Android Uygulaması Testi için İdeal Test Ön Çalışması

Gizem Mercan[1,2], Evrim Akgündüz[1,2], Onur Kılınççeker[3,4], Moharram Challenger[1], Fevzi Belli[3,5]

[1] Ege Üniversitesi, Izmir, Turkey
[2] Vestel, Manisa, Turkey
[3] Paderborn University, Paderborn, Germany
[4] Mugla Sitki Kocman University, Mugla, Turkey
[5] İzmir Yüksek Teknoloji Enstitüsü, İzmir, Turkey
gmercan91@gmail.com, evrimakgunduz@gmail.com, okilinc@mail.upb.de,moharram.challenger@mail.ege.edu.tr, belli@upb.de

Özet. Bu çalışmanın amacı yazılım testi alanında yaygın olarak kullanılan Bütünsel Test (Holistic Test) ve Mutasyon Testi (Mutation Testing) yöntemlerinin kullanılarak model tabanlı melez bir yöntemin Android uygulamalarının Grafiksel Kullanıcı Arayüz (GKA) testi için öne sürülmesidir. Ayrıca bu melez yöntem test alanında bilinirliği yüksek İdeal Test'in (Ideal Test) gereksinimlerini sağladığı için ayrı bir öneme sahiptir. Öne sürülen melez yöntem sayesinde sistem içindeki kullanıcı arayüz merkezli hataların model ölçeğinde varlığı veya yokluğu, karşılaştırmalı ve deneysel çalışmalar çerçevesinde test edilmiştir. Yöntemin ilk adımı olarak verilen uygulamanın kullanıcı arayüzü bir Sonlu Durum Makinası (SDM) ile modellenmekte ve ardından bu SDM bir Düzenli İfade'ye (Dİ) dönüştürülmektedir. Ardından elde edilen Dİ analizden geçirilerek bağlam tabloları ile ifade edilmekte ve bu tablolar vasıtası ile test dizileri üretilmektedir. Bu işlem pozitif testi tanımlamaktadır. Negatif test için ise aynı işlem SDM'lerden elde edilen mutantlara uygulanmakta ve test dizileri elde edilmektedir. Negatif ve pozitif test için elde edilen test dizileri karşılıklı olarak kod tabalı mutasyonla elde edilen mutantlara ve hatasız sisteme uygulanmaktadır. Test sonuçları tanımlanacak olan test seçim kriterlerine göre bir süzgeçten geçirilmekte ve hem pozitif hemde negatif test için süzgeçten geçirilen test kümeleri elde edilmektedir. Bu işlem sonunda görülmektedir ki bu test kümeleri ideal test gereksinimlerini karşılamaktadır.

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

Ideal Test for Android Testing: Preliminary Work

Gizem Mercan[1,2], Evrim Akgunduz[1,2], Onur Kilincceker[3,4], Moharram Challenger[1], Fevzi Belli[3,5]

[1] Ege University, Izmir, Turkey
[2] Vestel, Manisa, Turkey
[3] Paderborn University, Paderborn, Germany
[4] Mugla Sitki Kocman University, Mugla, Turkey
[5] İzmir Institute of Technology, İzmir, Turkey
gmercan91@gmail.com, evrimakgunduz@gmail.com, okilinc@mail.upb.de,moharram.challenger@mail.ege.edu.tr, belli@upb.de

Özet. This paper proposes a hybrid method combining well-known holistic test and mutation testing in software testing for Graphical User Interface (GUI) testing of an android application. Moreover, this hybrid method satisfies requirements of ideal testing that is well known and important in software testing. Presence and absence of GUI based faults are tested within this work experimentally and comparatively in the scale of given or constructed model. First step of the method is modeling the given GUI of android application by Finite State Machine (FSM) and then converting this FSM to Regular Expression (RE). Then, test sequences are generated from a context table that is obtained analysis of the RE model. This process defines first part of the Holistic Testing namely positive testing. In second part called negative testing, the test sequence generation procedure is applied mutants of the FSM obtained after applying selected mutation operators. The generated test sequences from original and mutant models are executed on mutant and original android applications respectively. Test sequences are filtered by using pre-defined selection criteria for both positive and negative testing to achieve ideal test suites that are satisfying requirements of the ideal testing.

Anahtar Kelimeler: Ideal Test, Android Application, GUI Testing, Test Generation, Holistic Test, Mutation Testing, Finite State Machine, Regular Expression.

1 Giriş

Akıllı cihazların (telefon, tablet vb.) kullanımı gün geçtikçe artış göstermektedir. Mobil uygulamalar bu cihazlar üzerinde çalışan çeşitli yazılımlardır. Bu uygulamalar cihazların kullandıkları işletim sistemine göre farklılıklar göstermektedir. Bu işletim sistemlerinden en yaygın kullanılanları Android, IOS ve Microsoft'tur. Bunların arasında Android, Dünya'da en fazla kullanıcı sayısına sahip işletim sistemidir. Android işletim sistemi için çeşitli dağıtımlara ait, kullanıcıları uygulama indirebileceği marketler bulunmaktadır. Bunların en yaygın ve popüler olan Google Play [1] marketidir. Günümüzde bu mağazadaki uygulama sayısı milyonları ve günlük eklenen uygulama sayısı binleri bulmaktadır [2].

Geliştiriciler tarafından marketlere yüklenen uygulama sayısı bir kalite sorunları barındırmaktadır. Bu sorunlar genellikle yazılımların yeterince test edilmemesinden kaynaklanmaktadır. Çünkü geliştiriciler arasında zamana karşı bir yarış vardır ve bu baskı bir çok hata (fault) barındıran uygulamanın markete yüklenmesi ile sonuçlanmaktadır. Böylece kalite standartlarını taşımayan uygulamalar kullanıcılar tarafından marketlerden indirilmektedir. Daha sonra bu uygulamalar kullanım sırasında bir çok bozukluğa (failure) sebep olmaktadır.

