UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

elaborato di laurea

Strumenti per il Testing di Applicazioni Android

Anno Accademico 2013/2014

relatore
Ch.mo prof. Porfirio Tramontana

candidato
Gaetano Valenti
Matr. 534/2174

Lorem ipsum dolor sit amet

Sommario

Capitolo 1 Analizza l'ambiente messo a disposizione degli sviluppatori dal *Framework Android*. Comincia con la struttura del sistema operativo, per poi approfondire i componenti specifici offerti dal Sistema Operativo (SO).

Capitolo 2 Esamina il testing Android, per elencare le principali criticità a cui vanno incontro gli sviluppatori e per permettere, nei successivi capitoli, di analizzare possibili soluzioni a detti problemi.

Capitolo 3 Esamina una serie di strumenti che consentono la scrittura di test per Mobile Application (App) *Android*, analizzandone pregi, difetti e potenzialità. Mostra le principali funzionalità e presenta esempi di Test per chiarirne meglio il funzionamento.

Capitolo 4 In questo capitolo vengono messe a confronto le principali peculiarità di tutti gli strumenti analizzati.

Indice

Elenco delle figure

Acronimi e Definizioni

AGPL v3 GNU Affero General Public License v3

API Application Programming Interface

App Mobile Application

BDD Behavior-Driven Development

CI Continuous Integration

DDD Domain-Driven Design

DFE Depth-First Exploration

DVM Dalvik Virtual Machine

GUI Graphical User Interface

I/O Input/Output

OOAD Object-Oriented Analysis and Design

PC Personal Computer

PDA Personal Digital Assistant

QoS Quality of Service

SATG Static Activity Transition Graph

SDK Software Development Kit

SO Sistema Operativo

TE Targeted Exploration

TDD Test-Driven Development

UI User Interface

VNC Virtual Network Computing

Introduzione

Sono passati più di 25 anni da quando, nel 1986, venne rilasciato sul mercato il primo Personal Digital Assistant (PDA) della storia: il *Psion Organiser II* [26], con le prime *mobile application* della storia, il *calendario*, la *rubrica*, il *block notes*, etc.
Questo diede il via ad un nuovo mercato, che ha visto la nascita di innumerevoli *PDA* . Tra questi, ricordiamo il *Nokia 9000 Communicator*, il primo telefono cellulare con funzionalità *PDA*.
Nel corso degli anni sono comparsi anche svariati *Sistemi Operativi* per *PDA*, tra cui i più diffusi sono [26] :

- Android: prodotto dalla *Open Handset Alliance* (gruppo composto da 84 società, che ha Google come capofila) e immesso nel 2008 sullo smartphone HTC Dream, è il sistema operativo per palmari e smartphone più diffuso.
- Palm OS: dalla sua introduzione nel 1996, la piattaforma PalmOS ha definito il trend e le aspettative per i palmari. Grazie a questa piattaforma, i palmari da semplici agende elettroniche si sono evoluti in veri e propri computer.
- Windows Mobile (Windows CE) di Microsoft per i suoi dispositivi Pocket PC.
- BlackBerry OS di Research In Motion. è un tipo di palmare wireless introdotto nel 1999. Permette di scaricare dalla propria casella di posta elettronica, di gestire la propria agenda, di scrivere messaggi SMS e permette effettuare chiamate verso i propri contatti.
- GNU/Linux, utilizzato da Sharp per i suoi Zaurus. Zaurus è un palmare prodotto dalla Sharp che utilizza un'implementazione del sistema operativo Debian/Linux. Inoltre esiste un'implementazione per questo palmare della distribuzione Gentoo.
- Symbian (prima noto come EPOC). è un sistema operativo aperto, adottato da varie aziende come standard.

- iOS: sistema operativo di Apple per i dispositivi palmari

Una *mobile application* è un software "autosufficiente" sviluppato per l'esecuzione di specifici task su dispositivi mobili (smartphone, lettori mp3, tablet, etc. etc.) [7].
La costante crescita del mercato delle applicazioni mobili, avuta negli ultimi anni, rendono questo mercato un ottima opportunità di business per sviluppatori e software house.
Quanto detto, fà comprendere la crescente necessita di strumenti per lo sviluppo ed il testing di mobile applications, che rendano il lavoro degli sviluppatori il più semplice e rapido possibile.
Il test di applicazioni Android si propone di eseguire l'applicazione utilizzando combinazioni di input e di stati per rivelare guasti[7]. Con il termine *guasto*, si intende l'incapacità manifesta di un sistema o componente, di eseguire la funzione richiesta entro specifici requisiti di prestazioni [1]. Le fonti di guasti comprendono:

- Anomalie di implementazione delle applicazioni.
- L'ambiente di esecuzione.
- Interfaccia tra l'applicazione ed ambiente.

Questa classificazione è frutto di un recente studio che ha analizzato 630 bug trovati in App realmente presenti all'interno dell'*Android Market*[2].
Un ulteriore studio, eseguito su 10 App gratuite presenti nel dell'*Android Market*, ha classificato 7 tipologie di errori:

- Activities
- Eventi
- Errori di tipo dinamici
- Input/Output (I/O)
- Errori di concorrenza

- Eccezioni non catturate
- Altro

Questo studio, inoltre, mostra che durante l'*esecuzione* di un errore, la natura delle applicazioni *Android* propagano gli errori ad altre classi.
I processi di *Test* devono, per quanto possibile, cercare di combinare differenti strategie di *Test* e prendere in considerazione diversi livelli di analisi, al fine di rivelare i diversi tipi di guasti osservabili nelle App. Le strategie possibili sono:

- Black box

  Progettazione di casi di *Test* in base alla funzionalità specifica dell'elemento da testare.

