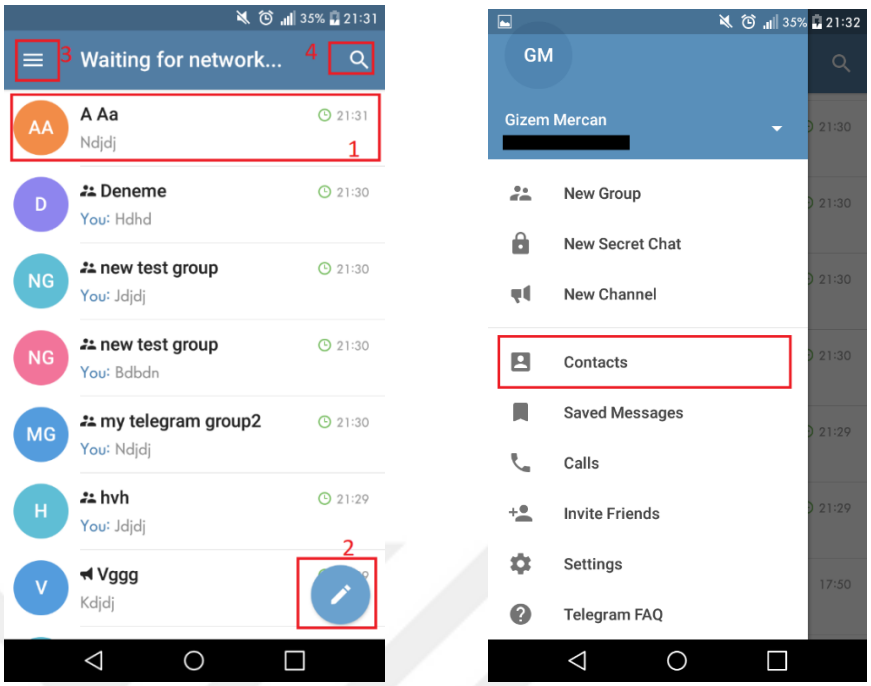
Aşağıdaki Şekil 5.18'de Telegram uygulamasının 2 numaralı modeline ait olan OSÇ çizimi yer almaktadır.



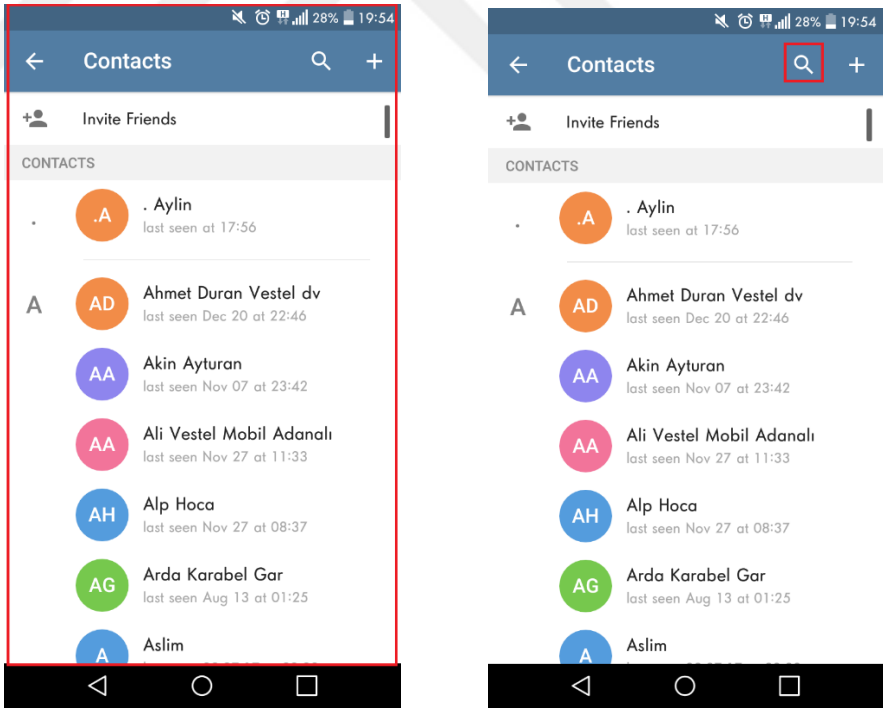
Şekil 5.18. 2: Yeni bir konuşma yaratma OSÇ

5.1.1.4 Rehber Sayfasının Modellenmesi

Telegram uygulamasının hamburger menüsü altında uygulamanın genel ayarları bulunmaktadır. Rehber (Contacts) sekmesi de bu menünün altında yer almaktadır. Yeni bir kişi eklemek için anasayfada bulunan hamburger menü ikonuna (üst katmanda w ile gösterilen) ve sonrasında “Contacts”ın tıklanılması (üst katmanda R ile gösterilen) gerekmektedir. Hamburger menüye tıklandıktan sonra, Şekil 5.19’da görülen ekrana gelmektedir. Bu kısımında kullanıcıya çağrı yapma, yeni bir grup yaratma, yeni bir kanal kurma, yeni bir kişi ekleme gibi seçenekler sunulmaktadır. Tez kapsamında bu kısmın modellemesini Şekil 5.20’de gösterilen rehberde bir kişi arama ile sınırlandırdık.

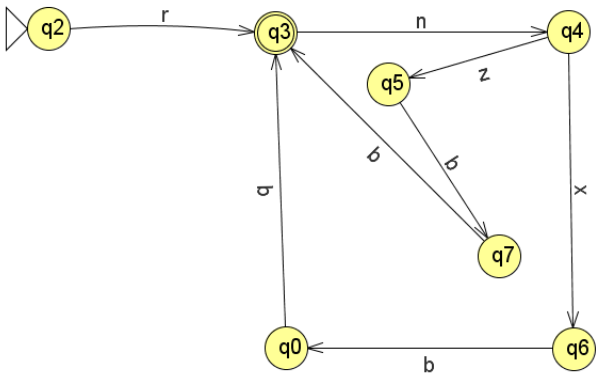


Şekil 5.19. 3: Hamburger menü arayüzü



Şekil 5.20. 3: Rehberde arama yapma arayüzü

Rehber sayfasında yer alan seçeneklere ait SDM aşağıdaki gibi olup 3 numara ile gösterilecektir.



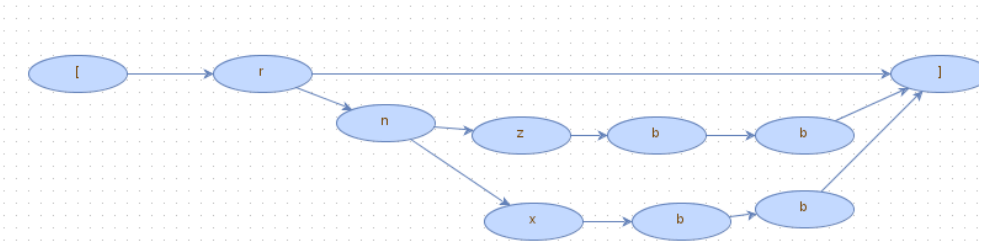
Şekil 5.21. 3: Rehber sayfası SDM (alt katman)

Bu SDM üzerinde kullanılan küçük harfler alt katmanda yapılabilecek olan geçişleri göstermektedir. Aşağıdaki Tablo 5.6'de SDM'ye ait harflerin anlamı yer almaktadır.

Tablo 5.6. 3 Numaralı Model Harf Gösterim ve Anlam Tablosu

Harf Gösterimi	Anlamı
b	Geriye tıklama
r	Rehbere tıklama (Contacts)
n	Rehberde arama (Search)
x	Listede olmayan bir metni arama
z	Listede olan bir metni arama

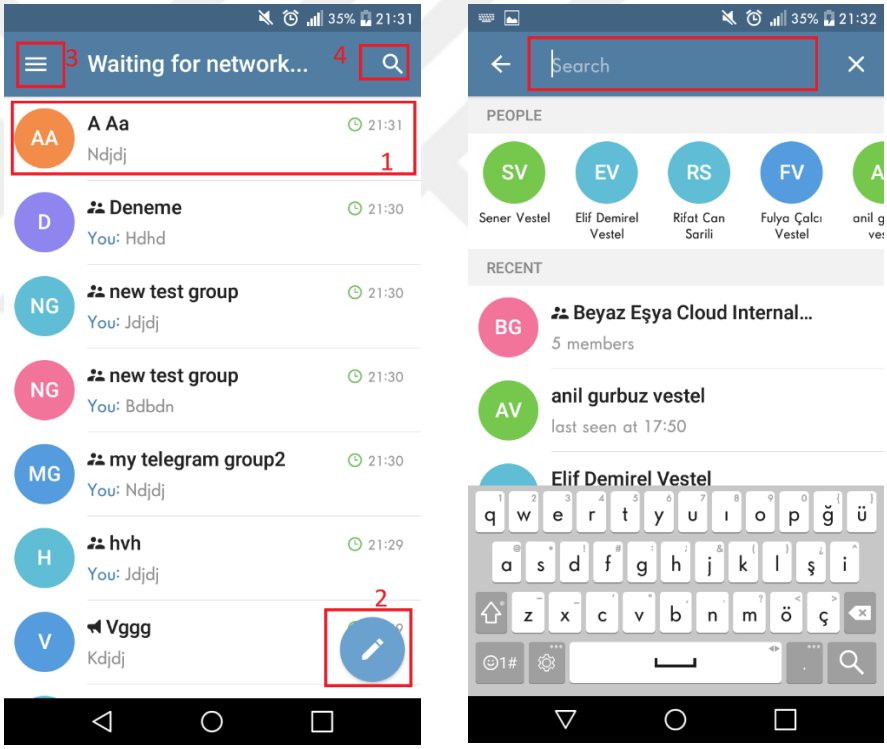
Aşağıdaki Şekil 5.22'de Telegram uygulamasının 3 numaralı modeline ait olan OSÇ çizimi yer almaktadır.



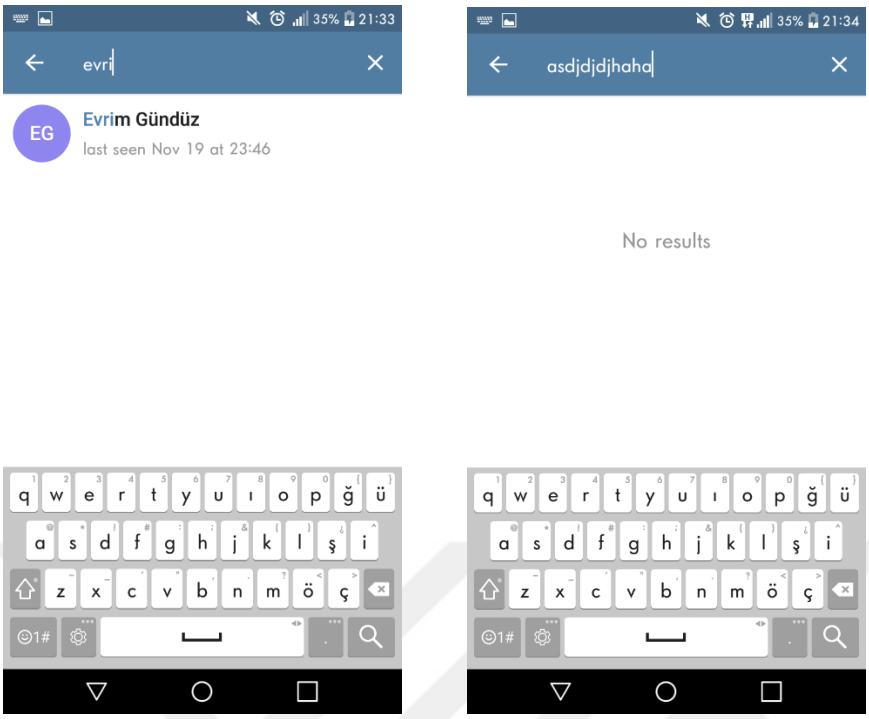
Şekil 5.22. 3 Rehber sayfası OSÇ

5.1.1.5 Mesajlarda Arama Yapma Sayfasının Modellenmesi

Telegram uygulamasında mesajlarda arama yapabilmek için anasayfada bulunan büyüteç ikonuna (üst katmanda S ile gösterilen) tıklanması gerekmektedir. Arama yapma sayfasıyla ilgili aksiyon alındıktan sonra Şekil 5.23'te görülen ekran gelmektedir. Bu kısımında kullanıcıya yerel mesajlarda, kişilerde veya dünya genelindeki kişiler, kanallar arasından arama sonuçları verilmektedir. Tez kapsamında bu kısmın modellemesi bir kişi arama ile sınırlandırılmıştır. Şekil 5.24'te rehberden olan ve olmayan kişiler için sonuç ekranları görülmektedir.

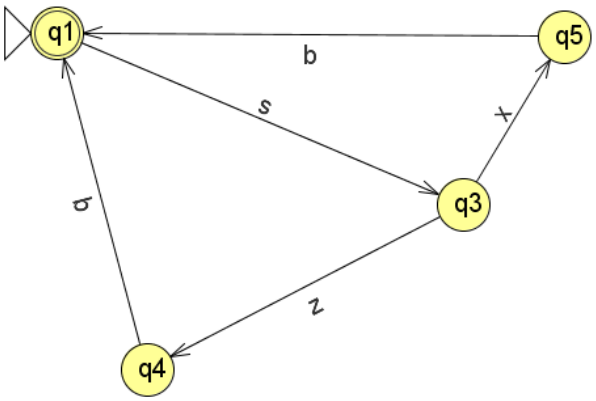


Şekil 5.23. 4: Anasayfada arama yapma arayüzü



Şekil 5.24. 4: Rehberde olan ve olmayan kişi arama sonuçları arayüzü

Anasayfada yer alan arama yapma alanına ait SDM aşağıdaki gibi olup 4 numara ile gösterilecektir.



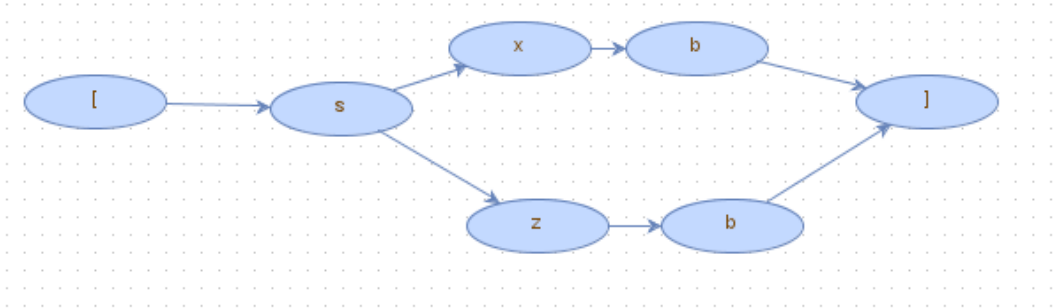
Şekil 5.25. 4: Anasayfa arama SDM (alt katman)

Bu SDM üzerinde kullanılan küçük harfler alt katmanda yapılabilecek olan geçişleri göstermektedir. Aşağıdaki Tablo 5.7'de SDM'ye ait harflerin anlamı yer almaktadır.