Android uygulamaları zengin kullanıcı arayüzleri (User Interface) ile kullanıcılara sunulmaktadır. Bu kullanıcı arayüzleri cihazlar ile son kullanıcı arasındaki etkileşimi sağlamakta ve çok çeşitli ortamlarla (örneğin; diğer bir uygulama, cihaz vb.) etkileşim halinde olabilmektedir. Bu yüzden kullanıcıların ileride karşılaşabilecekleri bozuklukların kaynaklarından büyük bir çoğunluğu kullanıcı arayüz tabanlı hatalardır.

Bu çalışmanın amacı uygulamalar marketlerde yayınlanmadan önce içerdikleri hataların tespitine yöneliktir. Çünkü bu uygulamaların kalitesi açısından çok önemlidir. Ayrıca bu çalışma Grafiksel Kullanıcı Arayüz (GKA) tabanlı hataların test edilmesini hedeflemektedir. Bunun için yazılım testinde bilinirliği yüksek bütünsel test (Holistic Test) [9] ve mutasyon testi (Mutation Testing) [17] birlikte kullanılarak melez bir yöntem ileri sürülmektedir. Bu çalışmada bütünsel testin pozitif ve negatif test durumları olarak kullanımına yer verilmiştir. Öne sürülen melez yöntemin kullanışlılığı örnek durum çalışması üzerinde gösterilmektedir. Daha da önemlisi bu melez yöntem vasıtası ile ileri sürülen yöntem ile yine yazılım testi konusunda bilinirliği yüksek İdeal Test'in (Ideal Test) [19] gereksinimlerinin sağlandığı deneysel olarak gösterilmiştir. Öne sürülen yöntemin bir benzeri Kılınççeker ve diğerleri [22] tarafından teorik altyapı ile detaylıca açıklanmaktadır. [22]'de öne sürülen yöntemin İdeal Test'in (Ideal Test) [19] gereksinimlerinin sağlandığı ile ilgili detaylar bu çalışmada bulunabilir. Güvenilirlik (reliability) gereksinimi ileri sürülen test kriterleri çerçevesinde üretilen test dizilerinin tutarlılığı (consistency) ve geçerlilik (validity) gereksinimi ise üretilen test dizilerinin hataları yakalama kabiliyeti olarak açıklanmaktadır. Bu gereksinimler testin kalitesi açısından çok önemli olduğu gibi sistem içerisindeki hatanın varlığını (presence) veya yokluğunu (absence), model ölçeğinde karşılaştırmalı ve deneysel çalışmalar çerçevesinde test edilmiştir.

2 İlgili Çalışmalar

Android uygulamalarının kullanıcı arayüz testleri incelendiğinde, öne sürülen metotlar genel olarak rastgele (random) ve model tabanlı (model based) olmak üzere ikiye ayrılabilir. Rastgele test üretimi yöntemleri android uygulaması kullanıcı arayüzü testi için rastgele test dizileri üretilmesi esasına dayanır. Model tabanlı yöntemlerde ise test altındaki android uygulaması arayüzü bir model (genellikle sonlu durum makinası) ile ifade edilir. Bu işlem elle veya otomatik olarak gerçekleştirilebilmektedir.

Rastgele test yöntemlerine Monkey [6] ve Dynodroid [7] araçları örnek verilebilir. Bu yöntemler genel itibari ile kara kutu (black box) testi esasına dayanmaktadır. Yani test altındaki uygulamanın kaynak kodlarına erişime gerek duyulmamaktadır. Monkey [6], test edici (tester) tarafından belirlenen sayıda test dizisi üretme özelliğine sahiptir. Bu test dizilerini ayrıca tekrarlanabilir şekilde üretebildiği için android uygulamalarının stres testi içinde kullanılabilir. Monkey [6] ayrıca sistem seviyesinde girdi değerleri üretebildiği için test altındaki uygulama haricinde sistemin

bileşenlerinin de test edilmesine olanak sağlar. Ancak üretilecek test dizileri sadece test altında ki uygulamaya da kısıtlanabilmektedir. Dynodroid [7] ise Monkey'e benzer bir rastgele test dizisi üretme esasına dayanmakla birlikte, deneysel çalışmalara göre Monkey'e göre kod kapsama açısından daha yüksek ve hata yakalama açısından daha iyi sonuçlar vermektedir [7]. Dynodroid, bu avantajlarına rağmen Monkey ile kıyaslandığında 5 kat daha yavaş çalışmaktadır. Her ne kadar rastgele tabanlı yöntemler test dizisi üretimi için kolay çözüm olarak gözüksede rastgele yöntemin doğası gereği çok sayıda fazlalık ve kullanışsız test dizisi üretimine sebep olabilmektedirler [13].

Model tabanlı test kavramı, yazılım testi alanında, Chow'un [8] çok büyük öneme sahip çalışmasına kadar dayanmaktadır. Bu çalışmada Chow verilen bir yazılım sistemini Sonlu Durum Makinası (SDM) ile modellemiş ve bu SDM'den test dizisi üretimine olanak sağlayan W-yöntemi adında yeni bir algoritma öne sürmüştür. Ayrıca SDM'lerden farklı olarak Olay Sıra Çizgesi (OSÇ) (Event Sequence Graph) ve Olay Akış Çizgesi (OAÇ) (Event Flow Graph) kavramları hemen hemen aynı zamanda sırasıyla Belli [9] ve Menon ve diğerleri [10] tarafından ortaya atılmıştır. Ayrıca model tabanlı test konusunda detaylı bilgi [11,12] çalışmasında verilmektedir.