- White box

  Progettazione di casi di *Test* in base al codice sviluppato.

- Gray box

  Progettazione di casi di *Test* che sfrutta sia, la progettazione *Black Box*, che quella *White Box*.

Sulla base di una indagine eseguita di recente [8], l'approccio al *Mobile Testing* può essere classificato in diversi modi. Un modo è quello di classificarli in base alle *tecniche di collaudo*. Le tecniche di collaudo, ad esempio, possono includere *Test random-based* e *Test model-based*. L'altro metodo È quello di classificarli in base alla sottostante infrastruttura client-server. Approcci popolari in questa categoria sono: (In figura 1 è possibile vederne degli esempi).

- *Emulation-based mobile testing*: Con quest approccio, le App vengono validate con l'uso di *emulator* di *Mobile devices*. Questo è uno deli approcci più economic, non richiedendo la necessityà di aver *Devices* reali.

Figura 1: Esempi di approccio al testing a seconda dell'infrastruttura client-server

- *Simulation-based mobile testing*: Questo sistema verifica le App simulatori di *Devices*. È simile al precedente approccio, ovvero, non sono necessari dispositivi reali. A differenza della precedente , questa soluzione ha il vantaggio di poter *simulare* differenti dispositivi. Tuttavia, ha come difetto, l'impossibilità di poter testare l'interazione con gli strumenti di sistema. Un tipico esempio può essere quello di avere la necessità di integrare nell'App i pagamenti *PayPal*.

- *Device-based mobile testing*: Questa tipologia di approccio richiede la necessità di acquistare un dispositivo reale da usare durante i test. Con questo sistema, è però possibile testare l'interazione con gli strumenti del *Device*, comportamenti dinamici e i Quality of Service (QoS). Questo approccio è tanto più costoso, quanto più è

performante il *Device* acquistato.

- *Remote device-based mobile testing*: In questa modalità, i *Devices* vengono affittati da remoto. Trattandosi, in genere, di servizi *pay-as-you-use*, questo sistema risulta sicuramente più conveniente dei precedenti, dando modo di testare le App su larga scala per le prestazioni e la scalabilitĹ del sistema.

Nei seguenti capitoli verranno analizzati vari strumenti per la progettazione e l'esecuzione di casi di *Test* su App *Android*, mettendo in evidenza, per ognuno di essi, pregi e difetti, in mo utilizzato.

Capitolo 1 Overview di Android

1.1 Struttura del SO

Android, una delle piattaforme open source più popolari al mondo, è sviluppata dalla Open Handset Alliance.

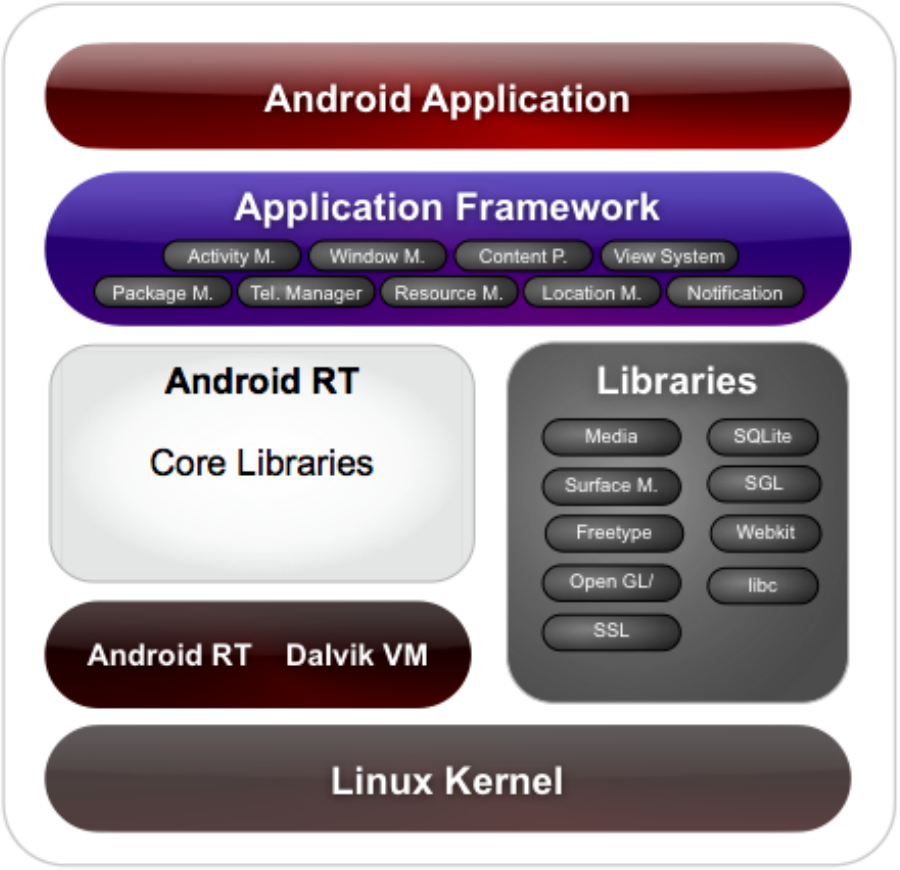

Figura 1.1: Android application stack [14]

Android è costituito da quattro strati software:

1. Linux kernel

   Il *Linux Kernel* astrae l'hardware del dispositivo e contiene il nucleo di servizi come:

   - Gestore dei processi
   - Gestore della memoria di sistema

- Gestore del filesystem
- etc. etc.

