Android Uygulamaları için Kötü Huylu Test Yaratımı

Yavuz Köroğlu ve Alper Şen

Boğaziçi Üniversitesi, Bilgisayar Mühendisliği Bölümü
{yavuz.koroglu,alper.sen}@boun.edu.tr

Özet. Mobil uygulamaların Grafik Kullanıcı Arayüzü (GKA) testi literatürde önem kazanmış bir konudur. Android GKA uygulamaları için otomatik test yaratımı üzerine yoğunlaşan birçok çalışma olmasına rağmen, bütün bu çalışmalar Test Altındaki Uygulamanın (TAU) derinliklerini anlamlı ve iyi huylu test yaratımı yoluyla araştırmayı amaçlamaktadır. Bu çalışmamızda standart test yaratımı araçlarının üzerine inşa edilen tam otomatik Kötü Huylu Test Yaratımı (KHTY) aracını önermekteyiz. KHTY iyi huylu test örneklerinden (test case) var olan olayları değiştirerek ve yeni olaylar ekleyerek kötü huylu test örnekleri yaratmaktadır. Bu çalışmamızda, KHTY'yi en yeni test yaratım yaklaşımı olan QBE (QLearning-Based Exploration) üzerine geliştirdik. İyi bilinen F-Droid uygulamalarından rastgele seçilmiş 100 uygulama üzerinde deneyler gerçekleştirdik. KHTY'nin verili bir zaman bütçesinde yeni çökmeleri (crash) tespit etmekte etkili olduğunu gösterdik.

Anahtar Kelimeler: Mobil Uygulama Testi, Grafiksel Kullanıcı Arayüzü Testi, Otomatik Test Yaratımı, Negatif Test, Test Örneği Mutasyonu

Bad Behaving Test Generation for Android Applications

Yavuz Köroğlu and Alper Şen

Bogazici University, Department of Computer Engineering
{yavuz.koroglu,alper.sen}@boun.edu.tr

Abstract. Graphical User Interface (GUI) testing of mobile applications has been an important topic in the literature. Although there are many studies that focus on automated test generation for Android GUI applications, all these studies aim to explore depths of the Application Under Test (AUT) by generating meaningful and well-behaving tests. In this study we propose a fully automated tool, Bad Behaving Test Generator (BBTG), that builds on top of standard test generation tools. BBTG generates bad-behaving test cases from the well-behaving test cases by modifying existing events and adding new events. In this study, we develop BBTG on top of the state-of-the-art test generation approach, QBE (QLearning-Based Exploration). We perform experiments on 100 AUTs randomly selected from the commonly known F-Droid applications. We show that BBTG is effective at detecting novel crashes in a given time budget.

Keywords: Mobile Application Testing, GUI Testing, Automated Test Generation, Negative Testing, Test Case Mutation

1 Giriş

Akıllı telefon kullanıcılarının sayısının 2019 yılında 5 milyarı geçmesi beklenmektedir [8]. Akıllı telefon pazarında Android %85.9 ile birinci sıradadır [9]. Çalışmalar Android'de geçirilen zamanın %90'ının Android uygulamalarına harcandığını göstermektedir [4]. Android uygulamaları, Grafiksel Kullanıcı Arayüzü (GKA) üzerinden GKA eylemleri (actions) ile girdi alan sistemlerdir.

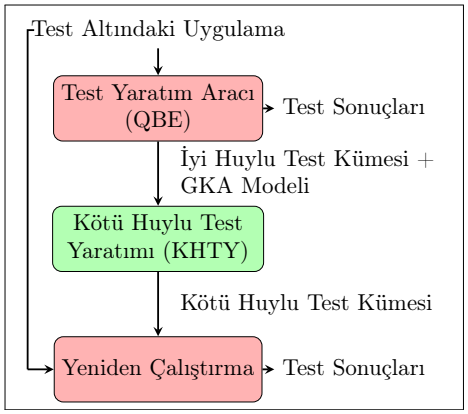
Android GKA için tam otomatik test yaratımı amacıyla A^{3}E [2], SwiftHand [5], PUMA [11], DynoDroid [13], Sapienz [15], ve QBE [12] araçları geliştirilmiştir. Bu araçlar Test Altındaki Uygulamaları (TAU) dinamik olarak icra ederek uygulamanın derinliklerini gezen testler yaratmaktadır. Bu araçların yarattıkları testler *iyi huylu*dur. İyi huylu testler gezdikleri GKA durumlarında beklenen GKA eylemlerini gerçekleştirmekteyi hedefler. Bu sayede iyi huylu testler TAU'nun derinliklerini gezebilmektedir.

Otomatik test yaratımı araçlarının performanslarını değerlendirmek için tespit edebildikleri farklı çökme (crash) sayılarını kıyaslamak çok kullanılan bir yöntemdir [6]. Otomatik test yaratımı araçlarının çökme tespitlerini iyileştirmek amacıyla standart test yaratımı araçlarının üzerine inşa edilen tam otomatik Kötü Huylu Test Yaratımı (KHTY) aracını önermekteyiz. Bu aracın temel varsayımı kötü huylu GKA eylemlerinin iyi huylulara göre daha fazla çökme tespit etme potansiyeli olmasıdır. Kötü huylu GKA eylemlerinin ait oldukları GKA durumlarına (state) ulaştıktan sonra çalıştırılmaları gerekmektedir. Bu yüzden KHTY, iyi huylu GKA eylemleriyle TAU'nun derinliklerini gezme amaçlı tam otomatik olarak yaratılmış test örneklerindeki (test case) GKA eylemlerini değiştirerek ve bu test örneklerine yeni GKA eylemleri ekleyerek kötü huylu test örnekleri yaratmaktadır. Böylece KHTY, uygulamanın derinliklerindeki GKA durumlarında kötü huylu GKA eylemleri çalıştırabilmektedir.