Android uygulamalarının model tabanlı testi [14,15,16] ise halen güncel bir çalışma alanıdır. Bu konuda MobiGUITAR (Mobile GUI Testing Framework) aracı Amalfitano ve diğerleri [14] tarafından öne sürülmüştür. Bu araç ile öne sürülen yöntem ripleme (ripping), üretim (generation) ve koşum (execution) aşamalarından oluşmaktadır. Ripleme yöntemi ile otomatik olarak elde edilen modelden üretim aşamasında test dizileri üretilmekte ve ardından koşum aşamasında daha önce üretilen test dizileri android uygulaması üzerinde koşulmaktadır.

Baek ve Bae [15] ise model tabanlı android GUI testi için çok seviyeli karşılaştırma kriter seti önermişlerdir. Bu sayede birden fazla soyutlama (abstraction) seviyesinin seçimini sağlamıştır. Öne sürülen çalışmanın aktivite tabanlı GUI modelinden daha etkin sonuçlar verdiğini göstermişlerdir.

Su ve diğerleri [16] tarafından ise olasılıksal model tabanlı test yaklaşımı olan Stoat öne sürülmüştür. Stoat, 93 açık kaynak android uygulamasında değerlendirilmiştir. Bunun sonucunda Stoat'ın var olan modelleme araçlarından %17-31 oranında daha fazla kodu kapsadığı sonucuna varılmıştır.

Bütünsel test yöntemi, Belli [9] tarafından ortaya atılmış ve ilk olarak yazılım GUI testinin model tabanlı testi için kullanılmıştır. Bu yaklaşımda pozitif ve negatif test olmak üzere iki bölüme ayrılmaktadır. Pozitif test aşamasında yazılım legal test girdileri ile negatif test aşamasında ise yazılım legal olmayan test girdileri ile test edilmektedir. Bu testler sonucunda elde edilen sonuçlar başarılı (successful) ve başarısız (fail) olma durumlarına göre değerlendirilmektedir. Örneğin günümüzde hemen her bireyin kullandığı online banka işlemleri giriş ekranını düşünecek olursak. Bu giriş ekranının ilk aşaması (giriş modülü) kullanıcının müşteri numarası ve şifresinin girilmesidir. Bu giriş modülünde müşteri numarası bölümüne girdi olarak bir rakam girilmesi positif test durumunu ve aksine rakam olmayan bir verinin girilmesi negatif test durumunu açıklamaktadır. Böylece bu giriş modülünün hem positif hemde negatif test durumları ile sınanması ise bütünsel test kavramı ile açıklanabilir.

Mutasyon testi, Demillo ve diğerleri [17] tarafından ilk olarak yazılım testi için ortaya atılmıştır. Bu yaklaşım test altındaki yazılımın her bir mutant bir veya birçok

hatayı içerecek şekilde değişime uğratılarak mutantların elde edilmesine dayanır. Test üretim aşamasından bağımsızdır ve hata tabanlı bir yaklaşımdır. Başka bir şekilde üretilen mutantlar hatayı yakalamalarına göre ölü (killed) veya yaşayan (lived) olarak sınıflandırılır ve test kalitesi için süreç sonunda mutasyon katsayısı (mutation score) hesaplanır. Mutasyon testinin amacı üretilen test dizilerinin hata bulma verimliliğinin ölçülmesi ve test edilmesidir. Mutasyon testi kod tabanlı (code-based mutation) ve model tabanlı (model-based mutation) olmak üzere ikiye ayrılabilir. Mevcut çalışma mutasyonları model tabanlı olarak üretir iken test dizilerinin koşumu aşamasında model tabanlı mutantlara karşılık gelen kod tabanlı mutasyonlar kullanılmıştır. Örneğin model üzerinde iki düğüm arasında var olmayan bir kenarın modele eklenmesi olarak açıklanabilir. Burada tek bir operatör (kenar ekleme) kullanılarak mutant oluşturulduğu için bu işlem birinci dereceden mutasyona örnektir. Birden fazla operatör kullanılarak oluşturulan mutasyonlar ise ikinci veya yüksek dereceden mutasyona örnek olarak verilebilir. Model tabanlı mutasyon için kullanılan operatörler çeşitlilik göstermektedir [28]. Mutasyon testinin hem android testi [18] hemde donanım sistemleri [22] testi için kullanımı mevcuttur.

İdeal test ilk olarak Goodenough ve Gerhart [19] tarafından yazılım testi için teorik olarak tanımlanmıştır. Bu çalışmada ideal testin gereksinimleri olan güvenilirlik (reliability) ve geçerlilik (validity) kavramlarının tanımları yapılmıştır. Bir önceki bölümde verildiği üzere güvenilirlik, tanımlanan test kriterlerine göre üretilen test dizilerinin tutarlılığına vurgu yapar iken, geçerlilik bu test dizilerinin hata yakalama kabiliyetleri ile açıklanır. Howden [20] ideal test için patika analizi yöntemini öne sürmüş ve bu yöntemin geçerlilik gereksinimini sağladığını göstermiştir. Bouge [21] ideal test için daha önce verilen teorik çerçeveye eğilim (bias) ve kabul edilebilirlik (acceptability) gereksinimlerini ekleyerek bunu genişletmiştir. Ayrıca Kılınççeker ve diğerleri [22], bu çalışmaya benzer bir ideal test önerisini donanım tanımlama dilleri (hardware description language) testi için öne sürmüşlerdir. Yapılan detaylı literatür taraması sonuçlarına göre ideal testin android uygulaması GKA testi için kullanımına rastlanılmamıştır.