Tablo 5.7. 4 Numaralı Modelin Harf Gösterim ve Anlam Tablosu

Harf Gösterimi	Anlamı
b	Geriye tıklama
s	Seçeneklere tıklama
x	Listede olmayan bir metni arama
z	Listede olan bir metni arama

Aşağıdaki Şekil 5.26'da Telegram uygulamasının 4 modeline ait olan OSÇ çizimi yer almaktadır.



Şekil 5.26. 4: Anasayfa arama OSÇ

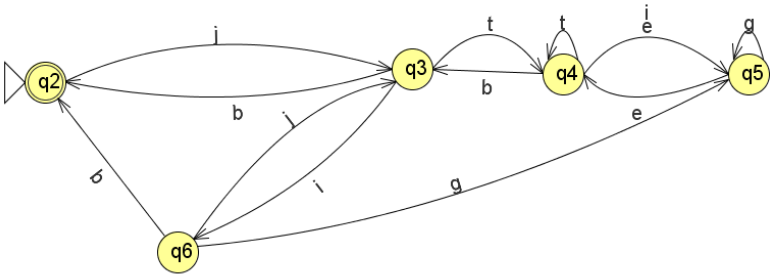
5.1.2 Mutantlar

Mutantların yaratılması için orijinal model üzerinde değişiklik yapılır. Orijinal modelde yer almayan bir sayfa geçişinin bu modele eklenmesi mutantı oluşturur. Model tabanlı mutasyonlar ise mutant modellerin oluşumunu sağlar.

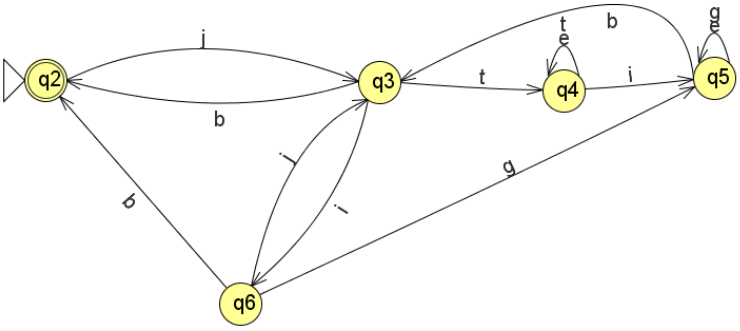
İleri sürülen yöntem için model tabanlı mutantların yanı sıra kod tabanlı mutantlar gereklidir. Üretilen kod tabanlı mutant model tabanlı mutantları meydana getirmektedir. Kod tabanlı mutasyon işlemi hatasız kod üzerine hatalı kod enjekte edilerek yapılır. İşlem sonucunda mutant bir uygulama ortaya çıkar. Model tabanlı hata modeline kıyasla enjekte edilen hatalı kod birinci veya daha yüksek dereceden mutasyonları oluşturabilir.

Bu aşamada kullanılacak olan mutantlar aşağıdaki gibidir;

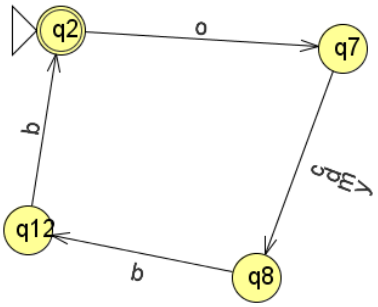
- Uygulamanın mesaj gönder butonuna tıklandığında emoji seçme ekranın açılması (1FB1 ile gösterilecektir, Bkz. Şekil 5.27 (1))
- Uygulamanın gönder butonuna tıklandığında, uygulamanın tepki vermemesi (1FB2 ile gösterilecektir, Bkz. Şekil 5.28 (2))
- Uygulamanın mesajlaşma sayfasında yer alan metin arama, geçmişi temizleme, mesajı silme ve gruptan çıkma isimli seçeneklerinin isimlerinin doğru bırakılıp fonksiyonaliteleri bozularak hepsinin mesajlaşma içinde metin arama yapması sağlanmıştır. (1FA ile gösterilecektir, Bkz. Şekil 5.29 (3))



Şekil 5.27. Mutantlar (1) 1FB1: Mesaj gönderme işlevini yerine getirmeyip emoji açan hatalı SDM



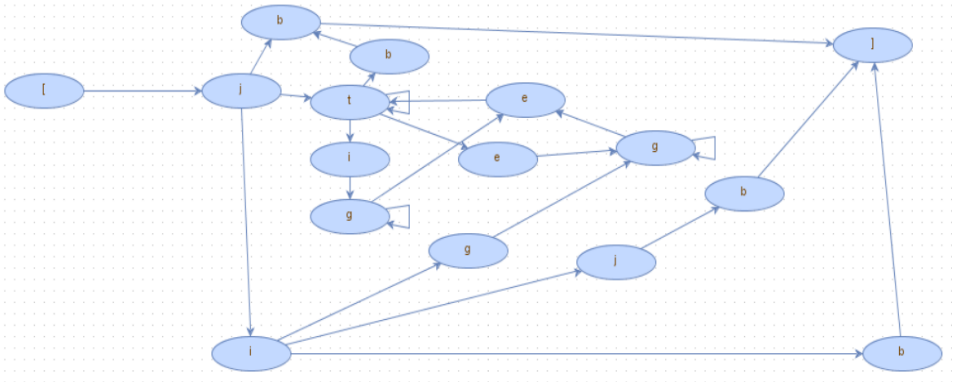
Şekil 5.28. Mutantlar (2) 1FB2: Mesaj gönderme işlevini yerine getirmeyip tepki vermeyen hatalı SDM



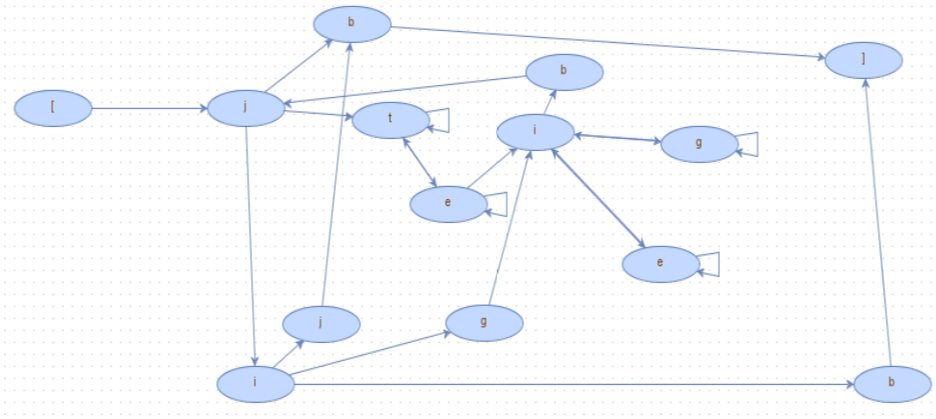
Şekil 5.29. Mutantlar (3) 1FA: Tüm seçenekleri metin araması yapan hatalı SDM

Aynı mutant modelleri kıyaslayabilmek için olay sıra çizgeleri (OSÇ) ve GraphWalker ile çizdiğimiz zaman oluşturulan mutantlar aşağıda verilmiştir;

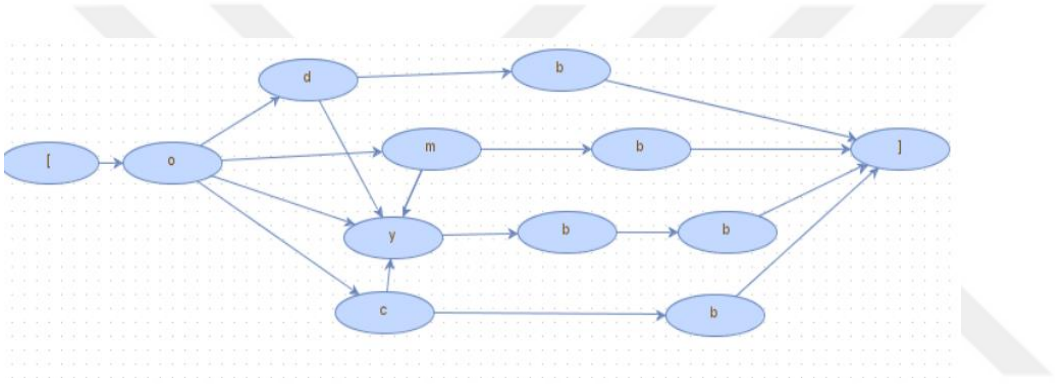
- Uygulamanın gönder butonuna tıklandığında mesaj gönderme işlemini yapmayıp emoji seçme ekranın açılması (1FB1 ile gösterilecektir, Bkz. Şekil 5.30 (1), Şekil 5.33 (1))
- Uygulamanın gönder butonuna tıklandığında uygulamanın tepki vermemesi (1FB2 ile gösterilecektir, Bkz. Şekil 5.31 (2), Şekil 5.34 (2))
- Uygulamanın mesajlaşma sayfasında yer alan metin arama, geçmişi temizleme, mesajı silme ve gruptan çıkma isimli seçeneklerinin isimleri doğru bırakılmış fakat fonksiyonaliteleri bozularak hepsinin mesajlaşma içinde metin arama yapması sağlanmıştır. (1FA ile gösterilecektir, Bkz. Şekil 5.32 (3), Şekil 5.35 (3))



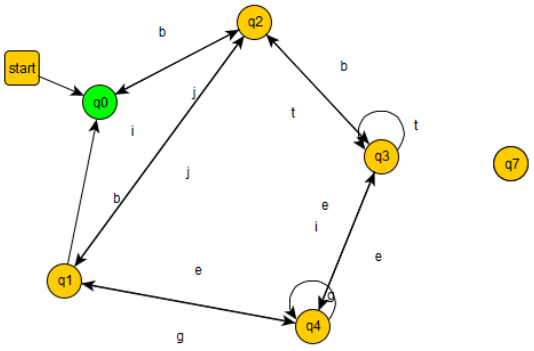
Şekil 5.30. Mutantlar (1) 1FB1: Mesaj gönderme işlevini yerine getirmeyip emoji açan hatalı OSÇ



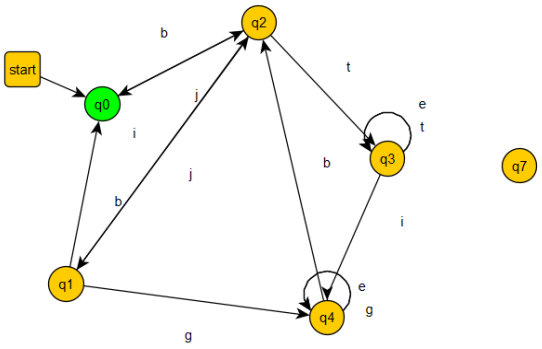
Şekil 5.31. Mutantlar (2) 1FB2: Mesaj gönderme işlevini yerine getirmeyip tepki vermeyen hatalı OSÇ



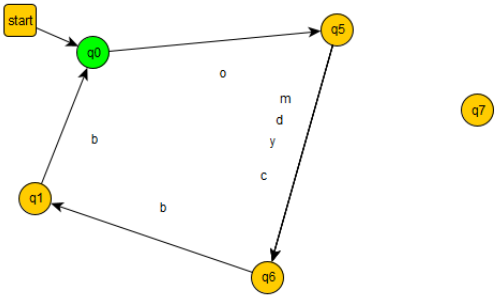
Şekil 5.32. Mutantlar (3) 1FA: Tüm seçenekleri metin araması yapan hatalı OSÇ



Şekil 5.33. Mutantlar (1) 1FB1: Mesaj gönderme işlevini yerine getirmeyip emoji açan hatalı GraphWalker modeli



Şekil 5.34. Mutantlar (2) 1FB2: Mesaj gönderme işlevini yerine getirmeyip tepki vermeyen hatalı GraphWalker modeli



Şekil 5.35. Mutantlar (3) 1FA: Tüm seçenekleri metin araması yapan hatalı GraphWalker modeli

5.1.3 Dönüşümler

Kapsam SDM'ler temel alınarak bunlara karşılık gelen Dİ'ler üretilecektir. JFLAP vasıtası ile SDM'lerden Dİ'lere dönüşümler gerçekleştirilir. Daha önce ölçeklenebilme probleminden dolayı katmanlı bir SDM modeliyle çalıştığımızdan bahsetmiştik. Bu kısımda üst katmanda SDM'den üretilen Dİ'lerin ve alt katman SDM'lerden gelen Dİ'lerden ayrı ayrı üretilecektir. Tezin bu kısmında daha önce modelleri anlatılan SDM'lere ait Dİ'ler verilecektir.

Telegram uygulaması **ana sayfasına** ait (üst katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $A(uOb + uJb + Sb + Wb + Pb + ub) *$

Telegram uygulamasının **mesajlaşma sayfası (1A) modeline** ait (alt katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $(oybb + omb + odb + ocb) *$

Telegram uygulamasının **mesajlaşma sayfası (1B) modeline** ait (alt katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $((jt(et + t) * ig * e + jt(et + t) * ei + ji)((jt(et + t) * i + g)g * e + jt(et + t) * ei + ji) * (jt(et + t) * eb + jb + b) + jt(et + t) * eb + jb) *$

Telegram uygulamasının **yeni bir konuşma oluşturma sayfası (2) modeline** ait (alt katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $p(q(lvz(bvz) * bbb + lbb) * (lvz(bvz) * bvx + lvx)((bvz(bvz) * bbb + bbb)(lvz(bvz) * bbb + lbb) * (lvz(bvz) * bvx + lvx) + bvz(bvz) * bvx + bvx) * (bvz(bvz) * bbb + bbb)(lvz(bvz) * bbb + lbb) * b + q(lvz(bvz) * bbb + lbb) * b + nxb + nzb + nb) *$

Telegram uygulamasının **rehber sayfası (3) modeline** ait (alt katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $((wrnz(bnz) * bnx + wrnx)(bnz(bnz) * bnx + bnx) * (bnz(bnz) * bb + bb) + wrnz(bnz) * bb + wrb) *$

Telegram uygulamasının **mesajlarda arama yapma sayfası (4) modeline** ait (alt katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $(sxb + szb) *$

Telegram uygulamasının **hatalı mesajlaşma sayfası (1FB1) modeline** ait (alt katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $((jt(bt+t)*(e+i)(e(bt+t)*(e+i)+g)*e(bt+t)*bi+jt(bt+t)*bi+ji)((jt(bt+t)*(e+i)+g)(e(bt+t)*(e+i)+g)*e(bt+t)*bi+jt(bt+t)*bi+ji)*((jt(bt+t)*(e+i)+g)(e(bt+t)*(e+i)+g)*e(bt+t)*bb+jt(bt+t)*bb+jb+b)+jt(bt+t)*(e+i)(e(bt+t)*(e+i)+g)*e(bt+t)*bb+jt(bt+t)*bb+jb)*$

Telegram uygulamasının **hatalı mesajlaşma sayfası (1FB2) modeline** ait (alt katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $((jt(e+t)*i(bt(e+t)*i+e+g)*bi+ji)((jt(e+t)*i+g)(bt(e+t)*i+e+g)*bi+ji)*((jt(e+t)*i+g)(bt(e+t)*i+e+g)*bb+jb+b)+jt(e+t)*i(bt(e+t)*i+e+g)*bb+jb)*$

Telegram uygulamasının **hatalı mesajlaşma sayfası (1FA) modeline** ait (alt katman) SDM'den üretilen düzenli ifade aşağıdaki gibidir.