Bu çalışmadaki katkılarımız aşağıdaki gibidir.

1. *Kötü Huylu Test Yaratım İşleçleri (Operators).* Yeni çökmelerin tespiti amaçlı olarak test örnekleri (test case) üzerinde tanımlı altı işleç (operator) önermekteyiz. Yazı Değişimi işleci dışındaki bütün işleçler tamamen yenidir.
2. *Köyü Huylu Test Yaratım Algoritması.* İşleçlerimizin iyi huylu testlere uygulanarak kötü huylu testler yaratılmasını sağlayan yeni bir algoritma geliştirmekteyiz.
3. *Vaka İncelemeleri.* Beş ayrı vaka incelemesi üzerinden geliştirdiğimiz işleçlerin gerçek dünyadaki karşılıklarını göstermekteyiz.
4. *Deneyler.* F-Droid [10] uygulamalarından rastgele seçilmiş 100 uygulama üzerinde gerçekleştirdiğimiz deneylerin sonuçlarına bakarak KHTY'nin verili bir zaman bütçesinde yeni çökmeleri tespit etmekte etkili olduğunu görmekteyiz.

Şekil 1, KHTY aracının akış çizelgesini göstermektedir. KHTY aracını literatürdeki en yeni test yaratım yaklaşımı olan QBE [12] üzerine geliştirdik. QBE, makine öğrenmesi tabanlı bir tam otomatik test yaratım aracıdır. Bu akışta ilk olarak QBE yardımıyla Test Altındaki Uygulama (TAU) için bir İyi Huylu Test Kümesi (test suite) yaratmaktayız. QBE test yaratımı sırasında TAU'nun Genişletilmiş Etiketli Durum-Geçiş Sistemi (GEDGS, Extended Labeled Transition System [2]) adı verilen bir sonlu durum-geçiş modelini çıkarmaktadır. KHTY uygulamanın modeline bakarak İyi Huylu Test Kümesinin test örneklerindeki GKA eylemlerini değiştirip bunlara yeni kötü huylu GKA eylemleri eklemekte ve kötü



Şekil 1: KHTY Akış Çizelgesi

huylu test örnekleri oluşturmaktadır. Kötü huylu test örnekleri Kötü Huylu Test Kümesini oluşturmaktadır. Son olarak KHTY, Kötü Huylu Test Kümesini TAU üzerinde yeniden çalıştırarak Test Sonuçlarını toplamaktadır.

Bildirimizin geri kalanı yedi bölüme ayrılmaktadır. Çalışmamızın anlaşılması için gerekli temel bilgiler Bölüm 2'de toplanmıştır. KHTY aracının nasıl çalıştığı Bölüm 3 ile anlatılmıştır. Bölüm 4'te vaka incelemeleri yaparak KHTY aracında kullanılan işleçlerin gerçek hayattaki karşılıkları gözlemlenmiştir. KHTY aracının diğer araçlarla kıyaslanması ve çökme tespitine katkıları Bölüm 5'te ele alınmıştır. Çalışmamız ile ilgili akademik literatür taraması Bölüm 6'de yer almaktadır. Son olarak KHTY ile alınan sonuçlar ve gelecekte yapacağımız araştırmalar Bölüm 7 ile özetlenmiştir.

2 Temel Bilgiler

2.1 Android Grafiksel Kullanıcı Arayüzü (GKA)

Android GKA, aktivite (activity) ve olay (event) tabanlıdır. Aktiviteler GKA bileşenlerinden (widget) oluşur. Her bir GKA bileşeni (örn. düğme veya metin girdisi), piksel cinsinden bileşenin sınır koordinatlarını (x_1, y_1, x_2, y_2) tanımlayan ve kullanıcının bileşenle hangi GKA eylemleri (GUI action) aracılığıyla etkileşime girebileceğini belirten birtakım özelliklere sahiptir. Bu özelliklere, *tür*, *etkin* (enabled), *tıklanabilir* (clickable), *uzun tıklanabilir* (longclickable), *kaydırılabilir* (scrollable), ve *şifre* (password) örnek olarak verilebilir.

Bir kullanıcı, Android sistemi ile GKA bileşenleri üzerinden *olaylar* (events) aracılığı ile etkileşime girer. Olayları temel olarak iki kategoriye ayırabiliriz, *sistem olayları* ve *GKA eylemleri* (GUI actions). Tipik olarak literatürde kullanılanlardan daha kapsamlı bir GKA eylemleri listesini Tablo 1'de göstermekteyiz. Eylemler üç kategoriden oluşmaktadır; bağlamsal olmayan (non-contextual), bağlamsal (contextual) ve özel (special). Bağlamsal olmayan eylemler kullanıcı hareketleriyle tetiklenen eylemlerdir. *Tıklama* ve *uzun tıklama* eylemleri, tıklanılacak x ve y koordinatları olmak üzere iki parametre alırlar. *Metin girdisi* eylemi x, y koordinatları ve girilecek metni belirten üç parametre alır. *Kaydırma* eylemi beş parametre alır; ilk dört parametre başlangıç ve bitiş koordinatlarını belirtirken, beşinci parametre ise kaydırma hızını ayarlamak için