2. Static library

   Lo *Static Library Layer* fornisce un set standard di librerie tra cui è possibile vedere l'*Android Runtime*. L'*Application framework layer* espone le funzionalità offerte dalle librerie agli sviluppatori. L'Android Runtime contiene al suo interno il nucleo delle *Runtimes libraries*, necessarie al sistema operativo ed alla Dalvik Virtual Machine (DVM) (virtual machine sviluppata ed ottimizzata appositamente per Android).

   L'applicazione Android gira in un proprio processo, con un particolare user ID in una propria istanza della DVM. Questo sistema fornisce un livello base di sicurezza.

3. Application framework

   L'*Application Framework Layer* fornisce un set di componenti riusabbili per lo sviluppo di nuove applicazioni, fornisce inoltre servizi per le applicazioni, come:

   - Gestore risorse
   - Gestore notifiche
   - Gestore delle Activity

4. Applications

   Lo strato superiore dello stack *Android application* contiene applicazioni chiave come:

   - Client mail
   - Calendario
   - Gestore contatti
   - etc. etc.

L'*Application framework* mette a disposizione degli sviluppatori quattro componenti Java:

- Activity

- Service
- Broadcast Receiver
- Content provider

1.2 Struttura delle applicazioni

Le quattro classi messe a disposizione dall'Application Framework hanno funzioni fondamentali per lo sviluppatore:

1.2.1 Activity

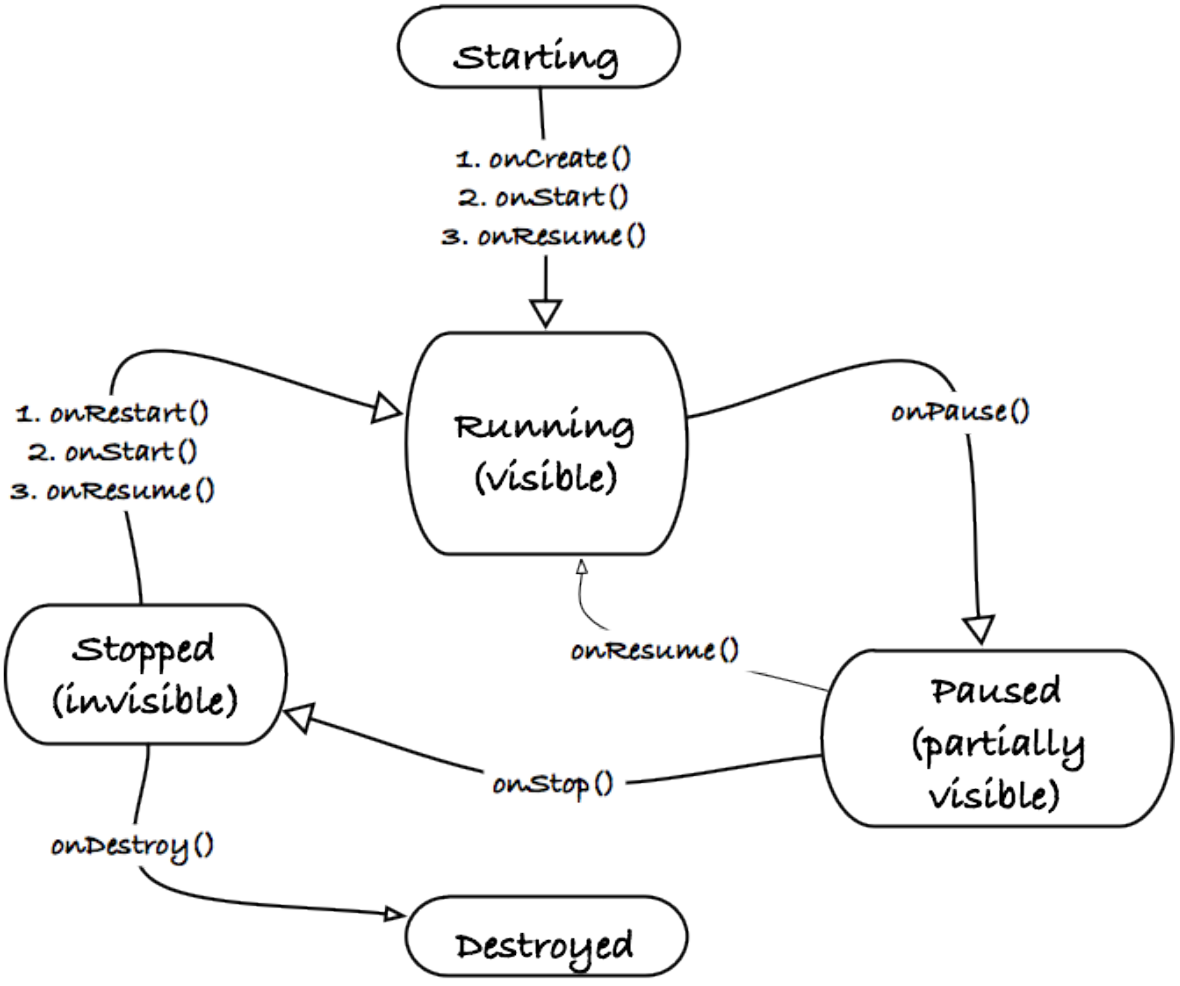

Figura 1.2: Ciclo di vita dell'Activity[10]