- $(o(c+d+m+y)bb)*$

5.2 Test Üretimi

PQ-Analysis aracı ile normal ve mutant modellerinin düzenli ifadeleri kullanılarak ileri sağ ve ileri sol bağlam tabloları elde edilir. Bu aracın çıktısı PQTestGen aracına verilerek optimal test dizileri elde edilir. Katmanlı bir SDM yapısı kullandığımızdan dolayı kullanılacak olan test dizilerinin birleştirme işlemlerinin yapılması gerekmektedir. Bu birleştirme sırasında yer değiştirme (substitution) metodu kullanılacaktır. Üst katmandan gelen test dizilerinde yer alan semboller (J, P, S, R, O) ilgili alt katmanda gelen test dizileri ile yer değiştirecektir.

Üst katmandan ve alt katmandan gelen test dizileri ile örnek bir test dizisi üretimi aşağıdaki gibi olmaktadır. Bu örnekte üst katmadan gelen sembol S'tir. Alt katmanda gelen test dizileri, üst katmanda S olan yerler ile yer değiştirir. Bu yer değişimi sonrasında sonuç test dizilerine ulaşılır.

Tablo 5.8. Yer değiştirerek test dizisi oluşturma örneği

Üst katmadan gelen test dizileri	**Alt katmadan gelen test dizileri**	**Sonuç Test Dizileri**
A	szb	A
ASbub	sxb	Aszbbub
Aub		Asxbbub
Aubub		Aub
ASbSb		Aubub
ASb		Aszbbszbb
AubSb		Asxbbsxbb
		Aszbb
		Asxbb
		Aubszbb
		Aubsxbb

Yukarıda verilen örneğe uygun olacak diğer modeline ait optimal test dizileri de üretilmiştir. Aşağıdaki tabloda test dizilerinin model isimleri, test dizisi sayıları, toplam karakter sayıları ve mutant olup olmadıkları verilmiştir. Test dizilerinin detayları Ek 2'de yer almaktadır.

Tablo 5.9. Modellerin (IRE) test dizisi sayıları

Model Adı	**Test Dizisi Sayısı**	**Toplam karakter sayıları**	**Mutant Model**
1B	77	759	-
1FB1	111	1185	X
1FB2	117	1281	X
1A	11	67	-
1FA	11	73	X
2	47	587	-
3	31	425	-
4	11	65	-

Aşağıdaki tabloda ise OSÇ'ye ait test dizilerinin model isimleri, test dizisi sayıları, toplam karakter sayıları ve mutant olup olmadıkları verilmiştir.

Tablo 5.10. Modellerin (OSÇ) test dizisi sayıları

Model Adı	Test Dizisi Sayısı	Toplam Karakter Sayısı	Mutant Model
Orijinal Model	22	145	-
1FB1	22	151	X
1FB2	20	142	X
1FA	24	174	X

Orijinal modelin (hatasız) TSD aracıyla OSÇ çizimi yapılırken katmanlı bir yapı kullanılmıştır. TSD aracı 1, 2, 3 ve 4 numaralı modellerin hepsinin tek seferde çizilebilmesine imkân tanımıştır. Mutant modeller için de OSÇ çizimi sırasında katmanlı bir yapı kullanılmıştır. Her bir mutant (1FA, 1FB1, 1FB2) için ayrı birer model çizilmiştir. Orijinal modele ve hatalı modellere ait test dizilerinin detayları Ek 3'te yer almaktadır.

Aşağıdaki tabloda ise GraphWalker aracıyla rastgele üretilen test dizilerinin model isimleri, test dizisi sayıları, toplam karakter sayıları ve mutant olup olmadıkları verilmiştir. Test dizilerinin detayları Ek 4'te yer almaktadır.

Tablo 5.11. Modellerin (GraphWalker) test dizisi sayıları

Model Adı	Test Dizisi Sayısı	Toplam Karakter Sayısı	Mutant Model
1 A	10	52	-
1B	13	73	-
1FB1	8	59	X
1FB2	4	74	X
1FA	10	58	X

Her bir model için GraphWalker aracıyla 10 defa rastgele test dizisi üretilmiştir. Testler sırasında kullanılacak olan test dizisine ise üretilen test dizilerinin karakter uzunluklarının ortalaması alınarak karar verilmiştir. Aşağıdaki

tabloda modellere göre her üretim sonrasında elde edilen sonuçlar ve ortalamaya yakın olduğu için seçilen test dizisinin üretim sırası gösterilmektedir.

Tablo 5.12. GraphWalker ile rastgele üretilen test dizilerinin uzunlukları

Model Adı / Üretim	1A	1B	1FA	1FB1	1FB2
1. Üretim	34	182	126	**88**	162
2. Üretim	76	144	30	80	86
3. Üretim	104	88	110	136	328
4. Üretim	82	78	**78**	64	78
5. Üretim	74	**96**	54	84	86
6. Üretim	**66**	82	38	110	76
7. Üretim	40	122	38	56	**134**
8. Üretim	70	46	118	**88**	188
9. Üretim	60	56	46	112	68
10. Üretim	64	66	102	80	238
Ortalama	67	96	74	89,8	144,4

5.3 Test Sıkıştırması

PQTestGen aracıyla her bir model için üretilen test dizilerinin içerisinden birbirini kapsayan ya da tekrar eden test dizileri çıkartılarak test sıkıştırma (test compaction) işlemi yapılmıştır. Bu işlem sonrasında birbirini kapsamayan ve en uzun olan test dizileri elde edilmiştir. Bu sıkıştırma işlemi için aşağıdaki şekilde yer alan java kodu geliştirilmiştir.

```
public static void main(String[] args) throws IOException {
    Reader r = new Reader("testCases1FB2.txt");
    ArrayList<String> list=r.getLines();
    getUniqueSequenceFromFile(list);
}
  public static ArrayList<String> getLongestSubsequences(ArrayList<String> list)
  {
      ArrayList<String> newlist=new ArrayList<String>();
      int counter = -1;
      int temp = 0;
      int index = -1;
      int occurence = 0;
      for(int i = 0; i < list.size(); i++) // substring index list
      {
          counter = 0;
          occurence = 0;
          for (int j = 0 ; j < list.size(); j++ ) // all strings list
          {
              if (list.get(j).startsWith(list.get(i)) && list.get(j).length() >= list.get(i).length())
              {
                  occurence++;
                  System.out.println(list.get(j) +" startwith=? " + list.get(i));
                  temp = list.get(j).length();
                  System.out.println(temp +" >=? " + counter);

                  if(list.get(j) != null)
                  {
                      if (temp >= counter)
                      {
                          counter = temp;
                          index = j;
                          temp = 0;
                      }
                  }
              }
          }
          System.out.println("occurence of "+list.get(i)+": "+occurence);
          System.out.println("index "+index+" "+list.get(index) +" added ");
          newlist.add(list.get(index));
      }
      System.out.println("All Cases:" + list);

      return newlist;
  }
 public static ArrayList<String> getUniqueSequences(ArrayList<String> newlist)
 {
     ArrayList<String>
     removeDupList = removeDuplicates(newlist);
     System.out.println(removeDupList.toString());
     return removeDupList;
 }

 public static void getUniqueSequenceFromFile(ArrayList<String> list)
 {
     ArrayList<String> newlist=new ArrayList<String>(getLongestSubsequences(list));
     newlist = getUniqueSequences(newlist);
     System.out.println("Unique Cases:"+newlist);
     System.out.println("Number of All Cases:"+list.size());
     System.out.println("Number of Unique Cases:"+newlist.size());
 }

 public static <T> ArrayList<T> removeDuplicates(ArrayList<T> list)
 {
     ArrayList<T> newList = new ArrayList<T>();
     for (T element : list) {
         if (!newList.contains(element)) {
             newList.add(element);
         }
     }
     return newList;
 }
```

Şekil 5.36. Test sıkıştırma işlemi için geliştirilen Java kodu

Test sıkıştırma işlemini bir örnekle açıklayabiliriz. Bir test kümesinin “A”, “Aub”, “Aubb”, “Aubub”, “Aubbu” ve “Aubbb” test dizilerinden oluştuğunu

varsayalım. Bu test dizileri içerisinde yer alan bazı durumlar tekrar etmektedir. “A”, “Aub” ve “Aubb” test dizileri; “Aubbu” ve “Aubbb” tarafından kapsanmaktadır. Dolayısıyla bu 3 test dizisini “Aubbu” ya da “Aubbb” test dizisiyle test ederek bir sıkıştırma uygulayabiliriz. Sonuç olarak sıkıştırma işlemi sonrasında; “Aubub”, “Aubbu” ve “Aubbb” şeklinde 3 ayrı test dizisi elde edebiliriz. PQTestGen aracıyla elde edilen test dizilerine sıkıştırma işlemi uygulandıktan sonrasında ortaya çıkan test dizileri Ek 5’teki gibidir.

5.4 Test Etme

PQTestGen, TSD ve GraphWalker araçlarıyla üretilen test kümeleri Telegram uygulaması üzerinde yazılan test aracı ile koşulur ve elde edilen sonuçlar toplanır. Bu sonuçlar başarılı, başarısız test senaryolarını, toplam test zamanını ve toplam karakter boyutunu içermektedir.

İdeal test kümelerinin oluşturulma aşamasında pozitif test ve negatif test için de başarısız test kümeleri kullanılır. Pozitif test için hatasız modelden oluşturulan test dizileri mutant uygulama (kod tabanlı mutasyon ile sağlanmıştır.) üzerinde koşulmuştur. Negatif test ise mutant modelden oluşturulan test dizileri orijinal uygulama üzerinde koşulmuştur. OSÇ ve GraphWalker ile rastgele üretilen test dizileriyle de aynı şekilde test dizileri pozitif ve negatif test yöntemleri kullanılır.

5.5 Test Seçimi

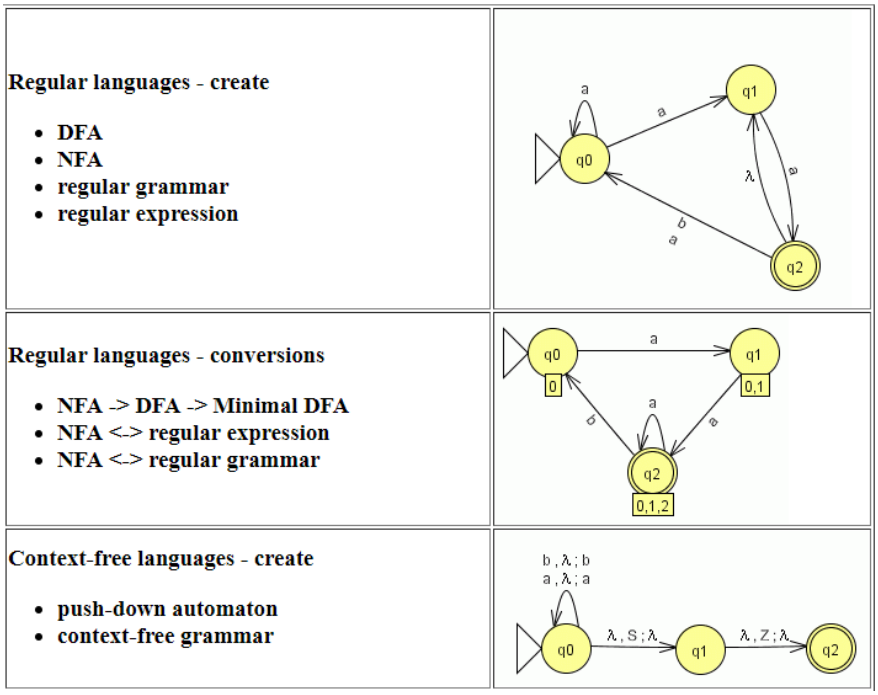
Test seçimi için Şekil 4.4’te verilen kriterler kullanılır. Bu kriterleri sağlayarak seçilen test dizileri, orijinal modelden elde edilen test dizilerinin mutant uygulama üzerinde ve mutant modelden elde edilen dizilerin ise orijinal uygulama üzerinde koşulması sonucu ortaya çıkan başarısız (test failed) test dizileridir. Seçilen test dizileri ile pozitif ve negatif test kümeleri oluşturulmuş olunur.

6. KULLANILAN ARAÇLAR

6.1 JFLAP

JFLAP, 1990 yılında Rensselaer Polytecnic Enstitüsünde Susan Rodger, önderliğinde çalışan öğrenciler tarafından otomatlar için yazılmıştır. Rodger'in Duke Üniversitesine geçmesiyle birlikte bu üniversite bünyesinde JFLAP projesi devam etmiştir. Java ile yazılan kaynak kodunun çoğu açıktır, bu kod kullanılarak farklı araçlarla entegrasyonlar sağlanabilir.

JFLAP, formal diller ve otomata teorinin temel yapısının öğrenilmesine yardımcı olan ve görselleştirme aşamasında grafiksel araçlar sunan bir programdır (JFLAP, 2018). Bu program ile düzenli gramer, düzenli ifadeler, DFA (Deterministik Sonlu Otomatlar) ve NFA (Deterministik Olmayan Sonlu Otomatlar) oluşturulabilir. NFA'den, DFA'e ve minimal DFA'e dönüşümler yapılabilir. Aynı zamanda context-free language de oluşturulabilir. Aşağıdaki verilen şekilde JFLAP ile yapılabilen işlemler ve görsel şekilleri özetlenmiştir.



Şekil 6.1. JFLAP aracı özellikleri (JFLAP, 2018)

Sonlu durumları çizmeye imkan sağlayan arayüzü oldukça basit ve kullanışlıdır. Arayüz üzerinde bulunan araç çubuğunda yapılan temel işlemler aşağıdaki gibidir.

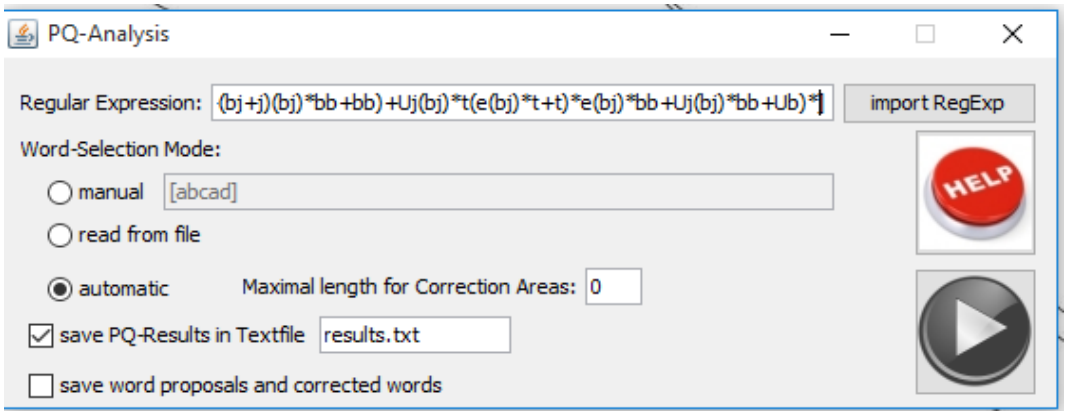
- Attribute Editor Tool: Özellik düzenleme aracı yardımıyla başlangıç ve son durumlar belirtilebilir.
- State Creator Tool: Durum oluşturma aracı sayesinde ardışık bir şekilde durumlar oluşturulabilir.
- Transition Creator Tool: Geçiş oluşturma aracı ile durumlar arasındaki geçişlerin oluşturulması sağlanır. Bu geçişlere isim verilebilir.
- Deletor Tool: Silme aracı ile silinmek istenen geçişler ve durumlar silinebilir.