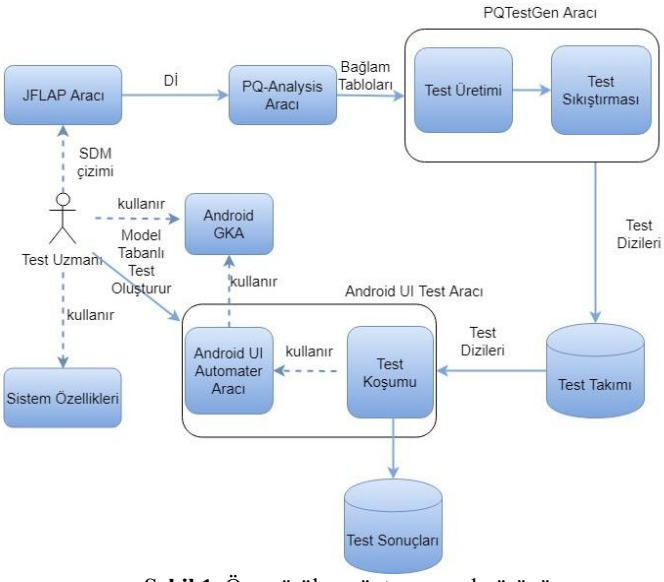
3 Yöntem

Çalışma kapsamında, android uygulaması GKA'sı üzerinde ideal test yaklaşımı için bütünsel test ile mutasyon testi birlikte kullanılarak yeni bir melez yöntem öne sürülecektir. Öne sürülen melez yöntemin test süreci genel görünümü Şekil 1'de verilmektedir.

Bu aşamalar ilk olarak sisteme uygun olacak şekilde test edici (tester) tarafından test altındaki Android uygulamasına ait SDM modelin oluşturulmasını kapsamaktadır. Ayrıca bu SDM oluşturulmasına gerek kalmaksızın, eğer mümkünse, sistem özelliklerinden (specification) SDM elde edilebilir. Uygun SDM modellerinin çizilmesinden veya elde edilmesinden sonra bu SDM modellerine ait olacak şekilde Düzenli İfadelerin (Dİ) (Regular Expression) üretilmesi gerekmektedir. Üretilen Dİ'den PQ-Analysis aracı [23] kullanılarak bağlam tabloları oluşturulmaktadır. Oluşturulan bağlam tabloları gerekli test dizilerinin üretilmesi için PQTestGen [22] aracına girdi olarak verilmektedir. PQTestGen aracının [22] çıktısında elde edilen dizilerden test kümesi oluşturmaktadır. Elde edilen test kümesi, test koşumu yapılabilmesi ve sonuçlarının alınabilmesi için modelden oluşturduğumuz test aracına verilmesi sağlanır. Test senaryolarının koşumu sırasında hatalı test dizileri ve başarılı

test dizileri için alınan sonuçlar toplanır. Toplanan sonuçlar üzerinden başarısız olan senaryoların hata durumları incelenir ve gerekli değerlendirmeler yapılır.

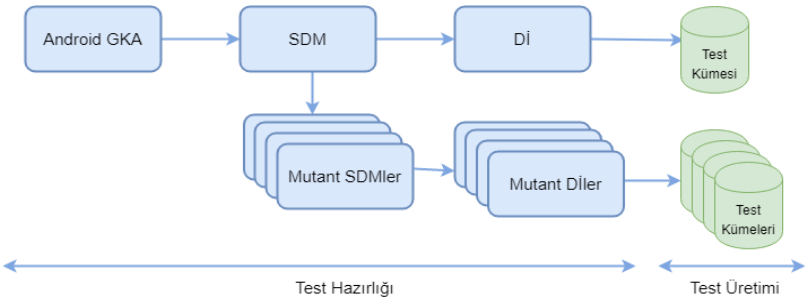
Öne sürülen melez yöntem 2 ana aşamadan oluşmaktadır. Bunların ilki test hazırlığı ve test üretimi, ikincisi ise test etme ve test seçimidir.



Şekil 1. Öne sürülen yöntem genel görünüm

3.1. Test Hazırlığı ve Test Üretimi

Bu aşama test altındaki android uygulamasının SDM ile modellenmesi ve bu modelden mutasyonların oluşturulması ve ardından hatasız ve mutant SDM modellerin Dİ'lere dönüştürülmesi aşamalarından oluşmaktadır. Bu aşamalar için Şekil 2'de verilmektedir.



Şekil 2. Test Hazırlığı ve Test Üretimi

İlk olarak modelleme aşamasında Android uygulamasına ait olacak şekilde JFLAP [24] programı kullanılarak bir SDM çizilecektir. Bu model üzerinde her bir düğüm kullanıcının uğrayabileceği sayfaları her bir kenar ise bu sayfa geçişleri arasında kullanılabilecek olan arayüz bileşenlerini göstermektedir. Modelleme sırasında

tasarlanan SDM'nin çok büyük olmaması ölçeklenebilmesi açısından daha iyi bir seçim olacaktır.