L'*Activity* è uno degli elementi centrali di ogni applicazione Android. Normalmente una Activity rappresenta una singola schermata dell'applicazione. Le applicazioni possono definire una o più Activity per trattare diverse fasi del software. Ogni Activity è responsabile del salvataggio del proprio stato in modo da poterlo ristabilire successivamente come parte del ciclo di vita dell'applicazione.

Generalmente una Activity corrisponde ad un'azione specifica che l'utente può fare. Essendo una delle componenti principali nel ciclo di vita di ogni applicazione Android, il modo in cui le Activity sono lanciate e in cui interagiscono tra loro è una parte fondamentale dello sviluppo. Ci può essere solo una Activity attiva per volta (foreground), nello stesso tempo: una Activity che in un determinato momento non si trova nello stato attivo e quindi non si trova in foreground (backgroud) potrebbe essere terminata dal sistema operativo, ad esempio, per memoria insufficiente. Questo significa che ogni applicazione Android può cambiare lo stato in ogni momento, devono essere presi, quindi, i dovuti accorgimenti per la gestione degli stati di un Activity.

1.2.2 Service

Un *Service* è un processo che gira in background senza la diretta interazione con l'utente (similmente ai *deamon* in ambiente Unix). La classe Service viene utilizzata per creare componenti software che possono svolgere attività senza interfaccia utente. Utilizzando i Service è possibile far girare applicazioni e farle reagire ad eventi anche quando queste non sono in primo piano: un Service può essere avviato, fermato e controllato da altri componenti dell'applicazione, inclusi altri Service e Activity.

Un Service avviato, ha una priorità più alta rispetto ad una Activity in stato di inattività, in questo modo vi è minore probabilità, per un Service, di essere terminato dal gestore delle risorse di runtime. L'unica ragione per cui Android potrebbe fermare un Service prematuramente è per fornire risorse addizionali al componente software in primo piano (normalmente una Activity). Quando questo si verifica il Service fermato sarà riavviato automaticamente non appena le risorse diventano nuovamente disponibili. Nel caso in cui il Service debba

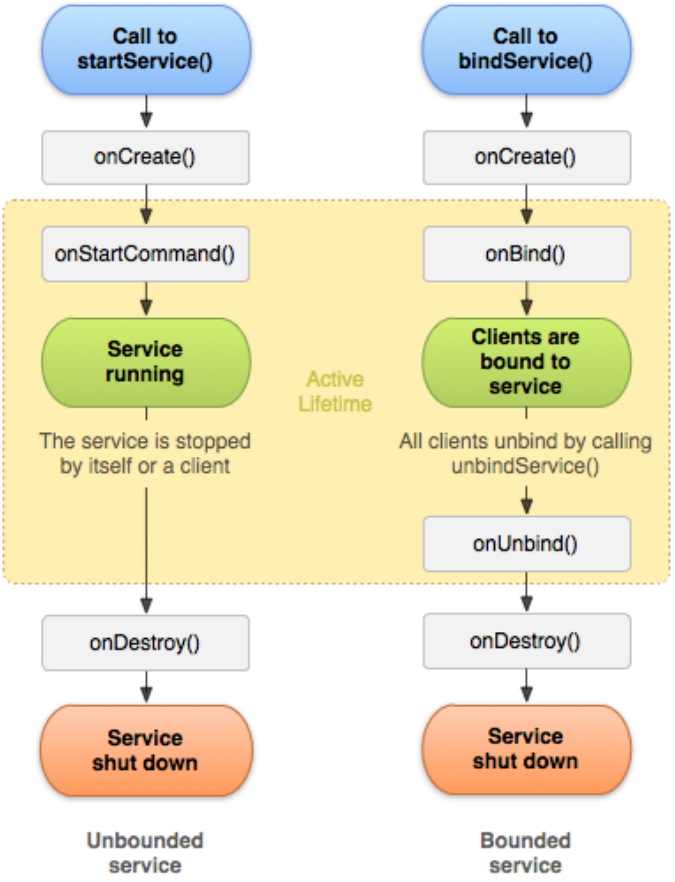

Figura 1.3: Ciclo di vita del Service[12]

interagire direttamente con l'utente, potrebbe essere necessario incrementare la sua priorità per portarla al pari dell'Activity in foreground. Questo assicura che i Service non vengano terminati se non in circostanze estreme. Tutto ciò, chiaramente, a scapito della gestione delle risorse a tempo di esecuzione.

