UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

Tesi di laurea in Ingegneria del Software

Confronto sperimentale tra tecniche di testing automatico per applicazioni Android

Anno Accademico 2011/2012

Relatore
Ch.mo prof. Porfirio Tramontana

Correlatore
Ing. Domenico Amalfitano

Candidato
Danilo Lapegna
Matr. 534/002295

Scopo

- **Comparare l'efficacia e l'efficienza di tre differenti approcci al GUI Testing automatico per Android**

  Il GUI Testing automatico è quella pratica attraverso cui si testa un software con l'utilizzo di tecniche, strumenti, tool in grado di generare e/o scatenare in maniera automatica eventi sulla relativa interfaccia utente

Metriche valutate

- **Linee di codice sorgente coperto**
- **Crash scatenati**
- **Difetti rilevati**
- **Tempo di esecuzione dei casi di test a parità di risorse hardware**

Possibili approcci al GUI Testing Automatico

- **Random testing**

  Nessuna conoscenza pregressa dell'interfaccia, che viene esercitata attraverso un insieme di eventi casuali o pseudo-casuali

- **Ripper-based**

  Nessuna conoscenza pregressa dell'interfaccia, che viene esplorata secondo precise strategie di navigazione

- **Model-based**

  Casi di test generati a partire da un modello astratto dell'interfaccia ricavato per reverse engineering

Android

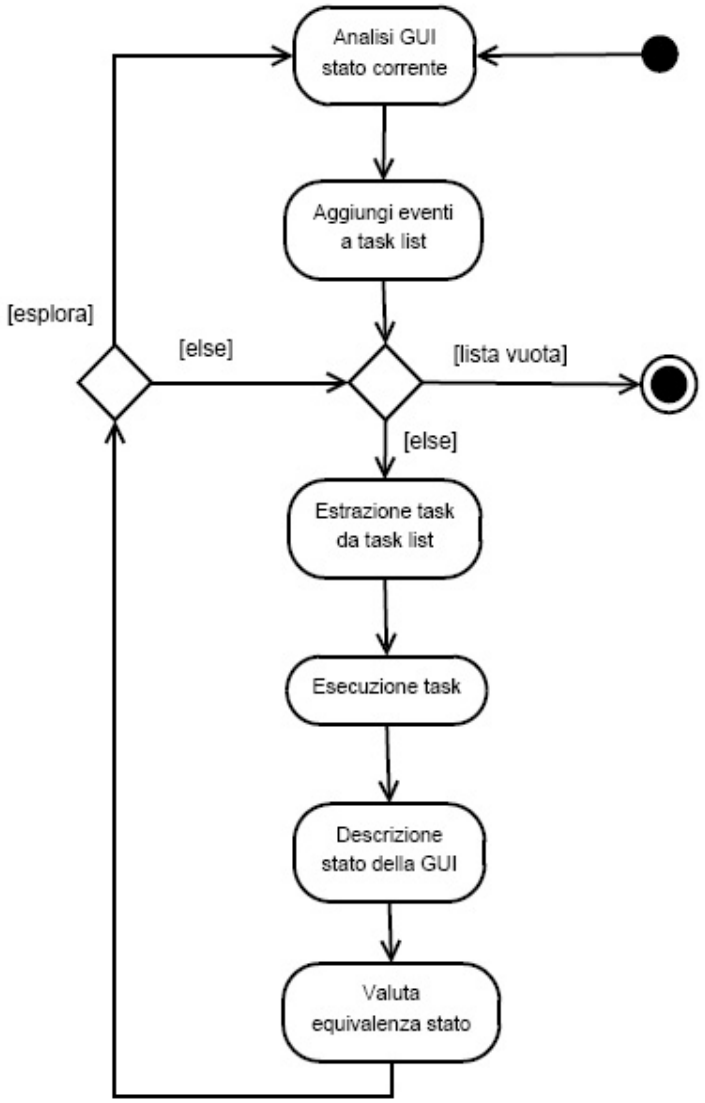
- Sistema operativo per dispositivi mobili sviluppato dalla Google Inc.
- Basato su un kernel derivato da quello di Linux
- Installato su dispositivi dotati di un insieme eterogeneo di sensori e dispositivi di comunicazione ma di ridotte risorse hardware
- Installato su oltre il 59% dei dispositivi mobili attualmente in uso
- Le applicazioni Android sono costituite da schermate dette Activity, a loro volta formate da widget (bottoni, caselle di testo, checkbox) che compongono l'interfaccia utente oggetto di testing

