UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

tesi di laurea magistrale

Confronto tra strategie di generazione di test per applicazioni mobili con strumenti di Capture and Replay

2018/2019

relatore
Ch.mo Prof Porfirio Tramontana

candidato
Federica Ventriglia
Matr. M63000808

Introduzione

Sviluppo di Applicazioni Mobile ed Importanza del Testing

Per Applicazione Mobile (**App**) si intende un'applicazione software dedicata a dispositivi di tipo mobile tipicamente realizzata in modo più *leggero* in termini di risorse hardware, in linea con le restrizioni imposte dal dispositivo stesso.

La verifica e validazione di applicazioni mobile può avvenire a livelli diversi: nasce infatti la necessità di valutare scenari specifici per piattaforme mobili come compatibilità, performance e sicurezza. In ambiente Android, particolare attenzione è data al testing delle interfacce (**GUI**)

- I cosiddetti strumenti di **Capture & Replay** permettono di astrarre il processo di realizzazione di test dell'interfaccia, attraverso una **simulazione** dell'interazione dell'utente con il dispositivo.

  L'obiettivo principale di questi strumenti è quello di permettere la registrazione dell'esecuzione di un'applicazione in modo da supportare la ripetibilità automatica dei test.

  Espresso Test Recorder è lo strumento proprietario di Google per la generazione automatica di test basati su tecniche di Capture & Replay. Totalmente integrato nell'IDE Android Studio, permette di generare test basati sul framework *Espresso.*

Definizione dell'Esperimento

Lo studio si concentra sulla ricerca di una serie di parametri di configurazione del tool di Capture & Replay studiato ed il loro impatto sulla generazione di casi di test **ripetibili** e **robusti**.

Analisi della Ripetibilità: valutazione della corretta esecuzione delle test suite nello stesso ambiente di esecuzione usato per la generazione

Analisi della Robustezza: valutazione della riseguibilità delle test suite in ambienti di esecuzione diversi (Locale, Orientamento, Dispositivo)

- **Android Studio:** ambiente di sviluppo integrato (IDE) per la realizzazione di software basato sul sistema operativo Android.
- **Espresso Test Recorder:** strumento di Capture & Replay analizzato, integrato nell'IDE, basato sulla libreria *Espresso*.
- **Android Virtual Devices:** simulatori di dispositivi mobili con sistema operativo Android
- **Lint**: strumento di analisi del codice sorgente
- **Linguaggi di Scripting:** Sono stati utilizzati Python e Bash per realizzare script di supporto all'automazione dell'esperimento.

Nome	Pixel API 26
CPU	Intel Atom x86
Target	Android 8.0
SD Card	512M
Risoluzione	1080x1920:420dpo
Orientamento	Portrait

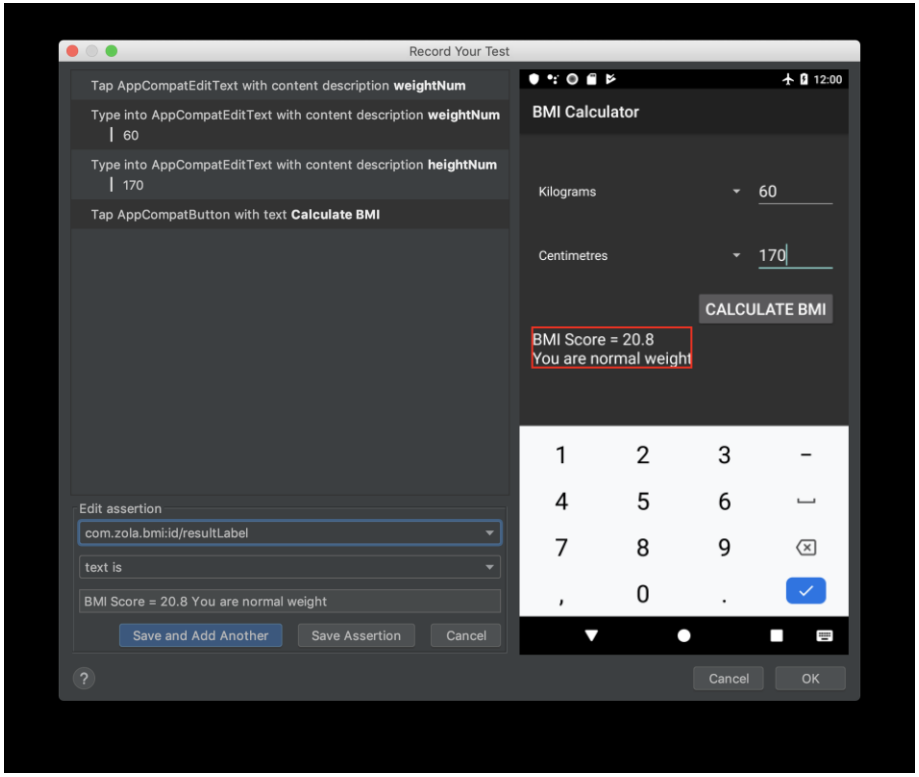
- Per l'esperimento sono state selezionate 7 applicazioni di tipo Open Source, sviluppate in Java e appartenenti a diverse categorie