Tablo 1: GKA Eylemler Listesi

Bağlamsal olmayan	Param1	Param2	Param3	Param4	Param5
tıklama	x	y	-	-	-
uzuntıklama	x	y	-	-	-
metin	x	y	yazı	-	-
kaydırma	x1	y1	x2	y2	süre
menü	-	-	-	-	-
geri	-	-	-	-	-

Bağlamsal	Parametre
bağlanırlık	açık/kapalı/değiştir
bluetooth	açık/kapalı/değştir
konum	gps/gps&ağ/kapalı/değiştir
uçuşmodu	açık/kapalı/değiştir
uyku	açık/kapalı/değiştir

Özel	Param1	Param2	Param3	Param4	Param5
yenidenbaşlatmak	paket	aktivite	-	-	-

kullanılır. *Menü* ve *Geri* eylemleri mobil cihaz üzerindeki ilgili düğmelerin basılmasını temsil eden eylemlerdirler ve herhangi bir parametre almazlar. Bağlamsal eylemler, kullanıcının Test Altındaki Uygulamanın (TAU) bağlamsal durumunu değiştirdiği eylemleri ifade eder. Mobil cihazın global niteliklerinin birleşimi (internet bağlanırlığı, bluetooth durumu, konum, uçak modu ve uyku modu) uygulamanın o anki bağlamsal durumunu oluşturur. *Bağlanırlık* eylemi mobil cihazın internet bağlanırlığını ayarlar (Wi-Fi veya mobil veri). *Bluetooth durumu*, *konum* ve *uçuş modu* nitelikleri açık ve anlaşılırdır. *Uyku* eylemi mobil cihazı güç düğmesine basarak uyku moduna alan veya uyku modundan çıkaran eylemdir. *Uyku* eylemi test edilen uygulamayı duraklatmak ve devam ettirmek için kullanılır. Özel (special) eylem olarak da uygulamayı yeniden yükleyip başlatmaya yarayan *yenidenbaşlatmak* (reinitialize) bulunmaktadır. Sistem olayları sistem tarafından oluşturulan olaylardır; örneğin, *pil seviyesi* olayları, *SMS almak*, ve *saat/süreölçer* olayları gibi.

2.2 Android GKA için Test Durumları

Bir GKA durumu veya kısaca bir *durum* v dört öğenin bitiştirilmesinden (concatenation) oluşur: (1) Paket adı, (2) aktivite adı, (3) bağlamsal durum, ve (4) GKA bileşenleri. Her durum v için GKA bileşenlerinden elde edilebilen bir etkin eylemler kümesi $\lambda(v)$ vardır. Bir GKA eylemi veya kısaca *eylem* $z \in Z$, ancak ve ancak bir v durumunun GKA bileşenlerinden en az biri ile ilişkilendirilebiliyorsa, z eylemi v durumunda etkindir, kısaca $z \in \lambda(v)$, denilir. Bir *geçiş*, (başlangıç-durumu, bitiş-durumu, eylem, süre) olacak şekilde dörtlü değişkenler grubu (tuple) olarak tanımlanır. Bir yürütme izi (execution trace) veya kısaca *iz (trace)* t, bir geçişler dizisidir. Örneğin n uzunluğa sahip bir iz aşağıdaki gibi olabilir.

$$t = (v_1, v_2, z_1, d_1), (v_2, v_3, z_2, d_2), \ldots, (v_n, v_{n+1}, z_n, d_n)$$

Eğer bir iz t'nin ilk durumu, TAU başlatıldığı andaki GKA durumu olan ilk durum v_0 ile aynıysa, t bir *test örneği*dir (test case). Test örneklerini içeren kümelere *test kümesi* (test suite), kısaca *TK* denilir.

Algoritma 1 Kötü Huylu Test Yaratımı (KHTY) Algoritması

Girdiler:

TK : Test Kümesi

X : Yeni Test Kümesinin Süre Limiti

Δ: Kötü Huylu Test Yaratım İşleçleri Kümesi

Çıktılar:

TK' : Yeni Test Kümesi

1: $TK' \leftarrow \varnothing$
2: $x \leftarrow 0$
3: **Tekrarla**
4: $t \leftarrow$ rastgele $t \in TK$ ▷ Rastgele bir test örneği seç
5: **Tekrarla**
6: $\delta \leftarrow$ rastgele $\delta \in \Delta$ ▷ Rastgele bir işleç seç
7: **Çıkış Koşulu:** $t \neq \delta(t)$ ▷ Test örneği değişene kadar tekrarla
8: $t' \leftarrow \delta(t)$ ▷ İşleci test örneğine uygula
9: $TK' \leftarrow TK' \cup \{t'\}$ ▷ Yeni test örneğini kümeye ekle
10: $x \leftarrow x + \sum_{(v_s,v_e,z,d)\in t'} d$ ▷ Toplam süreyi hesapla
11: **Çıkış Koşulu:** $x > X$ ▷ Toplam süre limiti aşana kadar tekrarla

3 Kötü Huylu Test Yaratımı

Bu bölümde Kötü Huylu Test Yaratımı (KHTY) aracımızın nasıl çalıştığını açıklamaktayız. KHTY'nin temelinde *Kötü Huylu Test Yaratım işleçleri*, ya da kısaca *işleçler* vardır. İşleçler bir test örneği alarak yeni test örnekleri oluşturan fonksiyonlardır ve $\delta(t) = t'$ şeklinde gösterilirler. Bu bölümde önce işleçleri kullanarak kötü huylu test kümesi yaratan algoritmamızı açıklamaktayız. Daha sonra kısaca bu çalışmada kullandığımız işleçleri anlatmakta ve son olarak da KHTY icrasını ufak bir örnekle açıklamaktayız.