JFLAP, kâr amacı gütmeyen, telif hakları konusunda esneklik ve paylaşımı yaygınlaştırmayı sağlayan Creative Commons (CC) lisansıyla tescillenmiştir.

6.2 PQ Analysis ve PQTestGen

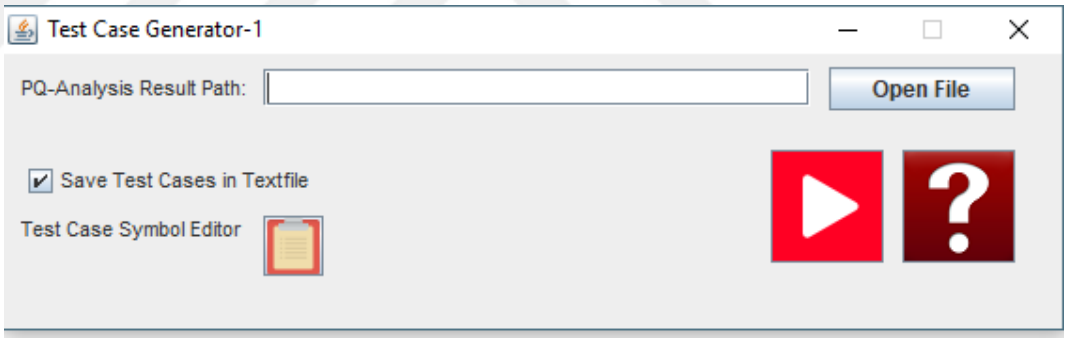
PQ-Analysis aracı, modellere ait düzenli ifadeleri kullanılarak ileri sağ ve ileri sol bağlam tabloları (Belli, 2015) elde edilmesine ve dosyaya kaydedilmesine yardımcı olur. PQ Analysis aracının arayüzü Şekil 6.2'de gösterildiği gibidir. Düzenli ifade yazılırken, başına açılış için "[" sembolü, sonuna ise kapanış için "]" sembolleri eklenir. Düzenli ifade dışarıdan eklenmek istenirse, V-File, BES-file, FSM-file ve RegEx-File olarak da "import RegExp" seçeneği ile yüklenebilir (PQ-Analysis, 2018).

Kelime seçme modu ile (Word-Selection Mode) istenirse, düzenli ifade içerisinde geçen spesifik bir kelimenin özellikleri kontrol edilebilir. Bu seçenek otomatik olarak bırakılırsa, bütün olası kelimeler üretiliyor olacaktır.



Şekil 6.2. PQ-Analysis aracı arayüzü

PQ-TestGen aracı ise optimal test dizileri için elde edilen bağlam tablolarını girdi olarak kullanarak test senaryolarının dosyaya kaydedilmesine yardımcı olur (Kilincceker et. al., 2018). Bu sırada aynı olan test senaryoları da elimine edilmiş olur. PQ-TestGen aracının arayüzü aşağıdaki gibidir.



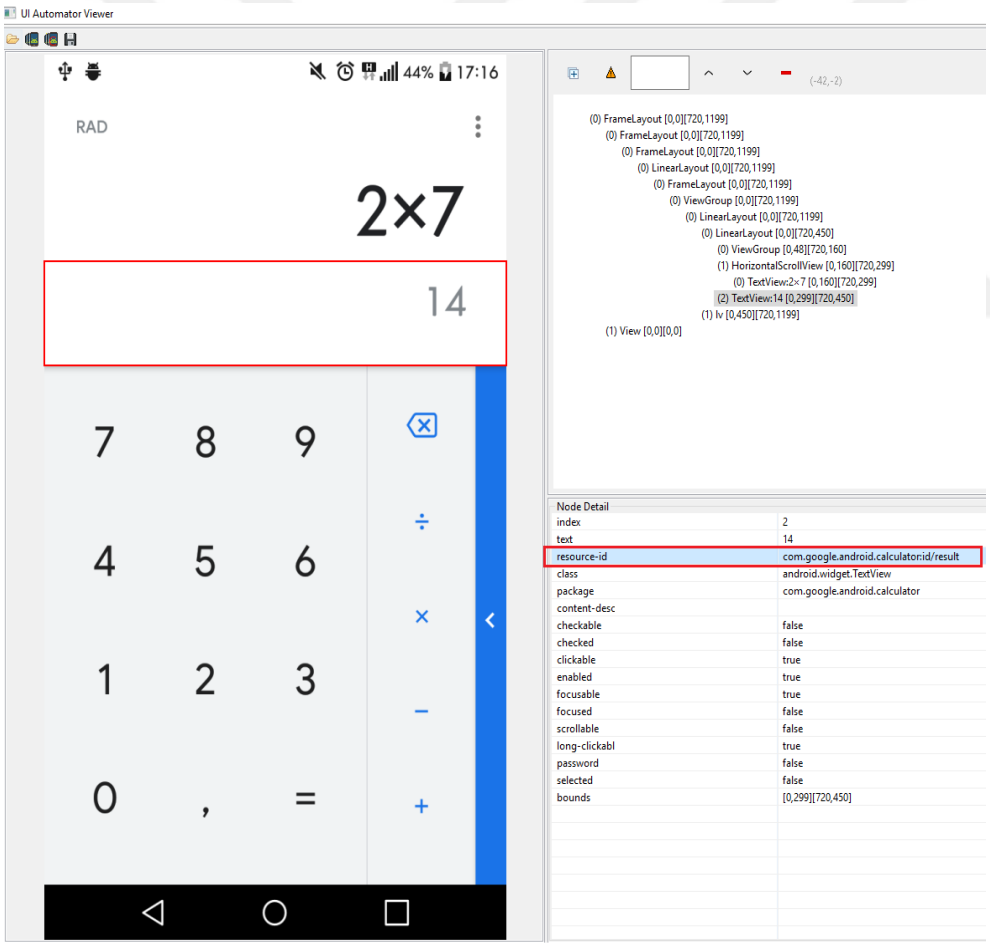
Şekil 6.3. PQ-TestGen aracı arayüzü

6.3 UI Automator Viewer

Android UI Automater Viewer Tool'u ise tersine mühendislik (Reverse Engineering) için kullanılır. Bu sayede GUI üzerindeki UI component'leri bulunmuş olur. Kara kutu testlerinin yazımı için idealdir. Android geliştirme araçları (SDK) içersinde bulunmaktadır. Kullanabilmek için aşağıdaki yolda yer alan uiautomatorviewer isimli .bat dosyasının çalıştırılması gerekmektedir. Sonrasında açılan UI Automator Viewer üzerinden uygulama ait bir ekran görüntüsü alınmaktadır.

Uygulama yolu: c:\users\<username>\AppData\Local\Android\sdk\tools

Aşağıdaki örnekte bir hesap makinesi uygulamasının arayüzü gösterilmektedir. Sonucun bulunduğu kısımda yer alan 14 sayısı bir text view bileşenin içinde yer almaktadır. Hangi bileşenin içinde yer aldığı sağ tarafta yer alan arayüz bileşen ağacından görülebilir. Bu ağaç üzerinde tıklanan bileşene ait olan uygulama bileşeni de anlık olarak sol tarafta yer alan ekran üzerinde işaretlenmekte ve göze çarpmaktadır. Bileşenin detayları ise sağ alt köşede Node Detail sekmesinde gösterilmektedir. Bu sayede arayüz üzerinden tersine mühendislik yapılmasına imkân sağlamaktadır.



Şekil 6.4. Android UI Automator Viewer aracı

6.4 Android UI Test Framework

Kullanıcı arayüz testleri manuel olarak bir tester tarafından da yapılabilir. Fakat bu manuel test işlemi oldukça zaman alıcı, sıkıcı ve hataya açık olabilir. Dolayısıyla daha etkili bir yaklaşışmla bu testlerin otomatik olarak yapılması hem zamandan tasarrufu hem de testlerin güvenirliğini arttırmaktadır. Bu sayede testler hızlı bir şekilde tekrarlanarak sonuçlar elde edilir.

Android Studio ile kullanıcı arayüzü testlerini otomatikleştirmek için test kodu ayrı bir Android test klasöründe (src/androidTest/java) yazılır. Gradle ile test koduna uygun bir şekilde test uygulaması oluşturur, ardından test uygulamasını hedef uygulama ile aynı cihazda yükler. Test uygulamasında, belirli kullanım senaryolarını kapsayan test arayüz test framework'leri kullanılarak hedef uygulamadaki kullanıcı etkileşimleri simüle edilir.

Özellikle birim testlerin yazımında kullanılır, aşağıda örnek bir kullanımı görebilirsiniz. *assertEquals* fonskyionunun birinci parametresi beklenen, ikincisi ise girdi olarak verilen değerdir. Sonucu başarılı ya da başarısız şeklinde elde edilir.

```
@Test
public void check_multiplication () throws Exception {
    assertEquals(4, 2 * 2);
}
```

Gerçek cihaz üzerinde yapılacak olan UI işlemlerinin simülasyonu da aşağıdaki şekilde sağlanır. Öncelikle cihaz nesnesi üzerinde yapılacak olan işlemler yaptırılır. Sonrasında ekrana çıkması beklenen arayüz bileşenleri kontrol edilir.

```
mDevice.click(624,1103);
SystemClock.sleep(5000);
mDevice.pressBack();
SystemClock.sleep(5000);
mDevice.findObject(new
UiSelector().className("android.view.View").index(0)).click();
SystemClock.sleep(5000);
mDevice.findObject(new
```

```
UiSelector().className("android.widget.EditText").index(1)).setTex
t("Ben iyiyim");
```

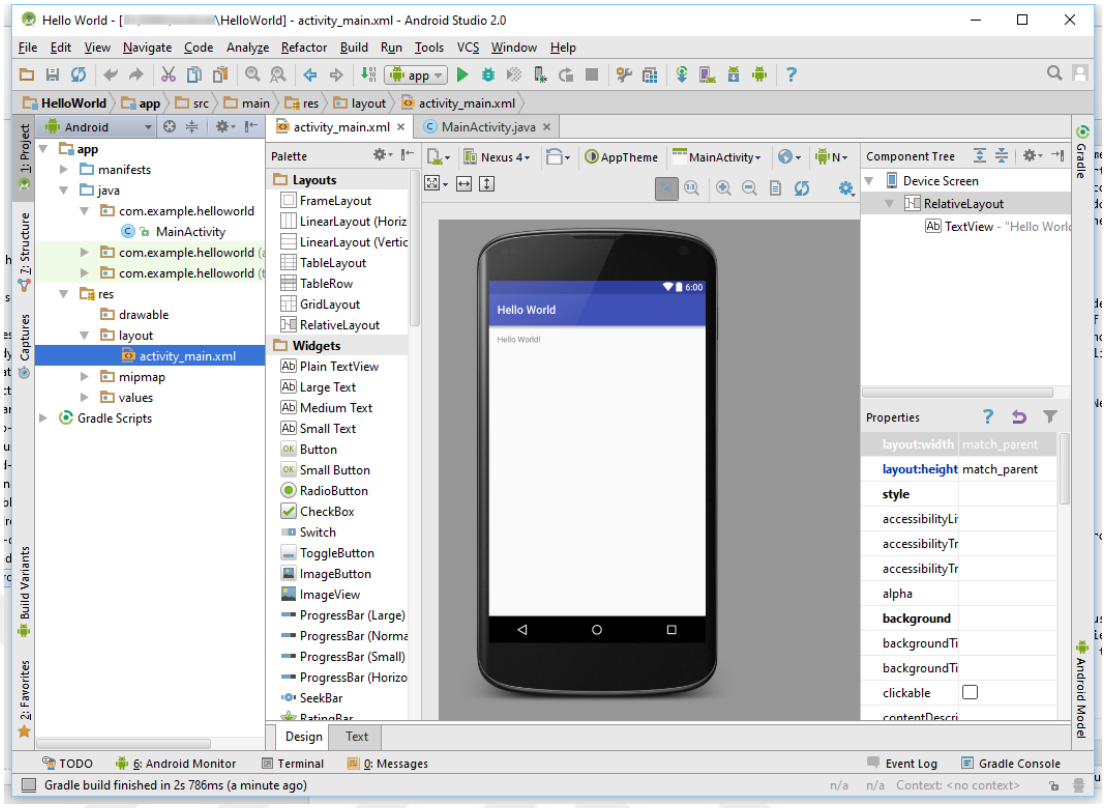
Geliştirme ortamı hazırlanırken Android Studio'da kullanılan Gradle build sistemine aşağıdaki bağımlılıkların (dependencies) eklenmesi gerekmektedir. Android UI Automator kütüphanesi Android 4.3 (API level 18) veya daha yüksek olan uygulamlarda çalışmaktadır.

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])