I Service hanno un ciclo di vita abbastanza semplice e possono essere classificati in:

- Unbounded

  Non interagisce, nè restituisce, alcun output al chiamante. Viene creato con il metodo *StartService*. Al termine dell'esecuzione dei propri *task* si ferma, oppure può essere fermato dal componente che lo ha creato, a questo punto interviene il SO che distrugge il Service e libera le risorse da esso occupate.

- Bounded

Sfrutta la comunicazione tra processi per interagire e/o restituire output al chiamante. Viene creato (o *agganciato*) con il metodo *BindService*. Per terminare il Service, bisogna che tutti i componenti che lo abbiano agganciato, lo *sgancino* con il metodo *onUnbind*. Solo quando non ci saranno più componenti ad interagire con il esso, interviene il SO che lo distrugge e libera le risorse occupate.

1.2.3 Broadcast receiver

Un *Broadcast receiver* permette di registrare eventi di sistema o applicativi. Tutti i ricevitori registrati per un evento saranno avvisati per Android una volta che questo evento si verifica. In pratica, il Broadcast receiver è l'implementazione dell'*Observer pattern* in Android.

1.2.4 Content provider

Un *Content provider* è un contenitore di dati, questo rappresenta uno dei modi migliori per condividere informazioni "globali" tra applicazioni. Nel modello di sicurezza implementato in Android, i file scritti da una applicazione non possono essere letti o scritti da altre applicazioni. Android è un sistema Linux-based in cui ogni applicativo ha il suo userid Linux, la sua directory "data" (//data/data/nome_pacchetto) e il suo spazio di memoria protetto. Questo è il motivo per cui gli applicativi Android necessitano dei Content provider per comunicare tra loro. I processi possono segnalare se stessi al sistema come Content provider di un insieme particolare di dati: quando queste informazioni sono richieste vengono richiamate da Android grazie ad un insieme specifico di API che permettono di agire sul contenuto secondo le specifiche definite.

1.2.5 File Manifest.xml

L'*AndroidManifest.xml* è il file che definisce i contenuti e il comportamento di una applicazione: al suo interno ne sono elencate le Activity e i Service, con i permessi necessari per il corretto funzionamento.

Ogni applicazione gira all'interno di un proprio processo Linux, per cui ci sono delle restri-

zioni ben specifiche: ogni processo non può accedere alla memoria di un altro processo, e ad ogni applicazione è assegnato uno specifico identificatore utente. Inoltre i file di un applicativo non possono essere letti o scritti da altri applicativi: anche l'accesso a diverse operazioni critiche sono protette, per tutte queste azioni bisognerà specificatamente richiedere i permessi nell'AndroidManifest.xml.
Tra i principali permessi disponibili ci sono:

- READ_CONTACTS: leggere (ma non scrivere) i dati dei contatti dell'utente.
- WRITE_CONTACTS scrivere (ma non leggere) i dati dei contatti dell'utente
- RECEIVE_SMS: monitorare l'arrivo di messaggi SMS.
- INTERNET: accedere ed utilizzare la connessione Internet.
- ACCESS_FINE_LOCATION: utilizzare un accurato sistema di localizzazione come il GPS.

1.2.6 Intent

Un *Intent* è un meccanismo che descrive un'azione specifica, come, ad esempio: "chiamare casa" o "inviare un sms". In Android quasi tutto passa attraverso un Intent e lo sviluppatore li può utilizzare per riusare o sostituire diversi componenti software. Ad esempio: è disponibile un Intent "inviare una email": se si ha la necessità di gestire l'invio di email è possibile invocare l'Intent messo a disposizione dal sistema, oppure, scriverne uno nuovo e utilizzare quest'ultimo sostituendo l'Intent di default.
Schematizzando, un Intent può essere utilizzato per:

- Trasmettere l'informazione per cui un particolare evento (o azione) si è verificato;
- Esplicitare un'intenzioni che Activity o Service possono sfruttare per eseguire un determinato compito o azione, solitamente con o su particolari insiemi di dati;
- Lanciare particolari Activity o Service;

- Supportare l'interazione tra applicazioni installate sul dispositivo, senza la necessità di dover conoscere a priori il tipo di applicazione.

L'Intent è uno dei concetti e dei meccanismi fondamentali di Android: la sua implementazione trasforma ogni dispositivo Android in una collezione di componenti indipendenti ma facenti parte di un singolo sistema interconnesso.

1.3 Classificazione delle applicazioni

Sui *Devices Android* è possibile eseguire più tipologie di App:

- *Native:* Per la loro creazione viene sfruttato l'ambiente nativo di sviluppo, cioŕ l'SDK di Android, Java e gli xml. Questo permette l'esecuzione diretta dell'App sul *Device*.

- *Ibride:* Queste App, anche se eseguite direttamente sul device, vengono sviluppate sfruttando linguaggi generalmente orientati al web development, quali HTML5, javascript e css. L'applicazione viene poi eseguita all'interno di un componente nativo che sfrutta il *browser engine* del *Device*. Questo meccanismo sacrifica leggermente le *performaces*, ma al contempo sono facilmente portabili su altri SO.

- *WebApp:* Sono comuni applicazioni server-side che producono HTML ottimizzato per dispositivi mobili e che quindi sfruttano il browser messo a disposizione dal *Device* per la loro visualizzazione. Generalmente sono più lente, non possono funzionare senza una connessione al Server, ma è possibile scriverle con qualsiasi linguaggio già utilizzato per questo tipo di software.