3.1 Algoritma

KHTY aracının temel prosedürü Algoritma 1 üzerinden anlaşılabilir. İyi Huylu Test Kümesi (TK), Kötü Huylu Test Kümesi (TK') için ayrılabilecek azami süre (X), ve TK'nın test örneklerinden kötü huylu test örnekleri yaratacak işleçlerin bir kümesi (Δ) bu algoritmaya girdi olarak verilir. KHTY çıktı olarak icrası azami süre (X) kadar vakit alacak olan bir Kötü Huylu Test Kümesi (TK') döner.

Algoritma 1 ilk olarak Kötü Huylu Test Kümesine (TK') boş küme ($\varnothing$) atar (bkz. satır 1). Dolayısıyla Kötü Huylu Test Kümesinin (TK') şu anki icra süresi (x) sıfırdır (bkz. satır 2). Daha sonra KHTY, İyi Huylu Test Kümesinden (TK) rastgele bir test örneği (t) ve verili işleçler kümesinden (Δ) bu test örneğini değiştirecek rastgele bir işleç seçer (bkz. satır aralığı 4-7). KHTY seçilmiş işleci (δ) seçilmiş test örneğine (t) uygular ve yeni bir kötü huylu test örneği (t') elde eder (bkz. satır 8). KHTY yeni test örneğini (t') Kötü Huylu Test Kümesine (TK') ekler (bkz. satır 9). Böylece Kötü Huylu Test Kümesinin (TK') icra süresi (x) yeni eklenen test örneğinin (t') icra süresi kadar artmış olur (bkz. satır 10).

KHTY Kötü Huylu Test Kümesinin (TK') icra süresi (x) verili limiti (X) aşana kadar test ekleme işlemini tekrarlar (bkz. satır 11).

3.2 İşleçler (Operators)

Bu bölümde Algoritma 1'de kullandığımız işleç kümesini (Δ) anlatmaktayız. Bu çalışmamızda Android uygulamaları üzerindeki gözlemlerimize ve Android hata örgeleri (crash patterns) üzerine yapılmış bir çalışmaya [3] dayanarak altı adet işleç geliştirdik.

1. **Döngü Stresleme ($\delta_{\mathbf{DS}}$) :** Bir test örneği içindeki bazı geçişlerin başlangıç ve sonlanış durumları aynı olabilir ($v_i = v_{i+1}$). Döngü Stresleme (δ_{DS}) işleci verili test örneğindeki döngü içeren geçişleri birden çok kere tekrarlayarak döngüyü stresleyen bir kötü huylu test örneği elde eder.
2. **Durdur-Başlat ($\delta_{\mathbf{DB}}$) :** Android uygulamaları herhangi bir anda arkaplana atılıp (durdur) geri çağırabilir (başlat). Durdur-Başlat (δ_{DB}) işleci verili test örneğinin her iki geçişi arasına *"uyku kapa"* ve *"uyku aç"* eylemleri koyarak bu işlevi suistimal eden bir kötü huylu test örneği elde eder.
3. **Yazı Değiştirme ($\delta_{\mathbf{YD}}$) :** Verili iyi huylu test örneğinde yazı girdileri varsa bu girdilerin beklenen, anlamlı girdiler olduğunu varsaymaktayız. Yazı Değiştirme (δ_{YD}) işleci verili test örneğindeki yazı girdilerini uzun ve anlamsız yazılarla, özel karakterlerle, veya boş yazı ile değiştirerek bir kötü huylu test örneği elde eder.
4. **Bağlamsal Durum Değiştirme ($\delta_{\mathbf{BDD}}$):** Verili iyi huylu test örneğinin uygulamanın işlevini düzgünce yerine getirebileceği bir bağlamsal durumda (bağlanırlık, konum, vb.) icra edildiğini varsaymaktayız. Bağlamsal Durum Değiştirme (δ_{BDD}) işleci verili test örneğine rastgele bağlamsal durum değiştirme eylemleri ekleyerek bir kötü huylu test örneği elde eder.
5. **Beklemeleri Kaldırma ($\delta_{\mathbf{BK}}$) :** Verili iyi huylu test örneğine ait bütün geçiş sürelerinin uygulamayı çalıştırdığımız aygıttan yanıt alabileceğimiz kadar uzun süreler olduğunu varsaymaktayız. Beklemeleri Kaldırma (δ_{BK}) işleci bu süreleri sıfırlayarak bir kötü huylu test örneği elde eder.
6. **Hızla Kaydırma ($\delta_{\mathbf{HK}}$) :** *Kaydırma* eyleminin kendi ayrı *süre* parametresi bulunmaktadır. Bu parametre ekrana dokunmaya başlayıp dokunmayı bırakana kadar geçecek süreyi tanımlar. Hızla Kaydırma (δ_{HK}) işleci verili test örneğindeki kaydırma sürelerini sıfırlayarak aşırı hızlı kaydırma eylemlerine sebep olur ve böylece bir kötü huylu test örneği elde eder.