androidTestCompile('com.android.support.test.espresso:espresso-
core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-
annotations'
    }
    )
    compile 'com.android.support:appcompat-v7:25.3.1'
    testCompile 'junit:junit:4.12'
    compile 'com.android.support.test:testing-support-lib:0.1'
    compile 'com.android.support.test.uiautomator:uiautomator-
v18:2.0.0'
}
```

6.4 Android Studio IDE

Bu Android Studio, Android için tümleşik geliştirme ortamı (IDE) sağlar. 2013 yılında Google I/O etkinliğinde tanıtılmıştır. Daha öncesinde Android ile ilgili geliştirmeler eclipse geliştirme ortamı ile sağlanmaktaydı. Android Studio, IntelliJ IDEA (JetBrains, 2018) temel alınarak Android için özel olarak oluşturulmuştur. Java programlama dili ile yazılmıştır. Cross-platform çalışma imkânı sağlar. Apache 2.0 lisansı ile lisanslanmış olup ücretsiz olarak edinilir. Şekil 6.5'te görülebileceği gibi, arayüzü geliştiriciler için zengin bir içeriğe sahiptir.



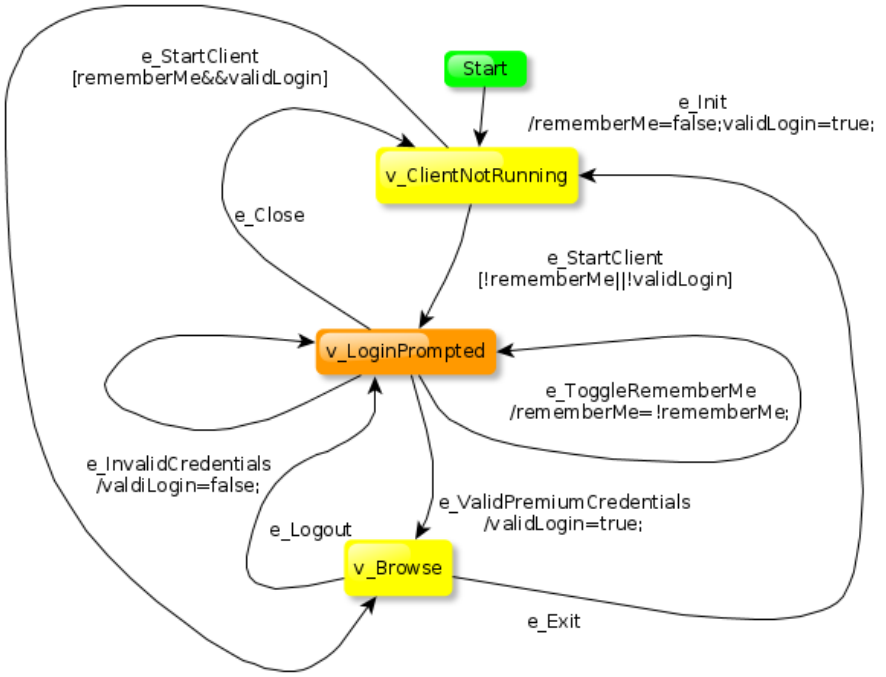
Şekil 6.5. Android Studio arayüzü

Geliştiricilere sağladığı temel özellikler ise şunlardır;

- Farklı özellik ve görünümlere göre birden fazla uygulama çıkartabilme.
- Hızlı ve zengin özelliklere sahip bir emulator aracı
- Gradle tabanlı esnek bir proje inşa sistemi
- Genel uygulama özelliklerini oluşturmaya yardımcı olan kod şablonları ve GitHub entegrasyonu
- Google hizmetlerinin çok fazla uğraş gerektirmeden uygulamaya eklenmesi
- C++ ve NDK desteğinin sağlanması
- Ekran tasarımları için zengin içeriğe sahip sürekle-bırak editörleri
- Uygulamanın performansını, kullanılabilirliğini, farklı sürümlerde çalışabilirliğini kontrol eden test araçları ve kütüphanelerinin olmasıdır.

6.5 GraphWalker

GraphWalker, model tabanlı test yöntemini temel alan açık kaynak olarak geliştirilen bir test otomasyon aracıdır. MIT lisansı ile dağıtımına ve değiştirilmesine izin verilmektedir. Test mühendisleri yEd (yEd, 2019) aracıyla yaptıkları .graphml uzantılı model çizimlerini GraphWalker aracıyla test senaryolarına dönüştürebilirler. Test edilecek olan model yönlü bir çizge şeklindedir. Bu yönlü çizgeler takip edilerek matematiksel algoritmalar sayesinde geçilen yollar (paths) test senaryolarına dönüştürülür. Bu modeller Şekil 6.6'da görüldüğü gibi; durumları ve doğrulamaları belirtilerek çizilir (GraphWalker, 2019)



Şekil 6.6. GraphWalker aracında kullanılan model örneği (GraphWalker, 2019)

GraphWalker'ı kullanabilmek için öncesinde aşağıdaki programların yüklü olması gereklidir:

- Java JDK 7 ya da 8
- grapwalker-cli

- yEd çizim aracı
- Maven (Mevcut projenize eklemek isterseniz yüklü olmalı)

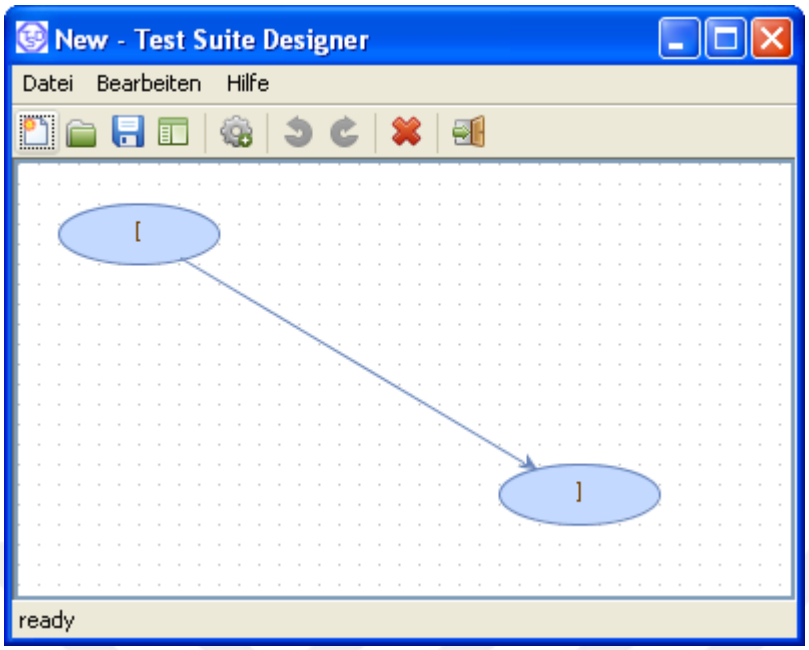
Model çizimi sonrasında ilgili test senaryolarının oluşumu aşağıdaki komut ile sağlanabilir. Login modeli kullanarak tüm yolların rastgele şekilde üretilmesi sağlanır. Kenar kapsamanın (edge coverage) 100 olarak tanımlanması çizge üzerinde tüm kenarlarını kapsayana kadar dolaşılması ve yol üretilmesi anlamına gelmektedir.