Nome	Versione	LOC	Linguag-gio	Login	Offline	Permessi	Google Play	F-Droid	Tipologia
Simply Do	0.9.3	8868	Java	no	si	Lettura /Scrittura SD Card	no	si	Utilità
Primary	0.3.2	13684	Java	no	si	Lettura /Scrittura SD Card	no	si	Istruzione
Shopping List	0.11	15073	Java	no	si	Lettura /Scrittura SD Card	no	si	Shopping
Editor	1.36	5820	Java	no	si	Lettura /Scrittura SD Card	no	si	Istruzione
Counter	2.0	9715	Java	no	si	Controllo Vibrazio-ne	si	si	Utilità
BMI Cal-culator	4.0.0	9380	Java	no	si	Nessuno	no	si	Salute
Beecount	2.4.7	16012	Java	no	si	Lettura /Scrittura SD Card	si	si	Utilità

Il funzionamento di Espresso Test Recorder prevede due fasi distinte:

Fase di Registrazione: l'utente registra il caso di test replicando il caso d'uso che si vuole verificare.

Fase di Generazione: produce come output un Android test basato sul framework Espresso ed è permessa la scelta tra Java e Kotlin



I **parametri osservati** durante l'esperimento sono riconducibili alle impostazioni dello strumento di Capture & Replay scelto, in particolare:

- **Depth** Level / **Assertion** Level
- Utilizzo dell'attributo **Content Descriptor** per l'individuazione di elementi grafici

Sulla base di tali parametri sono state scelte 6 differenti configurazioni del dispositivo, tali da ottenere 6 differenti codici di test relativi allo stesso caso d'uso.

	Content Description [*YES*]	**Content Description [*NO*]**
DepthLevel[*1*]	$C_{1,y}$	$C_{1,n}$
DepthLevel[*3*]	$C_{3,y}$	$C_{3,n}$
DepthLevel[*6*]	$C_{6,y}$	$C_{6,n}$

La **procedura** seguita per l'esperimento prevede diverse fasi che includono processi di automazione realizzati per standardizzare la realizzazione dei casi di test.

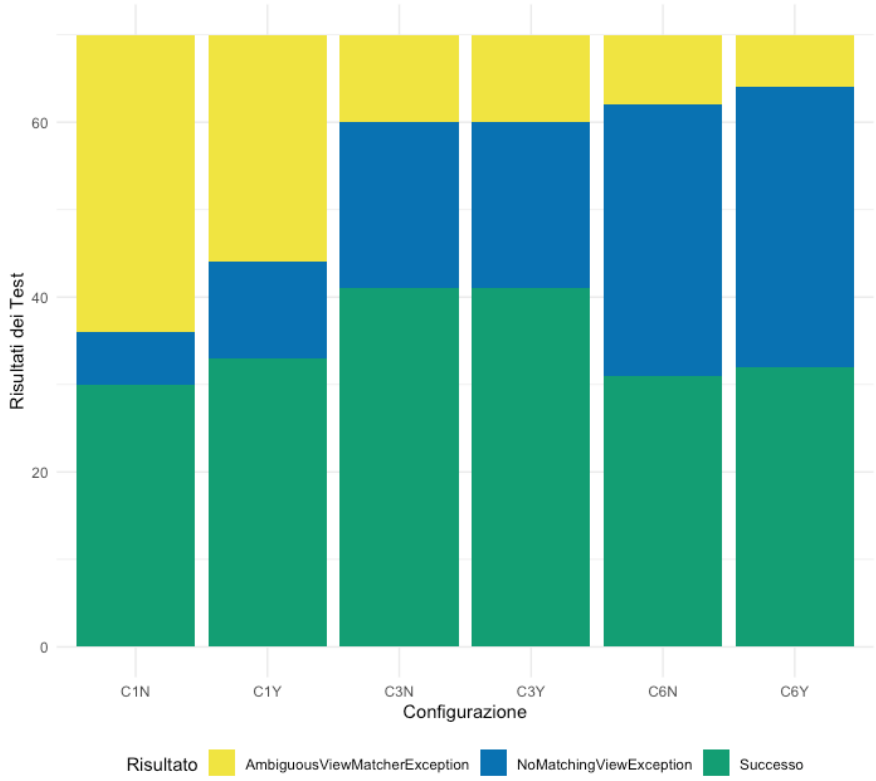
- Generazione con configurazione $C_{3,y}$ utilizzando Espresso Test Recorder
- Generazione con Configurazione C6,y utilizzando Espresso Test Recorder
- Generazione dei test nelle restanti configurazioni utilizzando uno script Python
- Esecuzione delle test suite sfruttando il **connectedCheck** di Gradle in ambiente di esecuzione standard attraverso uno script Bash
- Esecuzione delle test suite con modifiche all'ambiente di esecuzione tramite l'utilizzo di diversi script Bash