3.3 Örnek İcra

Bu bölümde Şekil 2 üzerinden KHTY'nin örnek bir icrasını açıklamaktayız. Şekil 2a, KHTY algoritmasına bu örnekte verdiğimiz İyi Huylu Test Kümesini (TK) göstermektedir. Örneğin basit olması için bu test kümesine sadece tek bir tane test örneği koyduk ama bundan daha fazla test örneği olabilirdi. Test örneklerindeki v1, v2, v3 gibi ifadeler GKA durumlarını göstermektedir. '_' ifadesi herhangi bir durumu (don't care state) ifade etmek için kullanılmıştır. Şekil 2b, KHTY algoritması yardımıyla yaratılmış bir dakika azami süreli ($X = 60$) Kötü Huylu Test Kümesini (TK') göstermektedir. Kötü Huylu Test Kümesi, geçişlerinin toplam süresi 59 saniye olan Kötü Huylu 1 ve Kötü Huylu 2 adında iki test örneğinden oluşmaktadır. KHTY bu iki test örneği sırasıyla Döngü Stresleme (δ_{DS}) ve Durdur-Başlat (δ_{DB}) işleçleriyle oluşturmuştur. İki kötü huylu test de dokuzuncu adımda çökme tespit etmiştir.

	İyi Huylu			
1	_	v1	başlat	11
2	v1	v1	geri	0
3	v1	v2	tıkla	2
4	v2	v1	geri	1
5	v1	v3	menü	3

(a) İyi Huylu TK

	Kötü Huylu 1			
1	_	v1	başlat	11
2	**v1**	**v1**	**geri**	**1**
3	**v1**	**v1**	**geri**	**1**
4	**v1**	**v1**	**geri**	**1**
5	**v1**	**v1**	**geri**	**1**
6	**v1**	**v1**	**geri**	**1**
7	**v1**	**v1**	**geri**	**1**
8	**v1**	**v1**	**geri**	**1**
9	**v1**	**v1**	**geri**	**1**
10	**v1**	**v1**	**geri**	**1**
11	v1	v1	geri	0
12	v1	v2	tıkla	2
13	v2	v1	geri	1
14	v1	v3	menü	3

	Kötü Huylu 2			
1	_	v1	başlat	11
2	**v1**	**_**	**uyku kapa**	**2**
3	**_**	**v1**	**uyku aç**	**2**
4	v1	v1	geri	0
5	**v1**	**_**	**uyku kapa**	**2**
6	**_**	**v1**	**uyku aç**	**2**
7	v1	v2	tıkla	2
8	**v2**	**_**	**uyku kapa**	**2**
9	**_**	**v2**	**uyku aç**	**2**
10	v2	v1	geri	1
11	**v1**	**_**	**uyku kapa**	**2**
12	**_**	**v1**	**uyku aç**	**2**
13	v1	v3	menü	3

(b) Kötü Huylu Test Kümesi (TK')

Şekil 2: Örnek KHTY İcrası

4 Vaka İncelemeleri (Case Studies)

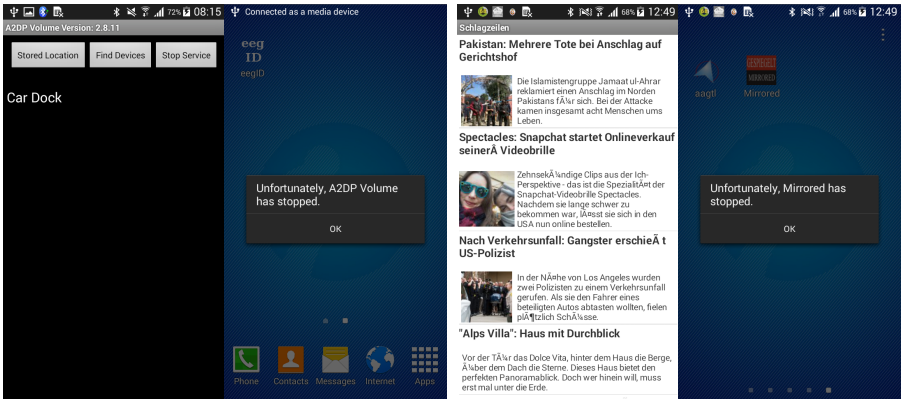
Bu bölümde işleçlerimizin gerçek dünyadaki karşılıklarını görmek amacıyla F-Droid uygulamaları üzerinde beş ayrı vaka incelemesi gerçekleştirmekteyiz. Bu incelemeler işleçlerimizi geliştirmemize yardımcı olmaktadır.

Vaka 1 Şekil 3a, *a2dpVol* uygulamasının çökme içeren bir aktivitesini göstermektedir. Bu çökmenin gözlemlenebilmesi için normalde kapalı olan *bluetooth* özelliğinin açılması gerekmektedir. KHTY, Bağlamsal Durum Değiştirme (δ_{BDD}) işleci yardımıyla test örneği içine *"bluetooth değiştir"* eylemi yerleştirerek uygulamanın çökmesine sebep olmaktadır.