```
java -jar graphwalker.jar offline -m Login.graphml "random(edge_coverage
(100))"
```

6.6 Test Suite Designer (TSD)

TSD, yazılım analizine ve test araştırmalarına açık, ticari olmayan bir yazılımdır (Belli, 2014). Olay sıra çizgilerinin (OSÇ) çiziminde kullanılan etkin bir araçtır. Modelleme yapan kişilere Şekil 6.7'de görüldüğü gibi oldukça basit bir arayüz sunmaktadır. Grafiksel kullanıcı arayüzlerinde yer alan her bir olay, birbirini takip eden bir akış içerisinde çizilir. Başlangıç ve bitiş olaylarını belirlemek için "[" ve "]" sembolleri kullanılır. Hiyerarşik bir yapı içerisinde olayların ve alt olayların tanımlanmasına imkân sağlanır.

Araç üzerinde çizim yapıldıktan sonra düğüm kapsamaya (node coverage) göre optimal test dizileri otomatik olarak üretilmektedir. Aynı zamanda mutant modele ait test dizileri de otomatik olarak üretilir. Çinli postacı probleminin (Chinese Postman Problem) çözümü ile çizge üzerinde optimizasyon sağlanmaktadır.



Şekil 6.7. TSD aracı model arayüzü

7. SONUÇ VE TARTIŞMA

Bu çalışmada yazılım testi için kullanılan bütünsel ve mutasyon testi yaklaşımlarının birlikte kullanılması ile melez bir yaklaşım öne sürülmüştür. Bu melez yöntemin ayrıca Android uygulaması GKA'sı için bir ideal test olma özelliği taşıdığı tespit edilip bunun için gerekli adımlar tanımlanmıştır.

Ayrıca kullanılan yaklaşımın gerçekleştirilmesi için ön çalışmalar Telegram isimli Android uygulaması GKA'sı için yapılmıştır. Bu yaklaşımı değerlendirmek için model tabanlı test dizisi üretimine olanak sağlayan TSD, MaTeLo (2018) ve GraphWalker (2019) araçları kullanılmıştır. Rastgele test için üretilen test dizileri MaTeLo ve GraphWalker vasıtası ile üretilmiştir, ardından bu testler ilgili uygulamalar üzerinde koşulmuş ve sonuçlar toplanmıştır. Yapılan deneysel çalışmalara MaTeLo'nun deneme sürümü sona erdiği için GraphWalker aracıyla devam edilmiştir. Sonuçlar kıyaslanırken random testing için GraphWalker aracından alınan sonuçlar kullanılmıştır. Model tabanlı üretilen rastgele olmayan test senaryolarının kıyaslanması ise TSD'den ve PQTestGen'den üretilen sonuçlar ile sağlanmıştır.

Kullanılan yaklaşımın getirdiği avantajlar özetlenecek olunursa;

- Bütünsel ve mutasyon testinin özelliklerini birleştirmesi
- İdeal test sayesinde sistem içerisindeki hataların varlığının (presence) ve yokluğunun (absence) test edilebilmesi

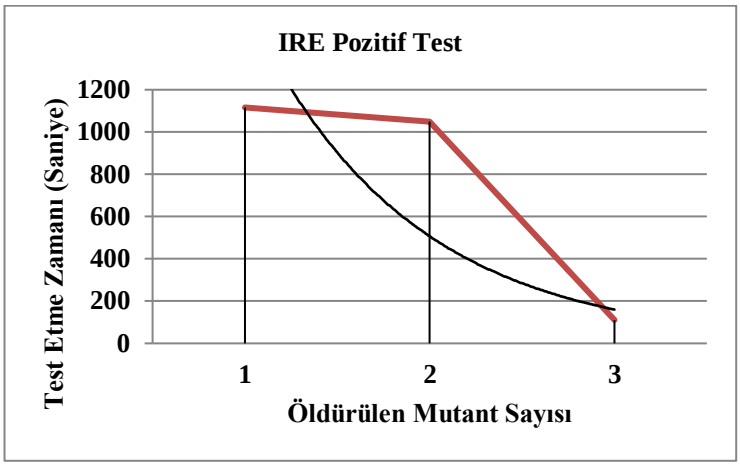
Android uygulaması GKA testi için öne sürülen yaklaşımın ilk aşamasında SDM modelin elde edilmesi adımı elle yapıldığı ve insan gücü, zamanı gerektirdiği için bu bir dezavantaj olarak sayılabilir. Ayrıca öne sürülen yöntem ölçeklenebilirlik (scalability) açısından bazı kısıtlar içermektedir. Bu yüzden modeli daha hiyerarşik olarak alt katmanlara ayırma yöntemi ile bu ölçeklenebilirlik sorununun üstesinden gelinmeye çalışılmıştır. Yalnız bu aşamada da yine elle yapıldığı için bir dezavantaj olarak sayılabilir. Elle yapılan SDM model oluşturulması ve bunun alt katmanlara ayrılması işleminin otomatik hale getirilmesi ileri ki çalışmalar olarak sayılabilir.

Pozitif test aşamasında orijinal modelden üretilen test senaryoları mutasyona uğratılmış 3 farklı mutant uygulama üzerinde test edilmiştir. Aşağıdaki tabloda pozitif test sonuçları görülmektedir. Beklenildiği gibi orijinal modele ait çoğu test senaryosu mutant uygulamalar üzerinde hata olarak saptanmıştır.

Tablo 7.1. IRE Pozitif Test Sonuç Tablosu

Pozitif Test	PQ Orijinal Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
1FA Modeline ait uygulama	5	2	3	1	1 dk 51 s	38
1FB1 Modeline ait uygulama	38	9	29	1	17 dk 30 s	417
1FB2 Modeline ait uygulama	38	9	29	1	18 dk 36 s	417
Hata Kapsama				(3/3)*100 = 100		

Pozitif test aşamasında elde edilen sonuçlar tüm mutantların öldürüldüğünü göstermektedir. Öldürülen mutant uygulamalara ait test koşum süreleri aşağıdaki şekilde gösterildiği gibidir.



Şekil 7.1. IRE Pozitif Test Süreleri

Negatif test aşamasında ise mutasyona uğratılmış 3 farklı mutant uygulama modeli üzerinden test senaryoları elde edilmiş ve her bir test kümesi orijinal uygulama üzerinde test edilmiştir. Aşağıdaki tablolarda (Tablo 7.2, 7.3 ve 7.4) negatif test sonuçları görülmektedir. Beklenildiği gibi bu tabloda da mutant modellere ait çoğu test senaryosu orijinal uygulama üzerinde hata olarak saptanmıştır. Bununla beraber toplam karakter boyutuna göre toplam test zamanı da artış göstermektedir.

Tablo 7.2. IRE Negatif Test Sonuç Tablosu (Mutant 1)

Negatif Test (Mutant 1)	1FB1 Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	55	21	34	1	25 dk, 57 s	647

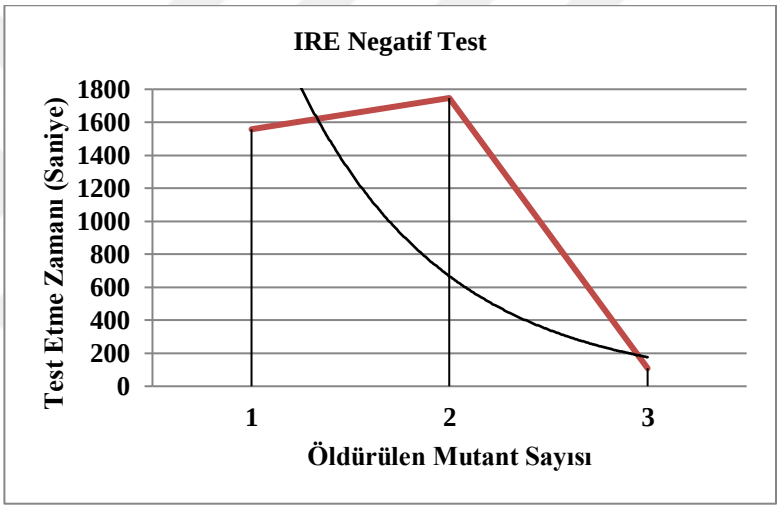
Tablo 7.3. IRE Negatif Test Sonuç Tablosu (Mutant 2)

Negatif Test (Mutant 2)	1FB2 Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	58	29	29	1	29 dk, 7 s	698

Tablo 7.4. IRE Negatif Test Sonuç Tablosu (Mutant 3)

Negative Test (Mutant 3)	1FA Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	5	2	3	1	1 dk, 49 s	41
Hata Kapsama				(3/3)*100 = 100		

Negatif test aşamasında elde edilen sonuçlar tüm mutantların öldürüldüğünü göstermektedir. Öldürülen mutant uygulamalara ait test koşum süreleri aşağıdaki şekilde gösterildiği gibidir.



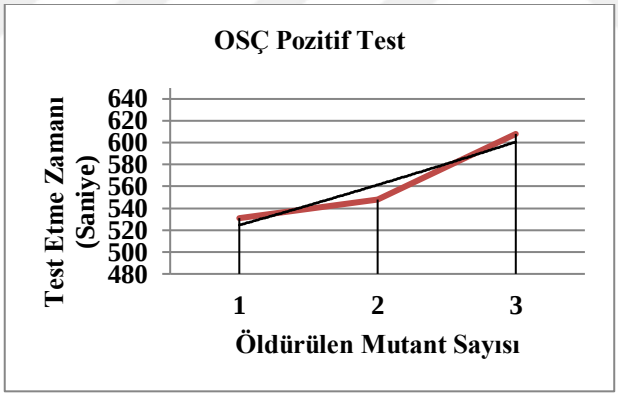
Şekil 7.2. IRE Negatif Test Süreleri

OSÇ'lerden üretilen test senaryoları için de pozitif test uygulanmıştır. Orijinal modele ait OSÇ'lerden üretilen 22 adet test dizisi bulunmaktadır. Bu test dizileri 3 farklı mutant uygulama üzerinde test edildiği zaman sonuç tablosu aşağıdaki gibi olmaktadır. Sonuç tablosu incelendiğinde IRE için yapılan pozitif teste göre, OSÇ'lerin test dizilerinin daha az olduğu fakat testi başarıyla tamamlayan test dizisi oranının daha fazla olduğu görülmektedir.

Tablo 7.5. OSÇ Pozitif Test Sonuç Tablosu

Pozitif Test (ESG)	ESG Orijinal Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
1FA Modeline ait uygulama	22	19	3	1	10 dk, 8 s	145
1FB1 Modeline ait uygulama	22	15	7	1	8 dk, 51 s	145
1FB2 Modeline ait uygulama	22	15	7	1	9 dk, 8 s	145
Hata Kapsama				(3/3)*100 = 100		

Pozitif test aşamasında elde edilen sonuçlar tüm mutantların öldürüldüğünü göstermektedir. Öldürülen mutant uygulamalara ait test koşum süreleri aşağıdaki şekilde gösterildiği gibidir.



Şekil 7.3. OSÇ Pozitif Test Süreleri

OSÇ'ler ile yapılan negatif test sırasında da mutasyona uğratılmış 3 farklı mutant uygulama üzerinde test senaryoları elde edilmiş ve her bir test kümesi orijinal uygulama üzerinde test edilmiştir. Aşağıdaki tablolarda (Tablo 7.6, 7.7 ve 7.8) negatif test sonuçları görülmektedir. Bu tablolarda IRE'lerden üretilen senaryolarda bulunan hatalara göre daha az hatalı test dizisi bulduğu saptanmıştır. Buna ek olarak her bir model için ayrılan toplam test zamanı çok farklılık göstermemektedir.

Tablo 7.6. OSÇ Negatif Test Sonuç Tablosu (Mutant 1)

Negatif Test (Mutant 1)	1FB1 ESG Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	22	18	4	1	8 dk, 49 s	151

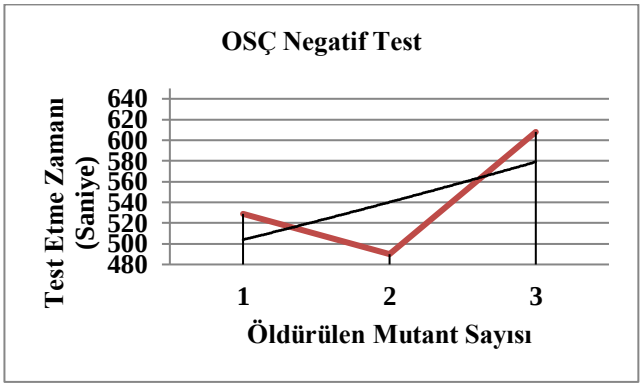
Tablo 7.7. OSÇ Negatif Test Sonuç Tablosu (Mutant 2)

Negatif Test (Mutant 2)	1FB2 ESG Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	20	16	4	1	8 dk, 10 s	142

Tablo 7.8. OSÇ Negatif Test Sonuç Tablosu (Mutant 3)

Negative Test (Mutant 3)	1FA ESG Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	24	13	11	1	10 dk, 8 s	174
Hata Kapsama				(3/3)*100 = 100		

Negatif test aşamasında elde edilen sonuçlar tüm mutantların öldürüldüğünü göstermektedir. Öldürülen mutant uygulamalara ait test koşum süreleri aşağıdaki şekilde gösterildiği gibidir.



Şekil 7.4. OSÇ Negatif Test Süreleri

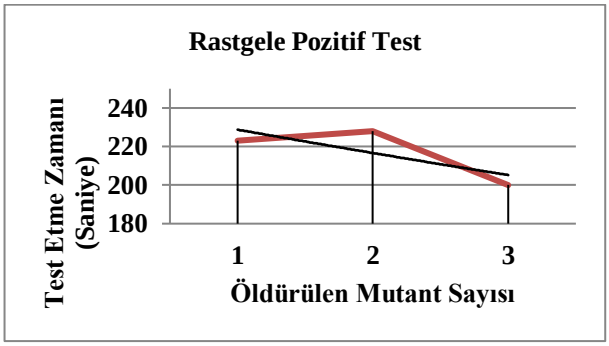
Son olarak GraphWalker aracılığıyla rastgele üretilen test senaryoları için de pozitif test ve negatif test yöntemleri uygulanmıştır. Orijinal modele ait çizgelerden 1A modeli için 10 ve 1B modeli için 13 adet test dizisi bulunmaktadır.

Bu test dizileri mutant uygulamalar üzerinde test edildiği zaman sonuç tablosu aşağıdaki gibi olmaktadır. Oluşturulan test dizilerinin uzunlukları ve koşum süreleri arasında çok fark görülmemektedir.

Tablo 7.9. Pozitif Test Sonuç Tablosu

Pozitif Test (GraphWalker)	GraphWalker Orijinal Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
1FA Modeline ait uygulama	10	2	8	1	3 dk, 20 s	52
1FB1 Modeline ait uygulama	13	11	2	1	3 dk 43 s	73
1FB2 Modeline ait uygulama	13	11	2	1	3 dk 48 s	73
Hata Kapsama				(3/3)*100 = 100		

Pozitif test aşamasında elde edilen sonuçlar tüm mutantların öldürüldüğünü göstermektedir. Öldürülen mutant uygulamalara ait test koşum süreleri aşağıdaki şekilde gösterildiği gibidir.



Şekil 7.5. Rastgele Pozitif Test Süreleri

GraphWalker ile yapılan negatif test sırasında mutasyona uğratılmış 3 farklı mutant uygulama üzerinde test senaryoları elde edilmiş ve her bir test kümesi orijinal uygulama üzerinde test edilmiştir. Aşağıdaki tablolarda (Tablo 7.10, 7.11 ve 7.12) negatif test sonuçları görülmektedir.

Tablo 7.10. GraphWalker Negatif Test Sonuç Tablosu (Mutant 1)

Negatif Test (Mutant 1)	1FB1 GraphWalker Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	8	6	2	1	2 dk 39 s	59

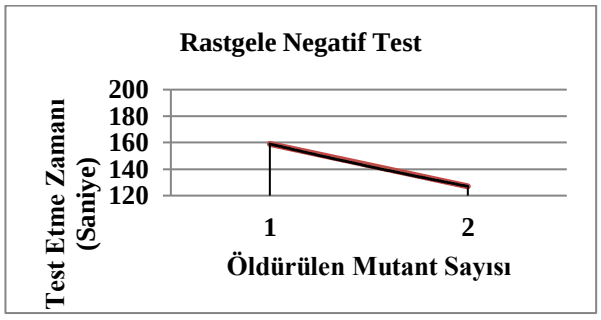
Tablo 7.11. GraphWalker Negatif Test Sonuç Tablosu (Mutant 2)

Negatif Test (Mutant 2)	1FB2 GraphWalker Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	4	0	4	1	1 dk 57 s	74

Tablo 7.12. GraphWalker Negatif Test Sonuç Tablosu (Mutant 3)

Negative Test (Mutant 3)	1FA GraphWalker Modeli Test Senaryoları	Başarılı	Hatalı	Mutasyon Skoru	Toplam Test Zamanı	Toplam karakter boyutu
Orijinal Uygulama	10	10	0	0	3 dk 8 s	58
Hata Kapsama				(2/3)*100 = 66,6		

Negatif test aşamasında elde edilen sonuçlar 1'i hariç olmak üzere diğer mutantların öldürüldüğünü göstermektedir. Öldürülen mutant uygulamalara ait test koşum süreleri aşağıdaki şekilde gösterildiği gibidir. IRE ve OSÇ ile üretilen test dizileriyle yapılan negatif test mutasyon skorunun, GraphWalker aracıyla üretilen test dizilerinden %33.4 oranında daha fazla olduğu görülmüştür.



Şekil 7.6. Rastgele Negatif Test Süreleri

IRE ve OSÇ için negatif test (NT) için kullanılan test dizilerinin toplam uzunlukları hesaplanırken mutant modellerden üretilen test dizilerinin toplam karakter boyutları hesaplanmış ve ortalaması alınmıştır (Bkz. Tablo 7.13).

- $(647 + 698 + 41) \,/\, 3 \;=\; 462$ (IRE)
- $(151 + 142 + 174)/\, 3 = 156$ (OSÇ)
- $(59 + 74 + 58)/\, 3 = 63{,}6 \;\approx 64$ (GraphWalker)

Pozitif test (PT) aşamasında ise orijinal test dizilerinin toplam uzunlukları hesaplanmıştır.

- $417 + 38 = 455$ (IRE)
- 145 (OSÇ)
- $52 + 73 = 125$ (GraphWalker)

Aşağıda yer alan sonuç tablosu incelendiğinde, IRE'lerin, OSÇ'ye ve GraphWalker'a göre daha fazla test senaryosu üretimi sağladığı gözlenmektedir. Bu farklılığın temel nedeni ise IRE'lerde sembolün yerine göre olayların (event) içeriklerine (context) bağlı permütasyonları içermesidir. Bu sayede üretilen test dizilerinde daha fazla hata yakalama özelliği sağlanabilmektedir. Bir sistemin ya da sadece belli bir modelin daha fazla test edilmesini istediğimiz durumlarda IRE ile üretilen test senaryoları tercih edilebilir. Fakat daha az senaryoyu test edilerek bulunan hataların oranından feragat edilmek istenirse OSÇ ile üretilen test dizileri tercih edilebilir. Test dizilerinin uzunluğu açısından $T(IRE) > T(OSÇ) > T(RND)$ ilişkisi olduğunu söyleyebiliriz.