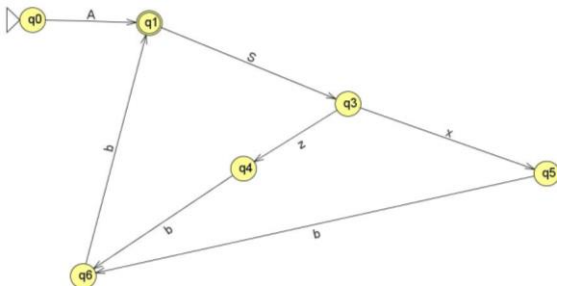
Sonraki aşama uygulamanın normal davranışı gösteren model dışında beklenmeyen durumlarını da mutant modeller üzerinden tanımlayabiliriz. Bu mutant modeller daha önceden belirlenen SDM'ye özgü mutasyon operatörleri (durum ekleme, durum silme, geçiş ekleme, geçiş silme gibi) ile elde edilir. Hatalı test dizilerinin üretiminin yapılabilmesi için mutant bir modele ihtiyacımız vardır. Bu modelin oluşturulması sırasında test edilmek istenen durumlar (states) için beklenmedik durum geçişleri (transition) gereklidir.

Bahsedilen mutasyonlu modele ait uygulama için hatasız uygulama üzerine hatalı kod enjekte işlemi yapılarak kod tabanlı mutant bir uygulama elde edilir. Bunun için iki veya yüksek mertebeden mutasyonlar kod tabanlı mutasyon operatörleri yardımıyla elde edilir. Çünkü ilerleyen aşamalarda açıklanabileceği gibi elde edilen test dizilerinin bazıları mutant uygulama üzerinde koşulacaktır.

Ideal test için gerekli olan mutasyonlu uygulama üretimi aşamasında tersine mühendislik yapılarak UI Automater [25] programı üzerinden GUI bileşenlerine ulaşılarak hangi GUI bileşenleri üzerinde bir mutasyon yapılabileceği konusunda bilgi sahibi olunur.

Test hazırlığı son aşamasında hatasız ve mutant SDM'ler temel alınarak Dİ'ler üretilecektir. Bu işlem JFLAP [24] üzerinden gerçekleştirilecektir. Dİ'ler kullanılarak PQTestGen aracı [22] ile test senaryoları üretilecektir. Örnek olarak; aşağıdaki SDM'den (Şekil 3) üretilen düzenli ifade A((Sxb+Szb)b)* şeklindedir.

Elde edilen Dİ PQ-Analysis aracı [23] ile kullanılarak bağlam tabloları elde edilmesi sağlanmaktadır. Bu elde edilen bağlam tabloları PQ-TestGen aracına [22] verilerek istenilen optimal test dizileri elde edilir.



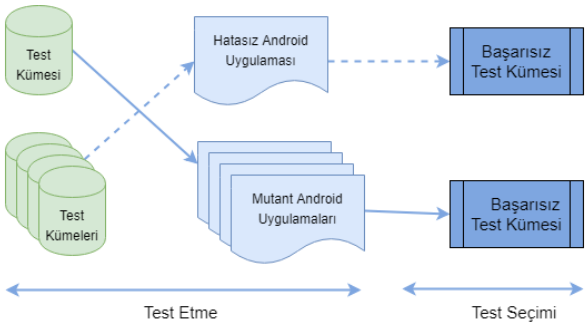
Şekil 3. Örnek SDM

3.2. Test Etme ve Test Seçimi

Bu aşamada test üretim sürecinden elde edilen test kümeleri çapraz olarak hatasız android uygulaması ve mutant android uygulamaları üzerinde koşulur. Mutant android uygulamaları, mutant modellere karşılık gelen kod tabanlı hata veya hatalar enjekte edilerek oluşturulur. Ardından test etme sürecinde elde edilen sonuçlar başarılı ve başarısız olma durumlarına göre test seçimi aşamasından geçirilir. Bu adımlar Şekil 4'te gösterilmektedir.

Burada görüldüğü üzere pozitif test aşamasında hatasız modelden elde edilen test kümesi mutant android uygulamalar üzerinde koşulur ve test seçim aşamasında

başarısız (fail) olan test dizileri seçilerek her bir mutanta karşılık gelen test kümeleri oluşturulur. Benzer şekilde, negatif test aşamasında mutant modellerden elde edilen test kümeleri hatasız android uygulamasında koşulur ve test seçim aşamasında yine başarısız (fail) olan test dizileri seçilerek her bir mutanta karşılık gelen test kümeleri oluşturulur. Burada açıklanan başarısız (fail) kelimesi aslında koşulan test dizisinin beklenen çıktıyı vermemesidir. Yani aslında aranan ve hata yakalama kabiliyeti olan test dizisidir. Başarılı (successful) olarak geçen test dizileri ise hata yakalama kabiliyeti olmayan test dizisidir. Sürecin sonunda elde ettiğimiz hata yakalama kabiliyeti olan test dizileri ideal test kümesini oluşturmuş olur.



Şekil 4. Test Etme ve Test Seçimi

Bu aşamalar sonucunda elde edilen pozitif ve negatif test kümeleri Goodenough ve Gerhart [19] tarafından ileri sürülen İdeal Test'in güvenilirlik (reliability) ve geçerlilik (validity) gereksinimlerini karşılamaktadır. Bu sağlama ile ilgili ispatlar ve detaylar [22] çalışmasında bulunabilir.

4 Örnek Durum

Bu kısımda öne sürülen melez yöntemin uygulanabilirliğini gösterebilmek açısından yapılan çalışmalar verilecektir. Öncelikle örnek durumun modeline ait test dizilerinin koşulabilmesi için bir test ortamı hazırlanmıştır. Bu test ortamı hazırlanırken Android uygulamalarında test yapılabilmesini sağlayan android testing support [26] yardımcı kütüphanesinden yararlanılmış, programlama dili olarak Java ve yazılım geliştirme ortamı olarak Android Studio IDE'si kullanılmıştır.