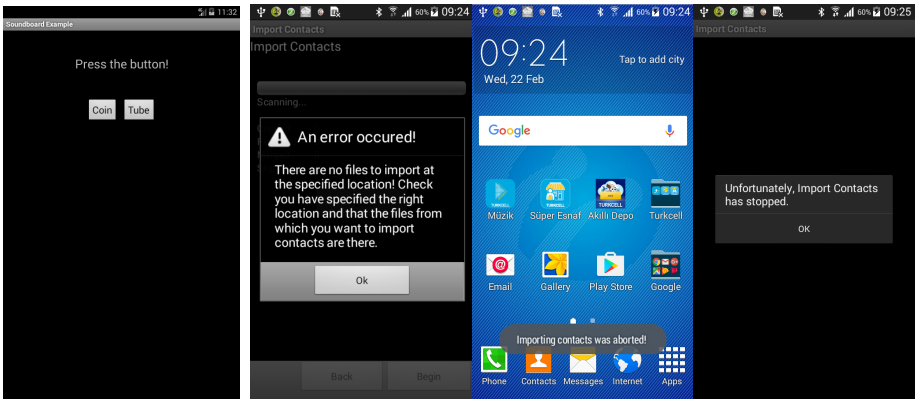
Vaka 2 Şekil 3b, *Mirrored* uygulamasının çökme içeren bir aktivitesini göstermektedir. Mirrored bir haber uygulamasıdır. Bu uygulama soldaki ekranda görüldüğü gibi bir haber listesi açmaktadır. Hızla Kaydırma (δ_{HK}) ve Beklemeleri Kaldırma (δ_{BK}) işleçleri ile yaratılmış kötü huylu testler bu uygulamanın çökmesine neden olmuştur.

Vaka 3 Şekil 3c, *SoundBoard* uygulamasının çökme içeren bir aktivitesini göstermektedir. *coin* ve *tube* tuşları uygulamanın GKA durumunu değiştirmemekte, sadece ses çıkarmaya yarayan AudioFlinger kütüphanesini çağırmaktadır. QBE bu tuşlara tıklayan ama çökme tespit etmeyen test örnekleri yaratmaktadır. KHTY, Döngü Stresleme (δ_{DS}) işlecini kullanarak kütüphanenin uygulama tarafından beklenmeyen ölümcül bir istisna (unhandled fatal exception) döndürmesine sebep olmaktadır.

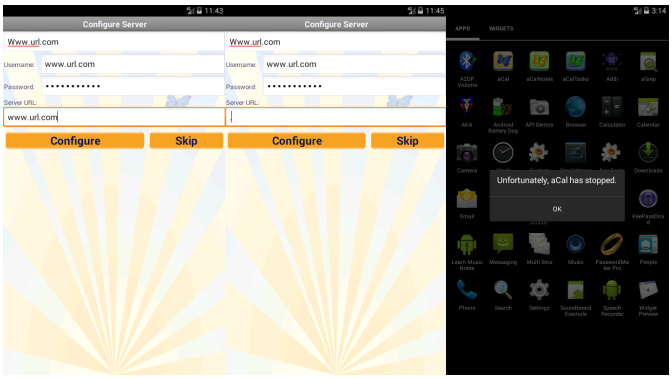
Vaka 4 Şekil 3d, *importcontacts* uygulamasının çökme içeren bir aktivitesini göstermektedir. Bu uygulama, kişi kayıtlarının dışarıdan yüklenemediği durumları en soldaki ekranda görüldüğü gibi çökmeden kurtarabilmektedir. Ama tam bu kurtarma sırasında Durdur-Başlat (δ_{DB}) işleci yardımıyla uygulamayı durdurup tekrar uyandırdığımızda en sağdaki hata ekranı çıkmakta ve uygulama çökmektedir.



(a) Vaka 1 (b) Vaka 2



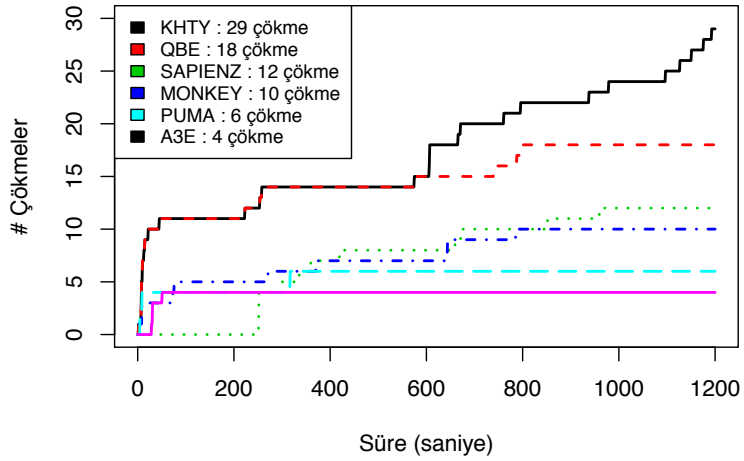
(c) Vaka 3 (d) Vaka 4



(e) Vaka 5

Şekil 3: Vaka İncelemeleri

Vaka 5 Şekil 3e, *aCal* uygulamasının çökme içeren bir aktivitesini göstermektedir. QBE internet adresi bekleyen en alttaki çubuğa internet adresi yazım



Şekil 4: Süreye bağlı Olarak Tespit Edilen Toplam Farklı Çökme Sayısı

kurallarına uygun bir adres girmektedir. KHTY, Yazı Değiştirme (δ_{YD}) işleci yardımıyla buraya beklenmeyen bir yazı girerek çökmeye sebep olmaktadır.