Testler çok sınırlı bir zaman dilimi içerisinde gerçekleştirilmek istenirse GraphWalker ile rastgele olarak üretilen test senaryolarının koşumu süre bakımından daha kısa sürecektir. IRE'lerin test senaryo sayısının daha fazla oluşu doğal olarak test sürelerini de uzatmıştır. Süre açısından $T(IRE) > T(OSÇ) > T(RND)$ ilişkisi görülmektedir. Testler sırasında gerçek bir kullanıcı davranışının sağlanabilmesi için sayfa geçişleri arasında 1 saniye, metin yazımları sırasında 2 saniyelik sleep fonksiyonları kullanılmıştır.

IRE ve OSÇ'ler için pozitif ve negatif test mutasyon skorları eşit olarak bulunmuştur. GraphWalker'da ise sadece negatif test için mutasyon skoru diğerlerinden az, pozitif test içinse aynı olduğu görülmüştür. Hata yakalama oranını mutantları bulma oranıyla ilişkilendirdiğimiz zaman $T(IRE) = T(OSÇ) > T(RND)$ olduğu görülmektedir.

Tablo 7.13. Kıyaslama ve Sonuç Tablosu

	Kullanılan Yöntemler					
	Indexed Regular Expression (IRE)		**Olay Sıra Çizgeleri (OSÇ)**		**Graph Walker**	
	Sembol Kapsama		**Olay Tuple Kapsama (Event-2)**		**Kenar Kapsama**	
	NT	**PT**	**NT**	**PT**	**NT**	**PT**
Hata Kapsama (%)	**100**	**100**	**100**	**100**	**66,6**	**100**
Mutasyon Skoru = Öldürülen mutant sayısı – Denk mutantların sayısı / Toplam mutant sayısı	**1**	**1**	**1**	**1**	**0,66**	**1**
Test Dizilerinin Toplam Uzunlukları (Semboller)	**462**	**455**	**156**	**145**	**64**	**125**
Testlerin Koşum Süresi (dk)	**19 dk 57 s**	**12 dk 39 s**	**9 dk 2 s**	**9 dk 23 s**	**2 dk 35 s**	**3 dk 37 s**

Sonuç olarak; model tabanlı test yönteminin insan etkisini azaltıp, otomatik bir şekilde test senaryolarının üretilmesini ve çalıştırılmasını sağladığı için, hata tespit yeteneğini, test süresi, test maliyeti ve test modellerinin gelişimi gibi avantajlar sağladığı görülmüştür.

Bunlara ek olarak; modelleme bilgisi ve test altındaki uygulamanın ihtiyaçlarının tam olarak bilinmemesinden, testin somut olmayan özelliklerinden dolayı bazı zorluklar da kaydedilmiştir.

Tez kapsamında elde edilen mutant sayısının azlığı bir sorun olarak görünsede bu bizlere genel hakkında yorumlar yapabilme imkanı vermektedir.

Ancak ileriki çalışma olarak mutant sayısını arttırma ve sonuçları tekrar değerlendirme planımız dahilindedir.

KAYNAKLAR DİZİNİ

Amalfitano, D., Fasolino, A. R., Tramontana, P., Ta, B. D., and Memon, A. M., (2015). MobiGUITAR: Automated model-based testing of mobile apps. IEEE Software, 32(5), 53-59.

Android, https://developer.android.com/about/dashboards/ (Erişim Tarihi: 24 Kasım 2018)

Anroid Lifecycle, https://developer.android.com/guide/components/activities/activity-lifecycle (Erişim Tarihi: 18 Nisan 2019)

Android UI, https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/ui/overview.html (Erişim Tarihi: 18 Nisan 2019)

Android Testing Fundamentals, https://stuff.mit.edu/afs/sipb/project/android/docs/tools/testing/testing_android.html (Erişim Tarihi: 18 Nisan 2019)

Android Testing Support, https://developer.android.com/training/testing/ (Erişim Tarihi: 11 Mart 2018.)

Apache License, http://www.apache.org/licenses/LICENSE-2.0, Ocak 2004

Appbrain Statistics, http://www.appbrain.com/stats/ (Erişim Tarihi: 11 Mart 2018)

Baek, Y. and Bae, D. , Automated model-based Android GUI testing using multi-level GUI comparison criteria, Proceeding, ASE 2016 Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, Pages 238-249.

Belli, F. Finite state testing and analysis of graphical user interfaces. Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on. IEEE, 2001.

Belli F., Beyazıt M. and Güler N., Event-Oriented, Model-Based GUI Testing and Reliability Assessment—Approach and Case Study, Advances in Computers, vol. 85, A. Memon, Ed., (2012) 277-326.

Belli F., Endo A.T., Linschulte M., and Simao A., A Holistic Approach to Model-Based Testing of Web Service Compositions, Software: Practice and Experience, vol.44, no.2, pp. 201–23, 2014

KAYNAKLAR DİZİNİ (devam)

Belli, F. Regular Expressions for Fault Handling in Sequential Circuits., ARCS 2015, March 24 – 27, 2015, Porto/Portugal

Belli, F, Budnik, C. J., Hollmann, A., Tuglular, T. and Wong, **W. E.** "Model-based mutation testing—approach and case studies." Science of Computer Programming 120 (2016): 25-48

Belli, F., Beyazıt M., Budnik C. J. and Tuglular, T., Advances in Model-Based Testing of Graphical User Interfaces. In: Atif M. Memon, editor, *Advances in Computers, Vol. 107*, Burlington: Academic Press, 2017, pp. 219-280., ISBN: 978-0-12-812228-0

Bougé, L.: A contribution to the theory of program testing. Theoretical Computer Science vol. 37, 151-181 (1985).

Choudhary S. R., Gorla A. and Orso, A., "Automated test input generation for android: Are we there yet? (e)", Automated Software Engineering (ASE) 2015 30th IEEE/ACM international conference on, pp. 429-440, Nov 2015.

Chow, T. S. "Testing software design modeled by finite-state machines." IEEE transactions on software engineering 3 (1978): 178-187.

Codeburst https://codeburst.io/how-your-android-code-compiles-to-deliver-apk-package-file-9f180129c9bf son erişim 2019/04/18

Lee, D. and Yannakakis, M, "Principles and Methods of Testing Finite State Machine - A Survey," IEEE, vol. 84, no. 8, pp. 1090-1123, August 1996.

DeMillo, R. A., Lipton R. J. and Sayward, F. G., "Hints on test data selection: Help for the practicing programmer." Computer 11.4 (1978): 34-41.

Deng, L., Offutt, J. and Samudio, D. "Is Mutation Analysis Effective at Testing Android Apps?." Software Quality, Reliability and Security (QRS), 2017 IEEE International Conference on. IEEE, 2017.

GNU General Public License, https://www.gnu.org/licenses/gpl-2.0.html, Haziran 1991

Goodenough, J.B. and Gerhart, S.L.: Toward a theory of test data selection. IEEE Transactions on software Engineering vol. SE-1, 156–173 (1975).

Google Play, https://play.google.com/store (Erişim Tarihi: 11 Mart 2018)

KAYNAKLAR DİZİNİ (devam)

Google, Test Your App, https://developer.android.com/studio/test/index.html (Erişim Tarihi: 11 Mart 2018)

GraphWalker, http://graphwalker.github.io/ (Erişim Tarihi: 12 Nisan 2019)

Embedded Systems Innovation, Model-based testing https://redesign.esi.nl/research/methods-and-tools/integration-test/19.dot (Erişim Tarihi: 11 Mart 2018)

F-Droid, https://f-droid.org/en/ (Erişim Tarihi: 11 Mart 2018)

Howden, W.E.: Reliability of the path analysis testing strategy. IEEE Transactions on Software Engineering vol.3, 208-215 (1976).

JetBrains, https://www.jetbrains.com/idea/ (Erişim Tarihi: 30 Kasım 2018)

JFLAP, http://www.jflap.org/, (Erişim Tarihi: 11 Mart 2018)

Jia, Y. and Harman, M., An Analysis and Survey of the Development of Mutation Testing, IEEE Transactions on Software Engineering (Volume: 37 , Issue: 5 , Sept.-Oct. 2011)

Machiry A., Tahiliani, R. and Naik, M. 2013. Dynodroid: an input generation system for Android apps. In Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013. 224–234.

Miguel, J. L. S. and Takada, S., Generating Test Cases for Android Applications through GUI Modeling, Usage Modeling, and Change Analysis, Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering, July 13-15, 2015, Yokohama, Japan DOI:10.1145/2790798.2790823

Kilincceker, O., Turk, E., Challenger, M., and Belli F., "Applying the Ideal Testing Framework to HDL Programs", ARCS 2018 – 31st International Conference on Architecture of Computing Systems, 14th Workshop on Dependability and Fault Tolerance (VERFE'18), in press.

Kramer, A. and Legeard, B., Model-Based Testing Essentials-Guide to the ISTQB Certified Model-Based Tester: Foundation Level. John Wiley & Sons, 2016

KAYNAKLAR DİZİNİ (devam)

MaTeLo, http://www.all4tec.com/matelo-generation-strategies (Erişim Tarihi: 11 Mart 2018)

Memon, A. M., Soffa M. L. and Pollack, M. E., "Coverage criteria for GUI testing." ACM SIGSOFT Software Engineering Notes 26.5 (2001): 256-267.

Memon A.M., Banerjee I. and Nagarajan A., GUI Ripping: reverse engineering of graphical user interfaces for testing, Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003), IEEE Computer Society, (2003) 260-269.

Monkey, https://developer.android.com/studio/test/monkey (Erişim Tarihi: 11 Mart 2018)

Muangsiri W. and Takada S., Random GUI Testing of Android Application Using Behavioral Model, International Journal of Software Engineering and Knowledge Engineering 2017 27:09n10, 1603-1612

Open Handset Alliance, https://www.openhandsetalliance.com/ (Erişim Tarihi: 24 Kasım 2018)

PQ-Analysis, http://download.ivknet.de/ (Erişim Tarihi: 11 Mart 2018)

Smieh, Anatomy Physiology of an Android, CC BY-SA 3.0, 2012

Telegram, https://fdroid.org/packages/org.telegram.messenger/, (Erişim Tarihi: 11 Mart 2018)

Ting, S., Guozhu, M., Yuting, C., Ke, W., Weiming, Y., Yao, Y., Geguang, P., Yang, L. and Zhendong, S., Guided, stochastic model-based GUI testing of Android apps, Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, September 04-08, 2017, Paderborn, Germany.

UI Automator, https://developer.android.com/training/testing/ui-automator (Erişim Tarihi: 11 Mart 2018)

Utting, M., and Legeard B., Practical model-based testing: a tools approach. Morgan Kaufmann, 2010.

yEd, https://www.yworks.com/products/yed (Erişim Tarihi: 11 Nisan 2019)

TEŞEKKÜR

Öncelikle bu tez konusu üzerinde bana çalışma fırsatı sunan ve ayrıca yüksek lisans eğitimim süresince danışmanımlarım olan Dr. Öğr. Üyesi Moharram CHALLENGER'a ve Prof. Dr. Fevzi Belli'ye deneyimleri, bilgileri ve önerileriyle araştırma ve geliştirmeye yönlendirmesi ve yüksek lisans eğitimim süresince hiçbir zaman desteğini esirgememesinden dolayı teşekkürü bir borç bilirim.

Ayrıca tüm fikir ve desteklerinden dolayı Onur KILINÇÇEKER'e, Dr. Öğr. Üyesi Nida GÖKÇE NARİN'e öte yandan tüm proje çalışmalarımda destek olan ve tezimin düzenlenmesinde emeği geçen iş arkadaşım Evrim AKGÜNDÜZ'e, her zaman yanımda olan Tuğçe KALKAVAN'a ve büyük bir özveri ve anlayışla beni destekleyen babama, anneme ve kardeşime teşekkürlerimi sunarım.

Bana yüksek lisans yapma imkânı sunan Uluslararası Bilgisayar Enstitüsü'ne, burada görevli değerli hocalarım ve idari personele manevi desteklerinden ötürü teşekkür ederim. Ayrıca isimlerini sayamadığım tezim boyunca bilimsel etkinliklerde fikir ve görüşlerini ileten tüm bilim insanlarına ve değerli hocalarıma teşekkürlerimi sunmak isterim.

08 / 08 / 2019

Gizem Mercan

ÖZGEÇMİŞ

Gizem Mercan
Adres: Uluslararası Bilgisayar Enstitüsü 35100 Bornova/İZMİR
gmercan91@gmail.com

Eğitim Durumu

Yüksek Lisans	: 2016- , EGE Üniversitesi, Uluslararası Bilgisayar Enstitüsü
Lisans	: 2009-2015, Dokuz Eylül Üniversitesi, Bilgisayar Mühendisliği
Lisans	: 2013 bahar, Konstanz Üniversitesi /Konstanz- Almanya Bilgisayar Bilimleri Değişim öğrencisi