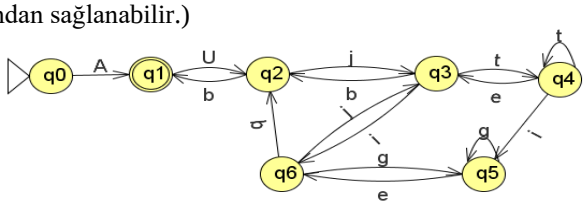
Model tabanlı test üretimi yapacağımız Android uygulaması olarak Telegram [5] kullanılacaktır. Telegram hız ve güvenlik odaklı bir mesajlaşma uygulamasıdır. Telegram projesinin kaynak kodu için Fdroid [4] üzerinde bulunan açık kaynak kodlu yazılım seçilmiştir.

4.1. Test Hazırlığı

Modelleme

İlk olarak Telegram mesajlaşma programına ait olacak şekilde JFLAP [24] programı kullanılarak bir SDM çizilmiştir. Bu model üzerinde her bir düğüm kullanıcının uğrayabileceği sayfaları her bir kenar ise bu sayfa geçişleri arasında kullanılabilecek olan GKA bileşenlerini göstermektedir. Örnek olarak; Şekil 5'de verilen SDM Telegram uygulamasının açılmasından başlayarak mesajlaşma alanına doğru giden bir modeli kapsamaktadır. (Bu model 1B, hatalı SDM'ler ise 1FBX ile

gösterilecektir. (Örnek verilen 1B SDM'nin (Şekil 5) daha kapsamlı hali istenirse yazarlar tarafından sağlanabilir.)



Şekil 5. 1B:Mesajlaşma alanına ait hatasız SDM

Bu modelde q0 uygulamanın başlangıç durumunu, q1 ise son durumunu temsil etmektedir. Büyük harf ile gösterilen kısımlar katmanlı bir SDM yapısı düşünüldüğünde üst katmanı, küçük harfler ise alt katmanda yer alan bileşenleri göstermektedir. SDM'de verilen harfler ve anlamları; A: Uygulama ana sayfa ekranı, U: Listedeki ilk kullanıcıyı seçme, b: Geriye tıklama, j: Metin yazma alanına tıklama, t: Metin yazma, e: Mesajı gönderme, i: Emoji ikonuna tıklama (Yüz ifadesi ikonu),g: Bir tane emoji seçme gibidir.

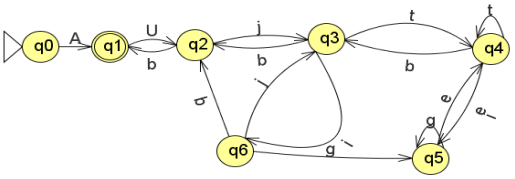
Mutasyonlar

Mutant modelin yaratılması için hatasız model kullanılır ve bu model üzerine örneğin beklenmeyen bir geçiş eklenir. Bu hata modeli ekstra geçiş tipi bir hatayı modellemektedir. Mutant modellerin oluşturulması model tabanlı mutasyon kısmını kapsamaktadır.

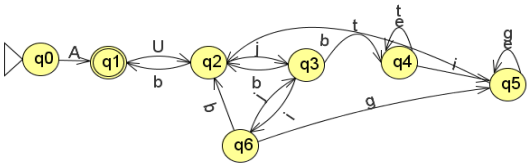
Önerilen yaklaşım çerçevesinde test koşumu için kod tabanlı mutasyona da ihtiyaç duyulmaktadır. Bu kod tabanlı mutasyonlar model tabanlı mutantlara karşılık gelmektedir. Bunun için hatasız uygulama üzerine hatalı kod enjekte işlemi yapılır ve mutant bir uygulama elde edilir. Enjekte edilen hata model tabanlı hata modeline göre ikinci veya yüksek dereceden mutasyonları gerektirebilmektedir.

Bu aşamada kullanacağımız mutasyonlar aşağıdaki gibidir;

- Uygulamanın gönder butonuna basıldığında gönderme yapmayıp emoji seçme ekranın açması (1FB1 ile gösterilecektir,bakınız Şekil 6 (a))
- Uygulamanın gönder butonuna basıldığında tepki vermemesi (1FB2 ile gösterilecektir, bakınız Şekil 6 (b))



(a)



(b)

Şekil 6. Mutantlar (a)1FB1: Gönderme işlevini yerine getirmeyip emoji açan hatalı SDM, (b) 1FB2: Gönderme işlevini yerine getirmeyip tepki vermeyen hatalı SDM

Dönüşümler

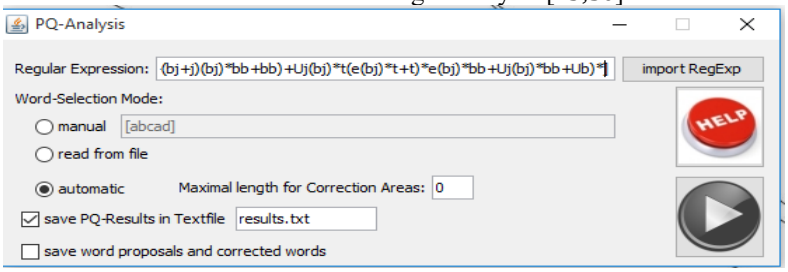
SDM'ler temel alınarak bunlara karşılık gelen Dİ'ler üretilecektir. Bu işlem JFLAP [24] vasıtası ile gerçekleştirilecektir. Dİ'ler kullanılarak ise test dizileri PQTestGen [22] aracı ile üretilecektir. Daha önce bahsedilen 1B modeline ait Dİ **Şekil 7**'deki gibidir.