Capitolo 2 Overview Application Testing

2.1 Mock

I *mock* sono degli *oggetti* creati per *imitare* componenti reali del software. Gli scopi principali del loro utilizzo sono:

- Consentire lo sviluppo di componenti software anche in assenza di componenti necessari al loro funzionamento (spesso sviluppati parallelamente).
- Permettere di isolare i componenti sotto test dal resto del software e quindi, di eseguire i test in modo indipendente e ripetibile[5].

Pertanto, il comportamento di un oggetto complesso può essere simulato da un mock, consentendo al programmatore di scoprire se l'oggetto sotto test risponde correttamente al variare degli stati del mock[22].
Se un oggetto (reale) ha le seguenti caratteristiche è indicato usare dei mock per il testing di tali componenti:

- Fornisce risultati non deterministici (es. misurazione della temperatura).
- Ha stati difficili da creare o riprodurre.
- È lento.
- Non è stato ancora sviluppato.

I mock devono inoltre condividere le interfacce dei componenti reali, in modo che i componenti che li utilizzano non debbano essere adattati a seconda di quali oggetti devono usare.

2.2 Test Unitari

Nella programmazione, un *Test unitario* è un sistema con cui singole unità di codice, uno o più moduli congiuntamente con i propri dati, procedure di utilizzo e procedure operative, sono sottoposti a test per determinare se sono idonei per l'uso[3].

Intuitivamente, si può immaginare un'unità come la più piccola parte verificabile di un'applicazione. Nella programmazione procedurale un'unità può essere un intero modulo, ma è più comunemente una singola funzione o una procedura. Nella programmazione orientata agli oggetti un'unità è spesso un intera classe, ma potrebbe essere anche un singolo metodo[4].

Gli *Unit test* sono test del software "scritti da programmatori, per programmatori, in un linguaggio di programmazione"[5]. Questi dovrebbero isolare il componente da testare e rendere possibile la ripetizione del test innumerevoli volte. Motivo per cui, in genere, unit test e mock vengono sviluppati congiuntamente e disposti negli stessi package.

Si immagini di dover sviluppare uno unit test che debba fare una prova di cancellazione. Se il test eliminasse realmente i dati presenti nella base dati o nel filesystem, ci sarebbero problemi nella ripetizione del test, questo rende necessario l'uso di un mock per evitare la modifica permanentemente dei dati presenti sul dispositivo.

Focalizzando i test unitari, nell'ambito della programmazione di App Android, bisogna, in prima istanza, rispondere a tre domande:

1. Cosa bisogna testare in una App Android?
2. Qual è la strategia di test con maggior efficacia?
3. Come si possono automatizzate gli unit test?

Come evidenziato nel capitolo 1, in una App Android coesistono, sia componenti nativi del SO Android, sia *comuni* classi Java. I test non devono soffermarsi solo su uno dei tipi appena descritti, ma devono ricercare bug in tutto il codice sviluppato. Da ciò la necessità di Framework di test che permettano di testare tutti i tipi di componenti, siano essi *nativi* Android o *semplici* classi Java.

Per quanto riguarda la strategia da adottare è possibile utilizzare tutte e tre le strategie di test (Bottom-Up, Top-Down, Sandwitch), resta fondamentale però prestare le dovute attenzioni agli *eventi* ed ai cicli di vita dei componenti nativi del Framework Android.

Infine, la possibilità di automatizzare i test dipende dal Framework di test utilizzato, infatti, a seconda della scelta si possono automatizzare i test di singoli componenti, così come, creare delle vere e proprie *Suite di test* per l'esecuzione automatica di innumerevoli test su svariati componenti.

2.3 Test di integrazione

I *test di integrazione* sono disegnati per testare il comportamento di componenti che lavorano congiuntamente tra loro[5]. Si eseguono quindi dei test integrando moduli che hanno passato i Test unitari.

In generale, per testare una Activity è necessario integrarla con svariati componenti del sistema, come il Manager dell'Activity, che deve regolare il corretto funzionamento del ciclo di vita, l'accesso alle risorse ed ai dati. Quindi, bisogna che l'ambiente di test fornisca tutti gli strumenti per permettere ciò.

Analogo ragionamento và fatto per altri componenti quali Service, Content provider e Broadcast Receiver.

In tutti questi casi il Framework di test adottato deve cercare, per quanto possibile, di semplificare i processi di scrittura di questi test, che risulta, in genere, lungo e tedioso, ma assolutamente necessario per lo sviluppo di codice di qualità.

Le strategie adottate in questi test, in genere, sono tre:

1. Bottom-Up

   Testa partendo dai componenti a basso livello, per arrivare poi ai componenti di alto livello.

2. Top-Down

   Parte dai componenti di alto livello per arrivare ai componenti di basso livello.

3. Sandwitch

   Si cerca la giusta via di mezzo tra i due sistemi sopra citati.