Yabancı Dil

Türkçe	: Anadil
İnigilizce	: İyi derecede

Yayınlar

- Gizem Mercan, Evrim Akgündüz, Onur Kılınççeker, Moharram Challenger and Fevzi Belli. "Android Uygulaması Testi için İdeal Test Ön Çalışması" Proceedings of the 12th Turkish National Software Engineering Symposium, Istanbul, Turkey, September 10-12, 2018.
- Gizem Mercan ve Moharram Challenger. Android Grafiksel Kullanıcı Arayüzleri için Model Tabanlı Test Dizisi Üretimi, Proceedings of the 12th Turkish National Software Engineering Symposium, Istanbul, Turkey, September 10-12, 2018.

Projeler

- Akıllı Ev Sistemleri Projeleri, (Vestel, 2017-halen)
- Akıllı Ev Asistanı Projesi, (Vestel, 2016-2017)
- STB'lerden Data Toplama Projesi (Vestel, 2015-2016)
- M2M Tübitak Araştırma Projesi (Ericsson, 2014-2015)
- Smart Fridge (2015), Bitirme Tezi Projesi (DEU)
- One Time Password Application (2015), Kriptoloji Dersi Dönem Projesi (DEU)
- Desert Recipes (2015), IOS Uygulamaları Dersi Dönem Projesi (DEU)
- Park Mobile (2014), Mobil Uygulamalar Dersi Dönem Projesi (DEU)

Burs ve Ödüller

- UYMS 2018 Yazılım Test Mühendisliği alt alanında "en iyi bildiri ödülü" Gizem Mercan, Evrim Akgündüz, Onur Kılınççeker, Moharram Challenger and Fevzi Belli. "Android Uygulaması Testi için İdeal Test Ön Çalışması"
- Erasmus değişim programı bursu (6 ay, 2013)
- %10 derecesi bursu, Dokuz Eylül Üniversitesi Bilgisayar Mühendisliği bölümü (2010).

İş Deneyimi

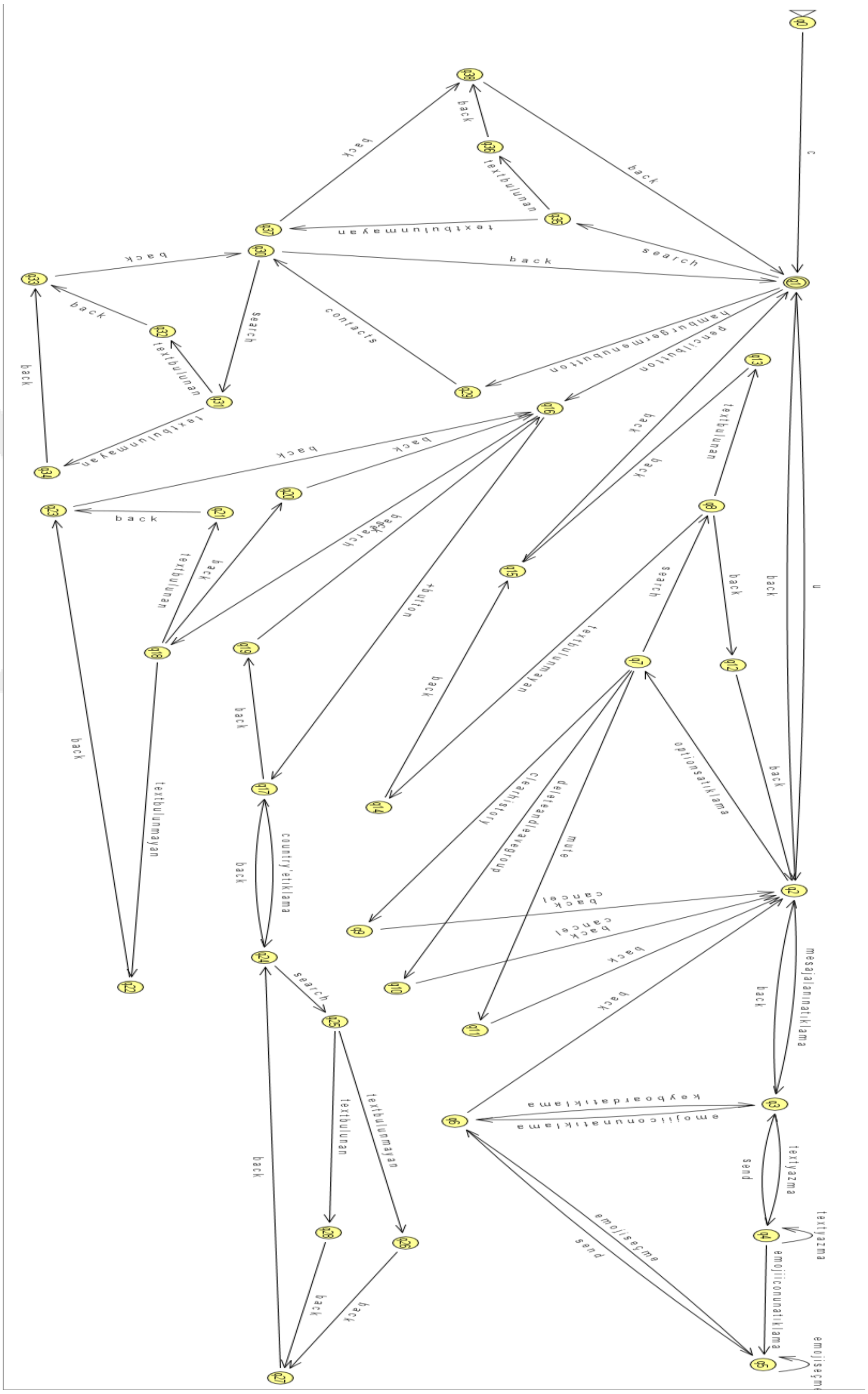
- Vestel, Kıdemli Yazılım Uzmanı, (2015- halen)
- Avea, Part-time Yazılım Test Mühendisi(2015, 6 ay)
- Ericsson, Part-time Yazılım Geliştirici (2013 - 2015)
- TAI Yazılım Mühendisliği stajı (2013)

Bilgisayar dilleri ve program deneyimi

- Java; C, C++; C#; ASP.Net, Php, Swift, Shell Script, HTML, Javascript, Python
- Microsoft Visual Studio, Eclipse IDE, NetBeans IDE, Soap UI, Wireshark, BeyondCompare, Tortoise SVN, Git, XCode, PuTTY, phpMyAdmin, Microsoft Windows Office,
- Veritabanı teknolojileri; MySQL, MSSQL, PostgreSQL, OracleDB, MongoDB, DocumentDB, DynamoD

EKLER

Ek 1 Çok Seviyeli Sonlu Durum Makinesi



Ek 2 PQTestGen ile Üretilen Test Dizileri

1B hatasız modeline ait optimal test dizileri		
A	Aujijbjbbub	Aujiggebb
Aujijtebjbbub	Aujtettebbub	Aujijttebb
Aujbjbbub	Aujbbub	Aujttiebb
Aujibbub	Aujijteijbbub	Aujtiejbb
Aujijtttebbub	Aujibjbbub	Aujtebjbb
Aujijbbub	Aujijtebbub	Aujijijbb
Aujttebbub	Aujijttiebbub	Aujtiebb
Aujijtetebbub	Aujteibbub	Aujtigebb
Aujigejbbub	Aub	Aujijtetiebb
Aujijtiebbub	Aubub	Aujijbjbb
Aujteijbbub	Aujijtebjbb	Aujtettebb
Aujtebbub	Aujbjbb	Aujbb
Aujigggebbub	Aujibb	Aujijteijbb
Aujijteibbub	Aujijtttebb	Aujibjbb
Aujtttebbub	Aujijbb	Aujijtebb
Aujigebbub	Aujttebb	Aujijttiebb
Aujtetebbub	Aujijtetebb	Aujteibb
Aujtiggebbub	Aujigejbb	Aujiggebb
Aujijibbub	Aujijtiebb	Aujijttebb
Aujijtigebbub	Aujteijbb	
Aujijtettebbub	Aujtebb	
Aujiggebbub	Aujigggebb	
Aujijttebbub	Aujijteibb	
Aujttiebbub	Aujtttebb	
Aujtiejbbub	Aujigebb	
Aujtebjbbub	Aujtetebb	
Aujijijbbub	Aujtiggebb	
Aujtiebbub	Aujijibb	
Aujtigebbub	Aujijtigebb	
Aujijtetiebbub	Aujijtettebb	

Ek 2 (devam)

1FB1 hatasız modeline ait optimal test dizileri			
A	Aujteebbjbbub	Aujttiebbb	Aujijtegebbb
Aujibbub	Aujijttbbbub	Aujijtbibb	Aujtttbbb
Aujigebbbub	Aujijtbtbbbub	Aujijtteebbb	Aujijbjbb
Aujteettbbbub	Aujteggebbbub	Aujigebibb	Aujtbibb
Aujijtiebbbub	Aujteebttbbbub	Aujttbbb	Aujtbttbbb
Aujijbbub	Aujtiebbbub	Aujijtttbbb	Aujtbijbb
Aujteetbbbub	Aujteebibbub	Aujijttiebbb	Aujiggebbb
Aujteebijbbub	Aujteebbbub	Aujijijbb	Aujibjbb
Aujijtbbbub	Aujteebtbbbub	Aujigebttbbb	Aujigebbjbb
Aujtteebbbub	Aujijtbbjbbub	Aujtbtbbb	Aujtegebbb
Aujijtbijbbub	Aujijibbub	Aujigettbbb	
Aujttiebbbub	Aujijtbttbbbub	Aujijtigebbb	
Aujijtbibbub	Aujigetbbbub	Aujtbbjbb	
Aujijtteebbbub	Aujijtegebbbub	Aujigggebbb	
Aujigebibbub	Aujtttbbbub	Aujigebijbb	
Aujttbbbub	Aujijbjbbub	Aujijteebbb	
Aujijtttbbbub	Aujtbibbub	Aujbb	
Aujijttiebbbub	Aujtbttbbbub	Aujtbbb	
Aujijijbbub	Aujtbijbbub	Aujbjbb	
Aujigebttbbbub	Aujiggebbbub	Aujtigebbb	
Aujtbtbbbub	Aujibjbbub	Aujigebtbbb	
Aujigettbbbub	Aujigebbjbbub	Aujteebbjbb	
Aujijtigebbbub	Aujtegebbbub	Aujijttbbb	
Aujtbbjbbub	Aub	Aujijtbtbbb	
Aujigggebbbub	Aubub	Aujteggebbb	
Aujigebijbbub	Aujibb	Aujteebttbbb	
Aujijteebbbub	Aujigebbb	Aujtiebbb	
Aujbbub	Aujteettbbb	Aujteebibb	
Aujtbbbub	Aujijtiebbb	Aujteebbb	
Aujbjbbub	Aujijbb	Aujteebtbbb	
Aujtigebbbub	Aujteetbbb	Aujijtbbjbb	
Aujigebtbbbub	Aujteebijbb	Aujijibb	
	Aujijtbbb	Aujijtbttbbb	
	Aujtteebbb	Aujigetbbb	
	Aujijtbijbb		

Ek 2 (devam)

1FB2 hatalı modeline ait optimal test dizileri			
A	Aujtibttibbbub	Aujigbtiebbb	Aujijibb
Aujtttibbbub	Aujtibibbub	Aujijteeibbb	Aujigbttibbb
Aujibbub	Aujtibteibbbub	Aujtteibbb	Aujtibbjbb
Aujtieebbbub	Aujigbbbub	Aujigbtigbbb	Aujtibtttibbb
Aujigebbbub	Aujijtttibbbub	Aujijttibbb	Aujijbjbb
Aujijtiebbbub	Aujtibbbub	Aujigbijbb	Aujigggbbb
Aujijbbub	Aujtiebbbub	Aujigbteibbb	Aujiggebbb
Aujiggbbbub	Aujtibtigbbbub	Aujtibijbb	Aujtiegbbb
Aujijtigbbbub	Aujtibtiebbbub	Aujijtteibbb	Aujibjbb
Aujijtetibbbub	Aujigbbjbbub	Aujtibtetibbb	
Aujijtibbbub	Aujtetibbbub	Aujigbibb	
Aujigbtttibbbub	Aujtigbbbub	Aujijijbb	
Aujtibtibbbub	Aujteibbbub	Aujigbtetibbb	
Aujigbtiebbbub	Aujijibbub	Aujijteibbb	
Aujijteeibbbub	Aujigbttibbbub	Aujigeebbb	
Aujtteibbbub	Aujtibbjbbub	Aujigegbbb	
Aujigbtigbbbub	Aujtibtttibbbub	Aujbb	
Aujijttibbbub	Aujijbjbbub	Aujteeibbb	
Aujigbijbbub	Aujigggbbbub	Aujbjbb	
Aujigbteibbbub	Aujiggebbbub	Aujtiggbbb	
Aujtibijbbub	Aujtiegbbbub	Aujtigebbb	
Aujijtteibbbub	Aujibjbbub	Aujttibbb	
Aujtibtetibbbub	Aub	Aujigbtibbb	
Aujigbibbub	Aubub	Aujtibttibbb	
Aujijijbbub	Aujtttibbb	Aujtibibb	
Aujigbtetibbbub	Aujibb	Aujtibteibbb	
Aujijteibbbub	Aujtieebbb	Aujigbbb	
Aujigeebbbub	Aujigebbb	Aujijtttibbb	
Aujigegbbbub	Aujijtiebbb	Aujtibbb	
Aujbbub	Aujijbb	Aujtiebbb	
Aujteeibbbub	Aujiggbbb	Aujtibtigbbb	
Aujbjbbub	Aujijtigbbb	Aujtibtiebbb	
Aujtiggbbbub	Aujijtetibbb	Aujigbbjbb	
Aujtigebbbub	Aujijtibbb	Aujtetibbb	
Aujttibbbub	Aujigbtttibbb	Aujtigbbb	
Aujigbtibbbub	Aujtibtibbb	Aujteibbb	

Ek 2 (devam)

1A	1FA
A	A
Aub	Aub
Aubub	Aubub
Auombbub	Auodbbbub
Auodbbub	Auocbbbub
Auoybbbub	Auombbbub
Auocbbub	Auoybbbub
Auombb	Auodbbb
Auodbb	Auocbbb
Auoybbb	Auombbb
Auocbb	Auoybbb

2 hatasız modeline ait optimal test dizileri	
A	Apqlvxbbvzbbbbbbub
Aub	Apbpb
Aubub	Apnzbbbpnzbbb
Apb	Apnxbbbpnxbbb
Apnzbbb	Apqlvxbbblvzbbbbbbpqlvxbbblvzbbbbbb
Apnxbbb	Apqbbbpqbbb
Apqlvxbbblvzbbbbbb	Apqlvzbbbbbbpqlvzbbbbbb
Apqbbb	Apqlvxbbbbbbpqlvxbbbbbb
Apqlvzbbbbbb	Apnbbbpnbbb
Apqlvxbbbbbb	Apqlbbbbpqlbbbb
Apnbbb	Apqlvxbbblbbbbpqlvxbbblbbbb
Apqlbbbb	Apqlvxbbvzbbbbbbpqlvxbbvzbbbbbb
Apqlvxbbblbbbb	Aubpb
Apqlvxbbvzbbbbbb	Aubpnzbbb
Apbub	Aubpnxbbb
Apnzbbbub	Aubpqlvxbbblvzbbbbbb
Apnxbbbub	Aubpqbbb
Apqlvxbbblvzbbbbbbub	Aubpqlvzbbbbbb
Apqbbbub	Aubpqlvxbbbbbb
Apqlvzbbbbbbub	Aubpnbbb
Apqlvxbbbbbbub	Aubpqlbbbb
Apnbbbub	Aubpqlvxbbblbbbb
Apqlbbbbub	Aubpqlvxbbvzbbbbbb
Apqlvxbbblbbbbub	

Ek 2 (devam)

3 hatasız modeline ait optimal test dizileri	
A	Awrnzbbnxbbbbwrnzbbnxbbbb
Awrnzbbnzbbbbub	Awrnzbbnxbbnzbbbbwrnzbbnxbbnzbbbb
Awrbbub	Awrnzbbnzbbbb
Awrnxbbbbub	Awrbb
Awrnxbbnzbbbbub	Awrnxbbbb
Awrnzbbbbub	Awrnxbbnzbbbb
Awrnzbbnxbbbbub	Awrnzbbbb
Awrnzbbnxbbnzbbbbub	Awrnzbbnxbbbb
Aub	Awrnzbbnxbbnzbbbb
Aubub	Aubwrnzbbnzbbbb
Awrnzbbnzbbbbwrnzbbnzbbbb	Aubwrbb
Awrbbwrbb	Aubwrnxbbbb
Awrnxbbbbwrnxbbbb	Aubwrnxbbnzbbbb
Awrnxbbnzbbbbwrnxbbnzbbbb	Aubwrnzbbbb
Awrnzbbbbwrnzbbbb	Aubwrnzbbnxbbbb
	Aubwrnzbbnxbbnzbbbb

4 hatasız modeline ait optimal test dizileri
A
Aszbbub
Asxbbub
Aub
Aubub
Aszbbszbb
Asxbbsxbb
Aszbb
Asxbb
Aubszbb
Aubsxbb

Ek 3 TSD ile Üretilen Test Dizileri

OSÇ orijinal modele ait test dizileri	
Aujbb	Aujbb
Asxbb	Awrb
Awrnzbbb	Awrnxbbb
Aplvzbblvxbblbbqbb	Apqbb
A	Apnbb
Auodbb	Apb
Aub	Apnzbbb
Aujttebb	Apnxbbb
Auombb	Aszbb
Auoybbb	Aujtigegejbb
Auocbb	Aujtiggebb
Aujibb	

OSÇ 1FB1 modeline ait test dizileri	
Aujbb	Aujibb
Asxbb	Awrb
Awrnzbbb	Awrnxbbb
Aplvzbblvxbblbbqbb	Apqbb
A	Apnbb
Auodbb	Apb
Aub	Apnzbbb
Aujtteggetiggetbbb	Apnxbbb
Auombb	Aszbb
Auoybbb	Aujijbb
Auocbb	Aujiggetbbb

OSÇ 1FB2 modeline ait test dizileri	
Aujigiggieeibjbb	Awrb
Asxbb	Awrnxbbb
Awrnzbbb	Apqbb
Aplvzbblvxbblbbqbb	Apnbb
A	Apb
Auodbb	Apnzbbb
Aub	Apnxbbb
Aujtteteeibjibb	Aszbb
Auombb	Auocbb
Auoybbb	Aujijbb

Ek 3 (devam)

OSÇ 1FA modeline ait test dizileri	
Aujigiggieeibjbb	Auocybbb
Asxbb	Auodybbb
Awrnzbbb	Awrb
Aplvzbblvxbblbbqbb	Awrnxbbb
A	Apqbb
Auodbb	Apnbb
Aub	Apb
Auombb	Apnzbbb
Auoybbb	Apnxbbb
Auocbb	Aszbb
Auomybbb	Aujtteteeibjibb
Auomybbb	Aujijbb

Ek 4 GraphWalker ile Rastgele Üretilen Test Dizileri

1A	**1B**	**1FA**	**1FB1**	**1FB2**
Auocb	Aujb	Auoybb	Aujigeb	Aujigggegbiggggeeebb
Auomb	Aujb	Auodbb	Aujigeb	Aujteibteieegebigbb
Auocb	Aujb	Auoybb	Aujb	Aujijtiebtigeegeeebigbiggbib
Auoybb	Aujiggeggejttiejb	Auocbb	Aujb	Aujteet
Auoybb	Aujb	Auoybb	Aujtiebtieigebtbb	
Auomb	Aujb	Auocbb	Aujijib	
Auomb	Aujb	Auoybb	Aujb	
Auomb	Aujb	Auocbb	Aujtbteet	
Auomb	Aujb	Auoybb		
Auodb	Aujijib	Auom		
	Aujijib			
	Aujb			
	Aujtte			

Ek 5 PQTestGen ile Üretilip Sıkıştırılan Test Dizileri

1B			
Aujijtetiebbub	Aujteijbbub	Aujijtettebbub	Aujijbjbbub
Aujijtebjbbub	Aujtebbub	Aujiggebbub	Aujtettebbub
Aujbjbbub	Aujigggebbub	Aujijttebbub	Aujbbub
Aujibbub	Aujijteibbub	Aujttiebbub	Aujijteijbbub
Aujijtttebbub	Aujtttebbub	Aujtiejbbub	Aujibjbbub
Aujijbbub	Aujigebbub	Aujtebjbbub	Aujijtebbub
Aujttebbub	Aujtetebbub	Aujijijbbub	Aujijttiebbub
Aujijtetebbub	Aujtiggebbub	Aujtiebbub	Aujteibbub
Aujigejbbub	Aujijibbub	Aujtigebbub	Aubub
Aujijtiebbub	Aujijtigebbub		

1FB1					
Aujijtegebbub	Aujijtbijbbub	Aujtbtbbbub	Aujtigebbbub	Aujteebtbbbub	Aujiggebbbub
Aujibbub	Aujttiebbbub	Aujigettbbbub	Aujigebtbbbub	Aujijtbbjbbub	Aujibjbbub
Aujigebbbub	Aujijtbibbub	Aujijtigebbbub	Aujteebbjbbub	Aujijibbub	Aujigebbjbbub
Aujteettbbbub	Aujijtteebbbub	Aujtbbjbbub	Aujijttbbbub	Aujijtbttbbbub	Aujtegebbbub
Aujijtiebbbub	Aujigebibbub	Aujigggebbbub	Aujijtbtbbbub	Aujigetbbbub	Aubub
Aujijbbub	Aujttbbbub	Aujigebijbbub	Aujteggebbbub	Aujtttbbbub	
Aujteetbbbub	Aujijtttbbbub	Aujijteebbbub	Aujteebttbbbub	Aujijbjbbub	
Aujteebijbbub	Aujijttiebbbub	Aujbbub	Aujtiebbbub	Aujtbibbub	
Aujijtbbbub	Aujijijbbub	Aujtbbbub	Aujteebibbub	Aujtbttbbbub	
Aujtteebbbub	Aujigebttbbbub	Aujbjbbub	Aujteebbbub	Aujtbijbbub	

Ek 5 (devam)

1FB2					
Aujtibtttibbbub	Aujijtibbbub	Aujtibijbbub	Aujteeibbbub	Aujijtttibbbub	Aujigbttibbbub
Aujtttibbbub	Aujigbtttibbbub	Aujijtteibbbub	Aujbjbbub	Aujtibbbbub	Aujtibbjbbub
Aujibbub	Aujtibtibbbub	Aujtibtetibbbub	Aujtiggbbbub	Aujtiebbbub	Aujijbjbbub
Aujtieebbbub	Aujigbtiebbbub	Aujigbibbub	Aujtigebbbub	Aujtibtigbbbub	Aujigggbbbub
Aujigebbbub	Aujijteeibbbub	Aujijijbbub	Aujttibbbub	Aujtibtiebbbub	Aujiggebbbub
Aujijtiebbbub	Aujtteibbbub	Aujigbtetibbbub	Aujigbtibbbub	Aujigbbjbbub	Aujtiegbbbub
Aujijbbub	Aujigbtigbbbub	Aujijteibbbub	Aujtibttibbbub	Aujtetibbbub	Aujibjbbub
Aujiggbbbub	Aujijttibbbub	Aujigeebbbub	Aujtibibbub	Aujtigbbbub	Aubub
Aujijtigbbbub	Aujigbijbbub	Aujigegbbbub	Aujtibteibbbub	Aujteibbbub	
Aujijtetibbbub	Aujigbteibbbub	Aujbbub	Aujigbbbub	Aujijibbub	

1A	1FA
Auoybbbub	Auoybbbub
Aubub	Aubub
Auombbub	Auodbbbub
Auodbbub	Auocbbbub
Auocbbub	Auombbbub