A((Uj(bj)*t(e(bj)*t+t)*ig*e+Uj(bj)*t(e(bj)*t+t)*e(bj)*i+Uj(bj)*i)(((bj+j)(bj)*t(e(bj)*t+t)*i+g)g*e+(bj+j)(bj)*t(e(bj)*t+t)*e(bj)*i+(bj+j)(bj)*i)*((bj+j)(bj)*t(e(bj)*t+t)*e(bj)*bb+(bj+j)(bj)*bb+bb)+Uj(bj)*t(e(bj)*t+t)*e(bj)*bb+Uj(bj)*bb+Ub)*

Şekil 7. 1B modeline ait Düzenli İfade

4.2. Test Üretimi

Elde edilen Dİ'lerden PQ-Analysis aracı [23] (bakınız Şekil 8) ile kullanılarak bağlam tabloları elde edilecek ve bu aracın çıktısı PQTestGen [22] aracına verilerek optimal test dizileri elde edilecektir. PQ-Analysis aracı kullanılarak 1B ve 1FB1 modellerinin Dİ'leri kullanılarak ileri sağ ve ileri sol bağlam tabloları elde edilir. Bu tablolar ve bunlara ait test üretim süreci ile ilgili detaylar [29,30]'de bulunabilir.



Şekil 8. PQ-Analysis aracı

1B (hatasız) modeline ait optimal test dizileri Tablo 1 gibidir ve toplam 19 test dizisinden oluşmaktadır.1FB1 (hatalı) modeline ait optimal test dizileri Tablo 2 gibidir ve toplam 53 test dizisinden oluşmaktadır.

4.3. Test Etme

Üretilen test kümeleri Telegram [5] uygulaması üzerinde koşularak farklı yöntemlerle elde edilen test dizilerine ait sonuçlar toplanır ve elde edilen sonuçlar test kümeleri hata kapsama oranı açısından değerlendirilecektir.

Test etme sürecinin sonunda pozitif test için başarılı test kümesi negative test için başarısız test kümeleri oluşturularak ideal test kümeleri oluşturulmuştur. Bunun için pozitif test aşamasında hatasız modelden elde edilen test dizileri (bakınız Tablo 1) mutant uygulama (bu uygulama kod tabanlı hatalar enjekte edilerek oluşturulmuştur) üzerinde koşulur. Negatif test için ise mutant modelden elde edilen test dizileri (bakınız Tablo 2) hatasız uygulama üzerinde koşulur.

4.4. Test Seçimi

Pozitif ve negatif test etme aşamaları için elde edilen sonuçlar değerlendirilir. Bu aşamada Şekil 4'de verilen test seçimi kriterleri kullanılır. Bunlar hatasız modelden elde edilen test dizilerinin mutant uygulama üzerinde koşulması sonucu başarısız (test failed) dizilerinin seçilmesi ve mutant modelden elde edilen test dizilerinin hatasız

uygulama üzerinde koşulması sonucu başarısız (test failed) dizilerinin seçilmesidir. Böylece seçilen test dizilerinden pozitif ve negatif test kümeleri elde edilmiş olur.

Dikkat edilirse burada test etme ve test seçimi adımları sadece bir tek mutant (1FB1) için gerçekleştirilmiştir. Normalde negatif test kümeleri her bir mutant için ayrı ayrı oluşturulur.

Tablo 1. 1B hatasız modeline ait optimal test dizileri

AUjbb	AUjijbb	AUjttebb	AUjtebb	AUjijtebb	AUjttiebb	AUjtiebb
AUjijibb	AUjigggebb	AUjijtiebb	AUjigebb	AUjijttebb	A	AUjtttebb
AUjijttiebb	AUjibb	AUjiggebb	AUjijtttebb	AUb		

Tablo 2. 1FB1 hatalı modeline ait optimal test dizileri

AUjijtbbbUb	AUjibjbb	AUjijteebbb	AUjijtteebbb	AUjtteebbb	AUjigebbjibb	AUjibbUb
AUjiggebbb	AUjteebbjbb	AUjtegebbb	AUjteggebbb	AUjbbUb	AUjtiebbb	AUjijbjibb
AUb	AUjijttbbb	AUjibjibb	AUjijttiebbb	AUjijtbbb	AUjttbbb	AUjigggebbb
AUjijtegebbb	AUjibb	AUjbjbb	AUjijtbibb	A	AUjijtttbbb	AUjtbbjibb
AUjttiebbb	AUjbb	AUjigebbjbb	AUjijtiebbb	AUjijibb	AUjtigebbb	AUjtbbjbb
AUjigebibb	AUjtbibb	AUjijtbbjibb	AUjijbb	AUjteebibb	AUjijtigebbb	AUjteebbbUb
AUbUb	AUjtttbbb	AUjtbbb	AUjtbbbUb	AUjijbbUb	AUjteebbb	AUjigebbbUb
AUjteebbjibb	AUjigebbb	AUjijtbbjbb	AUjijbjbb			

5 Sonuçlar ve Tartışma

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 öne sürülen yaklaşımın gerçekleştirilmesi için ön çalışmalar Telegram isimli android uygulaması GKA'sı için yapılmıştır. Bu yaklaşımı değerlendirmek için model tabanlı test dizisi üretimine olanak sağlayan MaTeLo aracı [27] seçilmiş ve gerekli test dizileri bu araç vasıtası ile üretilmiş, ardından bu testler ilgili uygulamalar üzerinde koşulmuş ve sonuçlar toplanmıştır. Ancak karşılaştırma için gerek zaman gerekse bildirinin sayfa sınırından ötürü ilgili bölümler çıkarılmıştır. Ancak kabaca deneysel çalışmalara göre PQ-TestGen aracı ile elde edilen test dizileri hata kapsama oranına göre MaTeLo aracından daha iyi sonuçlar vermektedir.