Si definisce un *componente di basso livello*, un componente software che non ha dipendenze da altri componenti software all'interno di una applicazione. Al contrario, un *componente di alto livello* è un componente che ha dipendenze da altri componenti software all'interno dell'applicazione stessa[7].
Ulteriori classificazioni possono essere fatte nella relazione tra componenti software, una delle più importanti per i test di integrazione è:

1. statica
2. dinamica

La prima relazione è la più semplici da individuare, infatti, detto tipo di relazioni sono dichiarate all'interno del file Manifest.xml . Quelle dinamiche, invece, dipendono dall'accoppiamento tra classi java, conseguentemente, serve una conoscenza approfondita del codice prodotto per poter comprendere a pieno detto accoppiamento.
In genere si sviluppano due tipi di App:

1. Server - Client
2. Client

Nelle App Server - Client, generalmente, i primi test vengono effettuati sull'applicazione lato client. Solo in un secondo momento si passa ai test lato server. Facendo in questo modo si riesce ad evitare che bug presenti lato Server, si ripercuotano sul lato Client del progetto. Questo problema, non è presente invece nel caso di App Client site, infatti, non dipendendo queste da software Server site, il tester è libero di testare l'App senza doversi preoccupare, che, ulteriori componenti software vadano ad influenzare i test sviluppati.
A valle delle classificazioni appena esposte analizziamo i vari componenti Android a che *livello* si trovano (alto o basso livello):

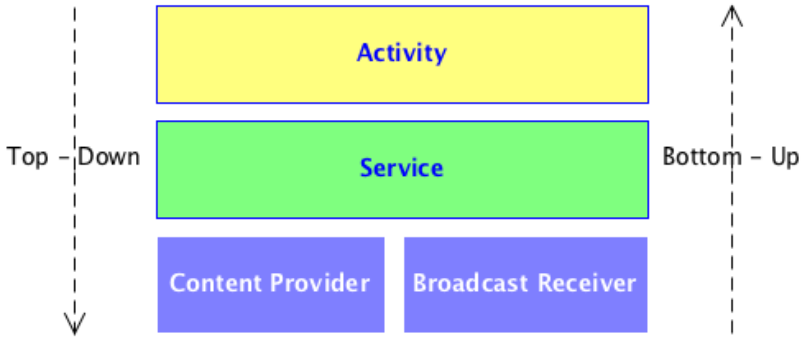

Figura 2.1: Livello componenti Android

- Content Provider

  Come già esposto in precedenza, questo componente ha il compito di fornire dati a chi ne fà richiesta, quindi, non dovrebbe dipendere da altri componenti che non siano le Application Programming Interface (API) di accesso ai dati fisici di Android. Quindi, questo componente, nella maggior parte dei casi, è da considerarsi di basso livello.

- Broadcast receiver

  Anche questo componente, come il Content provider dovrebbe dipendere dalle sole API del SO, dato che il suo compito principale è quello di raccogliere i messaggi di sistema (Intent) per poi notificarlo ad altri componenti all'interno dell'App, per cui, anche in questo caso, nella maggio parte dei casi, questo è un componente di basso livello.

- Service

  Con questo componente non si può più parlare di *basso livello*, dato che, in genere, i Service hanno bisogno, per il loro funzionamento, di interfacciarsi con altri componenti, come Content Provider e Broadcast receiver.

- Activity

Questo, generalmente è il componente di *alto livello* dell'App, infatti, è consuetudine che l'Activity dipenda da altri componenti, come Service, Content Provider e Broadcast receiver.

2.4 UI test

È molto importante testare il corretto funzionamento della UI, questo infatti, permetterà di trovare errori che potrebbero risultare *bloccanti* per l'utilizzo di una App. Bisogna prestare particolare attenzione per i test che controllano il corretto funzionamento dei componenti dell'interfaccia utente. Infatti, l'interfaccia utente di una App, può essere modificata solo dal thread principale.

2.5 Test di sistema

IEEE definisce il test di sistema come "il test condotto su un sistema completo e integrato per valutare la conformità del sistema con le *specifiche funzionali*"[1]. Il test del sistema verifica la conformità dei requisiti funzionali e non funzionali ed ha lo scopo di scoprire errori, a questo livello di test, probabilmente causati dal dispositivo o dall'ambiente di esecuzione reale dell'App.

La grande varietà di dispositivi mobili presenti sul mercato e di conseguenza, la quantità di hardware con cui una App può dover interagire è praticamente illimitata. Questo rende la fase dei test di sistema un processo molto complesso, contando che è praticamente impossibile testare l'App con tutti i dispositivi presenti in commercio. In generale, i tester devono eseguire le applicazioni su una serie di dispositivi diversi, ognuno con varie limitazioni di risorse. Un test di sistema, procede, generalmente, per stadi successivi, in modo, da meglio controllare la risposta dell'App alle caratteristiche dei dispositivi.

I Test di sistema procedono di solito in tre fasi:

1. Simulazione

   In questa fase il test è eseguito completamente in ambiente simulato.

2. Prototipazione

   In questa fase, le componenti reali vanno a sostituire gradualmente i componenti emulati, ma il processore continua ad essere simulato.

3. Pre-produzione

   Infine, viene sostituito anche il processore emulato, con un processore reale.

Diversi livelli di sforzo, costi e tempi caratterizzano ogni fase. Per esempio, la fase di simulazione è facile e poco costosa, mentre i test in fase di pre-produzione possono risultare molto costosi.