Analisi Dei Risultati: Ripetibilità

Per fallimento del Test si intende il manifestarsi si un evento di Test Breakage.

Nell'esperimento sono evidenziate due tipologie di **cause primarie di fallimento**, legate a due tipologie di Espresso Exceptions quali:

- *NoMatchingView* Exception
- *AmbiguousViewMatcher* Exception



Le **cause ultime dei fallimenti** osservati sono riconducibili a due categorie diverse. Una prima relativa al modo in cui è realizzata l'applicazione, dove si evidenziano:

- Fallimenti causati dalla mancanza di parametri identificativi sufficienti ad individuare un elemento nel layout
- Fallimenti dovuti ad una profondità del layout diversa rispetto alla profondità impostata in Espresso Test Recorder per l'applicazione

```
ViewInteraction button = onView(
    allOf(withId(android.R.id.button1),
        childAtPosition(
            childAtPosition(
                    IsInstanceOf.<View>instanceOf(android.widget.LinearLayout.class),
                     position: 0),
                position: 0),
        isDisplayed()));
button.check(matches(isDisplayed()));
```

Codice di test auto generato relativo ad un azione su un widget di tipo Button, identificato tramite solo id.

Sono stati inoltre osservati cause riconducibili a **limitazioni dello strumento** stesso, quali:

- Set di metodi della libreria Espresso limitati, non sufficienti a coprire particolari azioni e asserzioni del test
- Incapacità dello strumento di generare codice eseguibile per elementi nativi dell'interfaccia, non configurati nel codice sorgente

```
android.support.test.espresso.NoMatchingViewException: No views in hierarchy found
  matching: (with text: is "Item no. 1" and Child at position 0 in parent (with id:
  2131230845 and Child at position 0 in parent with id: 2131230846) and is displayed
  on the screen to the user)
```

Eccezione ottenuta dall'esecuzione del test, a causa di una mancata individuazione dell'elemento descritto a runtime.

Possibili **metodi risolutivi**:

- Refactor del Codice Sorgente dell'Applicazione, per aggiungere parametri mancanti a determinati Widget
- Modifiche manuali ai Test generati, da adattare al layout osservato

```
@Test
public void changeScoreC4() {
    ViewInteraction appCompatSpinner = onView(
            allOf(ViewMatchers.withId(R.id.weightSpinner),
                    withContentDescription( text: "weightSpin"),
                    childAtPosition(
                            allOf(withContentDescription( text: "relativeLayout"),
                                    childAtPosition(
                                            allOf(withId(R.id.container),
                                                    withContentDescription( text: "frameLayout")),
                                            position: 0)),
                            position: 4),
                    isDisplayed()));
    appCompatSpinner.perform(click());
```