Öne sürülen yaklaşımın getirdiği katkılar özetlenecek olursa;

- Bütünsel ve mutasyon testini kullanan melez bir yaklaşımın öne sürülmesi,
- Bu melez yöntem için ideal test adımlarının tanımlanması,
- Öne sürülen yaklaşımın bir örnek durum üzerinde gerçeklenmesi.

Öne sürülen yaklaşımın avantajları ise;

- Bütünsel ve mutasyon testinin özelliklerini birleştirmesi,

- İdeal test sayesinde system içerisindeki hataların varlığının (presence) ve yokluğunun (absence), model ölçeğinde karşılaştırmalı ve deneysel çalışmalar çerçevesinde test edilebilmesi,
- Yöntemin SDM ile modellenebilen başka sistemlerin testi içinde uygunluğu,
- Model olarak Dİ kullanması sebebi ile daha soyut ve sıkı test dizilerinin üretimine olanak sağlaması.

Öne sürülen yaklaşımın dezavantajları;

- Mutasyon testinin doğası gerektirdiği ciddi masraf, ancak bu test stratejisinin güvenilirliği düşünüldüğünde kullanıcıların tercihine göre göz ardı edilebilmektedir,
- SDM ve Dİ dönüşümleri arasında gerçekleşen ekstra masraf ise bir diğer dezavantaj olarak sayılabilir, ancak bu da SDM tabanlı yaklaşımların masrafları ile kıyaslandığında tercih edilebilir ölçekte kalmaktadır.

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 gelmeye 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 ileriki çalışmalar olarak sayılabilir.

Teşekkür

Yazarlar çok değerli yorumlarından ve düzeltme önerilerinden dolayı anonim hakemlere çok teşekkür ederler.

Referanslar

1. Google Play, https://play.google.com/store, son erişim 2018/03/11.
2. Appbrain Statistics, http://www.appbrain.com/stats/, son erişim 2018/03/11.
3. Google, Test Your App., https://developer.android.com/studio/test/index.html, son erişim 2018/03/11
4. F-Droid, https://f-droid.org/en/, son erişim 2018/03/11.
5. Telegram, https://fdroid.org/packages/org.telegram.messenger/, son erişim 2018/03/11.
6. Monkey, https://developer.android.com/studio/test/monkey, son erişim 2018/03/11.
7. Aravind Machiry, Rohan Tahiliani, and Mayur Naik. 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.
8. Chow, Tsun S. "Testing software design modeled by finite-state machines." IEEE transactions on software engineering 3 (1978): 178-187.
9. Belli, Fevzi. "Finite state testing and analysis of graphical user interfaces." Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on. IEEE, 2001.
10. Memon, Atif M., Mary Lou Soffa, and Martha E. Pollack. "Coverage criteria for GUI testing." ACM SIGSOFT Software Engineering Notes 26.5 (2001): 256-267.
11. Utting, Mark, and Bruno Legeard. Practical model-based testing: a tools approach. Morgan Kaufmann, 2010.

12. Kramer, Anne, and Bruno Legeard. Model-Based Testing Essentials-Guide to the ISTQB Certified Model-Based Tester: Foundation Level. John Wiley & Sons, 2016.
13. S. R. Choudhary, A. Gorla, A. Orso, "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.
14. D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, A. M. Memon, "Mobiguitar: Automated model-based testing of mobile apps", IEEE Software, vol. 32, no. 5, pp. 53-59, Sept 2015.
15. Young-Min Baek, Doo-Hwan Bae, 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.
16. Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, Zhendong Su, 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.
17. DeMillo, Richard A., Richard J. Lipton, and Frederick G. Sayward. "Hints on test data selection: Help for the practicing programmer." Computer 11.4 (1978): 34-41.
18. Deng, Lin, Jeff Offutt, and David Samudio. "Is Mutation Analysis Effective at Testing Android Apps?." Software Quality, Reliability and Security (QRS), 2017 IEEE International Conference on. IEEE, 2017.
19. Goodenough, J.B., Gerhart, S.L.: Toward a theory of test data selection. IEEE Transactions on software Engineering vol. SE-1, 156–173 (1975).
20. Howden, W.E.: Reliability of the path analysis testing strategy. IEEE Transactions on Software Engineering vol.3, 208-215 (1976).
21. Bougé, L.: A contribution to the theory of program testing. Theoretical Computer Science vol. 37, 151-181 (1985).
22. 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.
23. PQ-Analysis aracı, http://download.ivknet.de/, son erişim 2018/03/11.
24. JFLAP aracı, http://www.jflap.org/, son erişim 2018/03/11.
25. UI Automator, https://developer.android.com/training/testing/ui-automator, son erişim 2018/03/11.
26. Android Testing Support, https://developer.android.com/training/testing/, son erişim 2018/03/11.
27. MaTeLo aracı, http://www.all4tec.com/matelo-generation-strategies, son erişim 2018/03/11.
28. Belli, Fevzi, Christof J. Budnik, Axel Hollmann, Tugkan Tuglular, and W. Eric Wong. "Model-based mutation testing—approach and case studies." Science of Computer Programming 120 (2016): 25-48.
29. Kılınççeker, O., Belli, F., Coverage criteria for testing graphical user interfaces based on regular expressions (in Turkish), Proc. XI. Turkish National Software Engineering Symposium - VII. Ulusal Yazılım Mühendisliği Sempozyumu - UYMS 2017), CEUR-WS, pp. 332-343.
30. F. Belli, Regular Expressions for Fault Handling in Sequential Circuits, Proc. ARCS 2015 - 28th International Conference on Architecture of Computing Systems, 11th Workshop on Dependability and Fault Tolerance (VERFE'15), 2015.