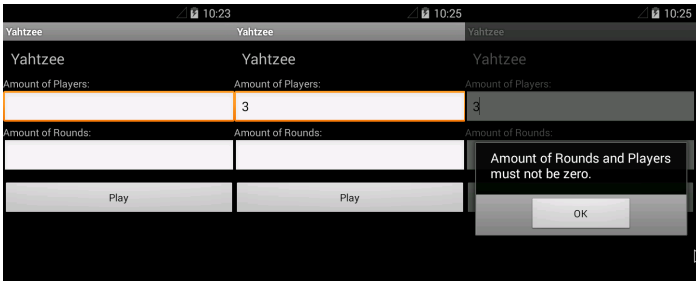
5 Deneyler

F-Droid uygulamaları [10] arasından deneylerimiz için 100 tane uygulamayı rastgele seçip indirdik. Vaka incelemelerimiz için kullandığımız uygulamaları bu listenin dışında bıraktık. Deneylerimizi gerçekleştirebilmek için 7 adet Android 4.4.r5 x86 VirtualBox sanal makinası kurduk. QBE, Sapienz, PUMA, Monkey, ve A^{3}E araçlarının her birini 100 uygulamanın her biri ile 20'şer dakika çalıştırdık. Sonra KHTY aracını çalıştırabilmek için ilk 10 dakika QBE ile test yaratımı gerçekleştirip kalan 10 dakikada ise azami 10 dakika süreli Kötü Huylu Test Kümesi yaratıp bu test kümesindeki testleri çalıştırdık.

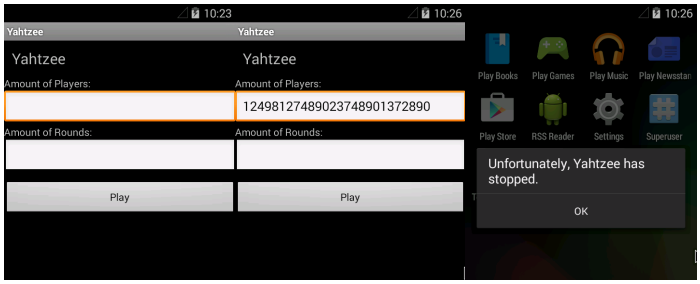
Şekil 4, süreye bağlı olarak tespit edilen toplam farklı çökme sayısını göstermektedir. Çökmelerin farklılığını Android sistem kayıtlarında rapor edilen yığın izlerini (stack trace) karşılaştırarak tespit ettik. Test yaratım araçlarının bazılarında rastgelelik bulunduğundan deneylerimizi beş defa tekrarlayıp sonuçların ortalamalarını aldık.

Şekil 4, QBE aracının erkenden diğer araçlara kıyasla daha çok çökme tespit ettiğini göstermektedir. KHTY aracını kullandığımız durumda ilk 10 dakika QBE çalıştırdığımız için QBE ile tamamen aynı sayıda çökme tespit edilmiştir (600 saniye). Bu 10 dakika içinde QBE 15 adet çökme tespit etmiştir. Kalan 10 dakikada ise QBE sadece 3 çökme tespit edebilirken KHTY 14 adet çökme tespit etmiştir. Sonuç olarak toplamda KHTY 29 çökme tespit ederken QBE 18 çökme tespit etmiştir. Son olarak QBE dahil bütün test yaratım araçlarının 20 dakikaya yaklaştıkça çökme bulma performanslarının azaldığını gözlemliyoruz. KHTY ise süre limitine yakınlaştığımızda bile çökme tespit etmeye devam etmiştir. KHTY QBE'den 14 adet daha fazla ve diğer araç arasında en iyi olan Sapienz aracından da 17 adet daha fazla çökme tespit etmiştir.

Şekil 5 ile KHTY yardımıyla tespit edilebilmiş ve diğer test yaratım araçlarıyla tespit edilememiş bir çökmeyi göstermekteyiz. Şekil 5a, Yahtzee uygulaması



(a) İyi Huylu Test Örneği



(b) Kötü Huylu Test Örneği

Şekil 5: Sadece KHTY ile Bulunmuş Bir Çökme Örneği

için QBE ile yaratılmış bir test örneğini göstermektedir. Bu test örneği çökme tespit etmemektedir. Şekil 5b, bu test örneğinden KHTY yardımıyla elde edilmiş bir kötü huylu test örneğini göstermektedir. Bu kötü huylu test örneği çalıştırıldığında uygulama çökerek sonlanmaktadır. Bu çökme kullandığımız diğer araçların hiçbiri tarafından tespit edilmemiştir. Bir başka çalışmada da Sapienz ve Dynodroid araçlarının bu uygulamada hiç çökme tespit etmediği raporlanmıştır [15].

6 İlgili Çalışmalar

Bu çalışmamız Test Örneği Mutasyonu (TÖM) olarak adlandırılan bir girdi mutasyonu tekniğidir. TÖM, test kümesinin kalitesini ölçmek için mutasyon işleçleri adı verilen fonksiyonlar yardımıyla program kaynak kodunu değiştiren standard Mutasyon Testinden [1] farklıdır. Çeşitli çalışmalar Android'de standard Mutasyon Testi için işleçler geliştirmiştir [18,7]. TÖM işleçleri ise program koduna değil, test kümesinin test örnekleri üzerine uygulanır ve test kümesini zenginleştirmeyi amaçlar. Önceki çalışmalarda geliştirilen standard Mutasyon Testi işleçleri bizim çalışmamıza uygulanamazlar.