Esempio di codice sorgente generato dallo strumento

In particolare è stata valutata la bontà dei test ottenuti rispetto ad un utilizzo estensivo degli attributi **id** e **contentDescriptor**, che permette di identificare un elemento grafico in modo univoco rispetto ad altri. Ciò che si è osservato è un possibile collegamento tra applicazioni che riscontrano una forte mancanza di tali attributi con la percentuale di test non rieseguibili.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:contentDescription="frameLayout"
    tools:context="com.zola.bmi.BMIMain"
    tools:ignore="MergeRootFrame" />
```

File XML che descrive elementi grafici dell'interfaccia

Analisi Dei Risultati: Robustezza

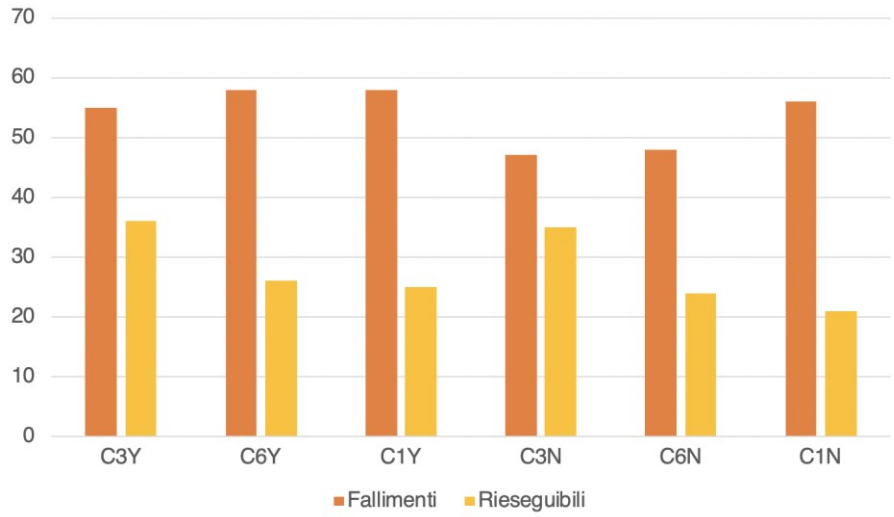
Le test suite generate sono state rieseguite in ambienti di configurazione diversi che prevedono:

- Cambiamento di **Locale** (Lingua di Sistema)
- Cambiamento di **Orientamento** (Portrait o Landscape)
- Cambiamento di **Device**

I valori ottenuti sono stati confrontati con il numero di test precedentemente *rieseguibili* in condizioni standard, per valutare l'impatto del cambio di ambiente esecutivo sul successo del test.

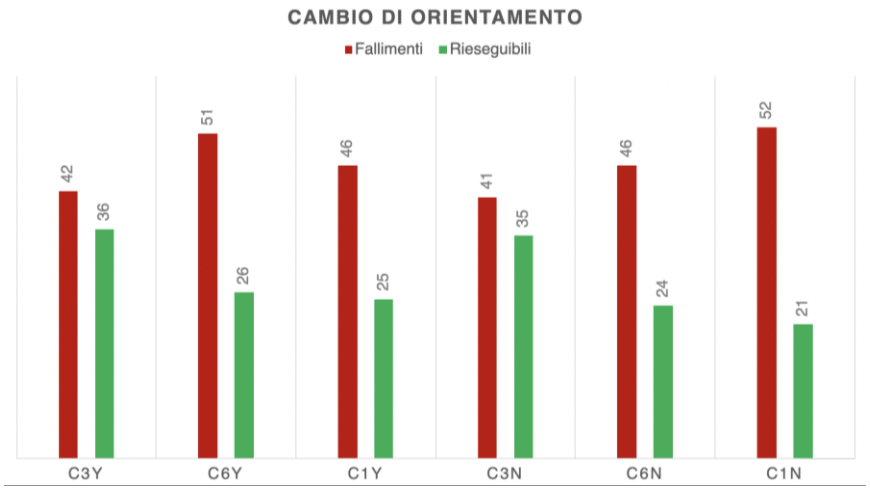
Per quanto riguarda il cambio di **Locale** si è osservato come il fallimento dei test fosse riconducibile alla mancata predisposizione di particolari applicazioni ad essere eseguite in lingue diverse.

Mancate traduzioni di valori testuali utilizzati per descrivere elementi grafici nella GUI generano un fallimento del test, in particolare per quanto riguarda i valori dell'attributo **contentDescriptor**.



Nell'analisi relativa al cambio di **Orientamento** e di **Device** di esecuzione la causa predominante di fallimento dei test è riconducibile ad una **modifica del layout** e quindi dell'interfaccia.

Un'organizzazione gerarchica diversa per il layout, combinata ad un utilizzo non estensivo di attributi identificativi per tali widget, non permette di rieseguire con successo il caso di test precedentemente generato.



Considerazioni Finali

Non è possibile definire una configurazione dello strumento che si adatti ad una generica applicazione, bensì è necessario adattare parametri come profondità rispetto al progetto in esame.

Lo stato attuale di questo strumento offre funzionalità ancora limitate se confrontate alla complessità del mobile software attualmente disponibile sul mercato, rendendo necessario un refactor del codice generato.

Nonostante ciò, esso permette di velocizzare la generazione di tali test, limitando l'impegno dello sviluppatore ad una semplice analisi del codice, partendo da una base già definita.