UI/Application Exerciser Monkey

- **Tool di GUI random testing integrato nel sistema operativo Android**
- **Permette di scatenare sull'interfaccia utente un numero definibile dall'utente di eventi pseudo-casuali**
- **Oltre al numero di eventi, consente di regolarne la frequenza in base alla tipologia (tasti di navigazione, di sistema, etc.) nonché il tempo tra due eventi consecutivi**
  - *Si è cercato di inserire frequenze eventi che potessero massimizzare la percentuale di linee di codice coperte nonché la quantità di bug e crash rilevati*

GUI Ripper

- Necessita di parametri preliminari, come il criterio di terminazione dell'esplorazione, il tipo di eventi da fornire ed il tempo tra due eventi consecutivi
- Simile ai web crawler, esplora metodicamente la GUI di un'applicazione in maniera da raccoglierne informazioni, generarvi eccezioni non gestite e dedurne un modello a stati
- Il modello generato, in formato xml, è il punto di partenza per la generazione dei casi di test GUITAR



GUITAR Test Case Generator

- Sviluppato dal Department of Computer Science dell'University of Maryland negli U.S.A
- Genera un insieme di casi di test a partire da un modello dell'interfaccia dell'applicazione
- I casi di test possono essere convertiti ed eseguiti in formato JUnit

Le applicazioni utilizzate

Sono state selezionate applicazioni:

- *OpenSource, in maniera da potere individuare le linee di codice coperte durante l'esecuzione dei casi di test*
- *Cui sia possibile segnalare i bug rilevati al relativo sviluppatore*

- **Aard Dictionary 1.4.1**
- **Tomdroid 0.5.0**
- **Andbible 1.3.0**
- **BookCatalogue 3.8.1**
- **Mileage 3.1.0**
- **Wordpress 2.0, release 394**

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

Aard Dictionary 1.4.1 - risultati

	Bug	Crash	Tempo esecuzione test	Linee di codice coperte
Monkey - 50.000 eventi scatenati - 50% eventi touch - 30% system keys - 5% motion - 5% trackball - 5% navigation - 5% major navigation	2	25	6 ore	63%
GUI Ripper - 83 tracce - Criterio di equivalenza attivato	1	1	3 ore, 20 minuti	66%
GUITAR -678 tracce	1	4	19 ore	70%

Tomdroid 0.5.0 - risultati

	Bug	Crash	Tempo esecuzione test	Linee di codice coperte
Monkey - 50.000 eventi scatenati - 50% eventi touch - 30% system keys - 5% motion - 5% trackball - 5% navigation - 5% major navigation	0	0	5 ore, 30 minuti	42%
GUI Ripper - 75 tracce - Criterio di equivalenza attivato	1	1	2 ore, 30 minuti	39%
GUITAR -711 tracce	1	13	18 ore	39%

Altri risultati

AndBible 1.3.0

	Bug	Crash	Tempo	Coverage
Monkey (50.000 eventi)	0	0	7 ore, 22min	23%
GUI Ripper (253 tracce)	1	1	26 ore, 36 min	40%

Book Catalogue 3.8.1

	Bug	Crash	Tempo	Coverage
Monkey (90.000 eventi)	2	3	33 ore	37%
GUI Ripper (548 tracce)	2	6	8 ore	46%

Mileage 3.1.0

	Bug	Crash	Tempo	Coverage
Monkey (50.000 eventi)	2	2	4 ore	53%
GUI Ripper (556 tracce)	0	0	4 ore, 12 min	54%

Wordpress 2.0 release 394

	Bug	Crash	Tempo	Coverage
Monkey (250.000 eventi)	1	5	22 ore, 12 min	28%
GUI Ripper (193 tracce)	5	9	4 ore, 42 min	37%

Conclusioni

- L'approccio con GUI Ripper è generalmente dotato di una buona efficacia ed una buona efficienza
- L'approccio model-based di GUITAR è tendenzialmente il più efficace ma temporalmente molto dispendioso
- L'approccio random con Monkey spesso pecca sia in efficacia che in efficienza, ma può essere utile per trovare difetti non individuabili con gli altri metodi

Sviluppi futuri

- Un ripper dotato della possibilità di simulare sensori esterni nonché la ricezione di chiamate e messaggi SMS.
- Un ripper in grado di effettuare random testing