Android Grafiksel Kullanıcı Arayüzü (GKA) için TÖM sınırlı da olsa Sapienz [15] ve Evodroid [14] çalışmaları ile başlamıştır. Bu çalışmalar mutasyon olarak eylemlerin gerçekleştirilme sıralarını değiştirmekte ve Yazı Değiştirme işlecini kullanmaktadırlar. Biz bu çalışmamızda beş adet yeni işleç tanımlayarak bu çalışmaları ilerletmeyi ve deneylerle TÖM yaklaşımının standard test yaratımına katkısını ölçmekteyiz.

A^{3}E [11], DynoDroid [13], PUMA [11], ve QBE [12] iyi huylu test yaratım araçlarıdır. Bu araçlar Android GKA'sının derinliklerini keşfetmeyi amaçlamaktadır. Monkey [16] ise tamamen rastgele testler üreten bir araç olduğu için kötü huylu testler yaratabilmekte ama uygulamanın derinliklerine inemediği için etkililiği kısıtlı olmaktadır. KHTY iyi huylu testleri kullandığı için uygulamanın derinliklerine de erişebilmektedir.

7 Sonuç

Bu çalışmamızda Android uygulamalarında önceki çalışmalardan daha fazla çökme tespit eden bir tam otomatik kötü huylu test yaratım yaklaşımı geliştirdik. İyi huylu Grafiksel Kullanıcı Arayüzü (GKA) testlerini kötü huylulara dönüştüren altı tane işleç geliştirdik. İşleçlerimizi var olan testlere uygulayan bir Kötü Huylu Test Yaratımı (KHTY) Algoritması geliştirdik ve bunu literatürdeki en yeni Android test yaratım aracı olan QBE üzerinde kodladık. Vaka incelemeleri ile işleçlerimizin gerçek uygulamalarda daha önce tespit edilememiş çökmelerin tespit edilmesini sağladığını gösterdik. Standard bir test yaratım aracı üzerine eklendiğinde KHTY'nin verili bir zaman bütçesinde yeni çökmeleri (crash) tespit etmekte etkili olduğunu deneylerle gösterdik.

İleride *rotasyon* ve *çift-tıklama* gibi daha geniş bir GKA eylemleri kümesi üzerinde çalışacağız. İşleçleri tamamen rastgele seçmek yerine çökme tespit etme sayılarına doğru orantılı olarak seçmenin verili zaman bütçesinde tespit edilen çökme sayısını daha da artıracağını düşünüyoruz. Son olarak da QBE ile KHTY'nin eşit süreyle çalıştırılması yerine verili zaman bütçesini daha etkili bölmenin yollarını araştıracağız.

Kaynaklar

1. Ammann, P., Offutt, J.: Introduction to Software Testing. Cambridge University Press, 1 edn. (2008)
2. Azim, T., Neamtiu, I.: Targeted and depth-first exploration for systematic testing of android apps. In: Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA) (2013)
3. Azim, T., Neamtiu, I., Marvel, L.M.: Towards self-healing smartphone software via automated patching. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE) (2014)
4. Chaffey, D.: Statistics on consumer mobile usage and adoption to inform your mobile marketing strategy mobile site design and app development (2017), http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/
5. Choi, W., Necula, G., Sen, K.: Guided gui testing of android apps with minimal restart and approximate learning. In: Proceedings of the ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA) (2013)
6. Choudhary, S.R., Gorla, A., Orso, A.: Automated test input generation for android: Are we there yet? In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering. ASE (2015)
7. Deng, L., Offutt, J., Ammann, P., Mirzaei, N.: Mutation operators for testing android apps. Inf. Softw. Technol. 81(C), 154–168 (2017)

8. eMarketer, AP: Number of mobile phone users worldwide from 2013 to 2019 (in billions) (2015), https://www.statista.com/statistics/274774/forecast-of-mobile-phone-users-worldwide/ (accessed 6/3/18, 11:51 AM)
9. Gartner: Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 1st quarter 2018 (2017), https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ (accessed 6/3/18, 11:49 AM)
10. Gultnieks, C.: F-Droid Benchmarks (2010), https://f-droid.org/
11. Hao, S., Liu, B., Nath, S., Halfond, W.G., Govindan, R.: Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps. In: Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys) (2014)
12. Koroglu, Y., Sen, A., Muslu, O., Mete, Y., Ulker, C., Tanriverdi, T., Donmez, Y.: QBE: QLearning-Based Exploration of Android Applications. In: IEEE International Conference on Software Testing, Verification and Validation (ICST) (2018)
13. Machiry, A., Tahiliani, R., Naik, M.: Dynodroid: An input generation system for android apps. In: Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE) (2013)
14. Mahmood, R., Mirzaei, N., Malek, S.: EvoDroid: Segmented Evolutionary Testing of Android Apps. In: 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). pp. 599–609 (2014)
15. Mao, K., Harman, M., Jia, Y.: Sapienz: Multi-objective automated testing for android applications. In: Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA) (2016)
16. Android ui/application exerciser monkey, http://developer.android.com/tools/help/monkey.html
17. Moran, K., Vásquez, M.L., Bernal-Cárdenas, C., Vendome, C., Poshyvanyk, D.: Automatically discovering, reporting and reproducing android application crashes. In: IEEE International Conference on Software Testing, Verification and Validation (ICST) (2016)
18. Oliveira, R.A.P., Alégroth, E., Gao, Z., Memon, A.: Definition and evaluation of mutation operators for gui-level mutation analysis. In: IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (2015)
19. Zeller, A.: Yesterday, my program worked. today, it does not. why? In: Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-7) (1999)