2.6 Continuous Integration

La Continuous Integration (CI) è una pratica di sviluppo software in cui i membri di un team integrano frequentemente il loro lavoro (generalmente usando sistemi di *versioning*). Ogni integrazione è verificata da un *build*, che riscontra se vi siano problemi di compilazione. Spesso vengono usati strumenti quali *jenkins* o *hudson*, che permettono la creazione di un insieme di step da eseguire. Tra le operazioni eseguite in fase di integrazione c'è anche l'esecuzione di test che evidenzino errori di integrazione o di regression.

Capitolo 3 Strumenti di Testing

3.1 JUnit

3.1.1 Overview

L'ambiente di sviluppo di Android contiene, al suo interno, un *Framework*, basato su JUnit, che fornisce l'achitettura e gli strumenti per un test completo delle App sviluppate [21].

I punti di forza di questo ambiente sono:

- È basato su JUnit, questo consente di usarlo sia per il test di classi "puramente java", ovvero, che non sfruttano componenti nativi del SO Android, sia per classi che sfruttano il *Framework Android*.
- Le estensioni JUnit di Android forniscono specifiche classi di test case. Queste classi forniscono metodi di supporto per la creazione di mock e metodi che consentono il controllo del ciclo di vita dei componenti.
- Le suite di test sono contenute in packages simili a quelli delle applicazioni principali, questo consente la progettazione e la creazione di test senza eccessivo dispendio di tempo per lo studio di ulteriori tecniche.
- Gli strumenti SDK per lo sviluppo ed i test sono disponibili in Eclipse grazie al plug-in ADT, in alternativa, è possibile sfruttare gli strumenti messi a disposizione dalla riga di comando. Detti strumenti, analizzano ed estrapolano informazioni dal progetto da sottoporre a test e le utilizzano per creare automaticamente file di compilazione, il file Manifest, e la struttura delle directory per il pacchetto di prova.
- L'SDK fornisce *monkeyrunner*, strumento che mette a disposizione le API per la creazione di programmi python, capaci di:
  - Installare App
  - Testare App

  - Eseguire App
  - Inviare sequenze di tasti
  - Creare screenshots dell'interfaccia utente

Il presente capitolo tratterà:

- 3.1.2 Creazione di un progetto di Test
- 3.1.3 Test di componenti specifici di Android
- 3.1.3 Mock
- 3.1.3 Testing delle Activity
- 3.1.3 Testing dei Service
- 3.1.3 Testing dei Content Provider
- 3.1.3 Testing dei Broadcast Receiver
- 3.1.3 Test delle UI
- 3.1.4 Test delle classi java

3.1.2 Creazione di un progetto di Test

La creazione di un progetto di Test JUnit per Android è molto simile alla creazione di un comune progetto Android [6]:

- Da riga di comando

  Esecuzione comando:

  ```
  android create test-project -m <main_path>
                              -n <project_name> -p <test_path>
  ```

  dove:

- main_path: è il folder contenente il progetto da testare.
- project_name: è il nome del progetto di test.
- test_path: è il folder che conterrà il progetto di test.

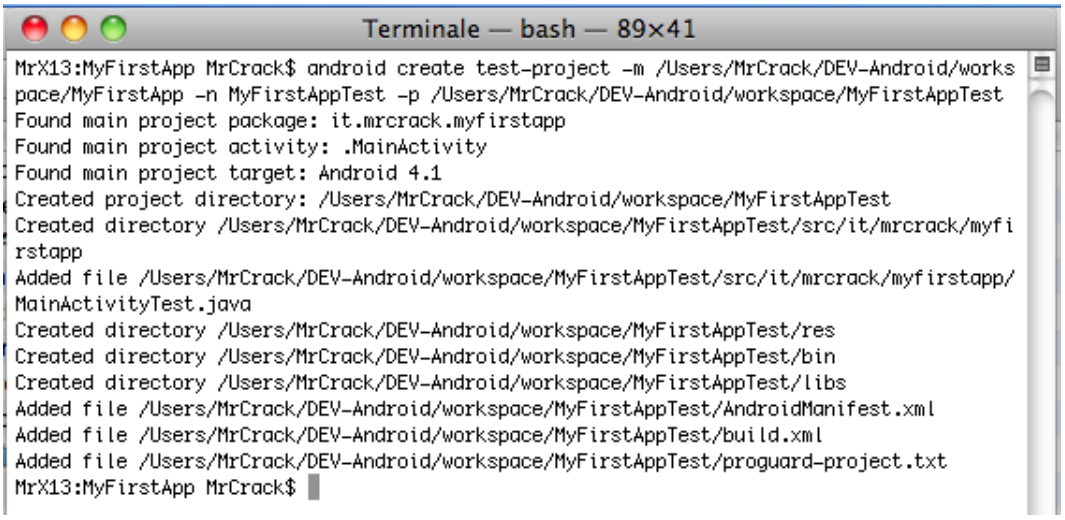

Figura 3.1: Creazione progetto di test da Shell

- Da Eclipse

  In Eclipse:

  1. File - New - Project...
  2. Android - Android Test Project.
  3. Compilazione dati richiesti dal Wizard.

3.1.3 Test di componenti specifici di Android

Mock

In fase di test, per facilitare l'iniezione di dipendenze, JUnit fornisce classi che creano mock per oggetti di sistema. Ad esempio:

- Context
- ContentProvider

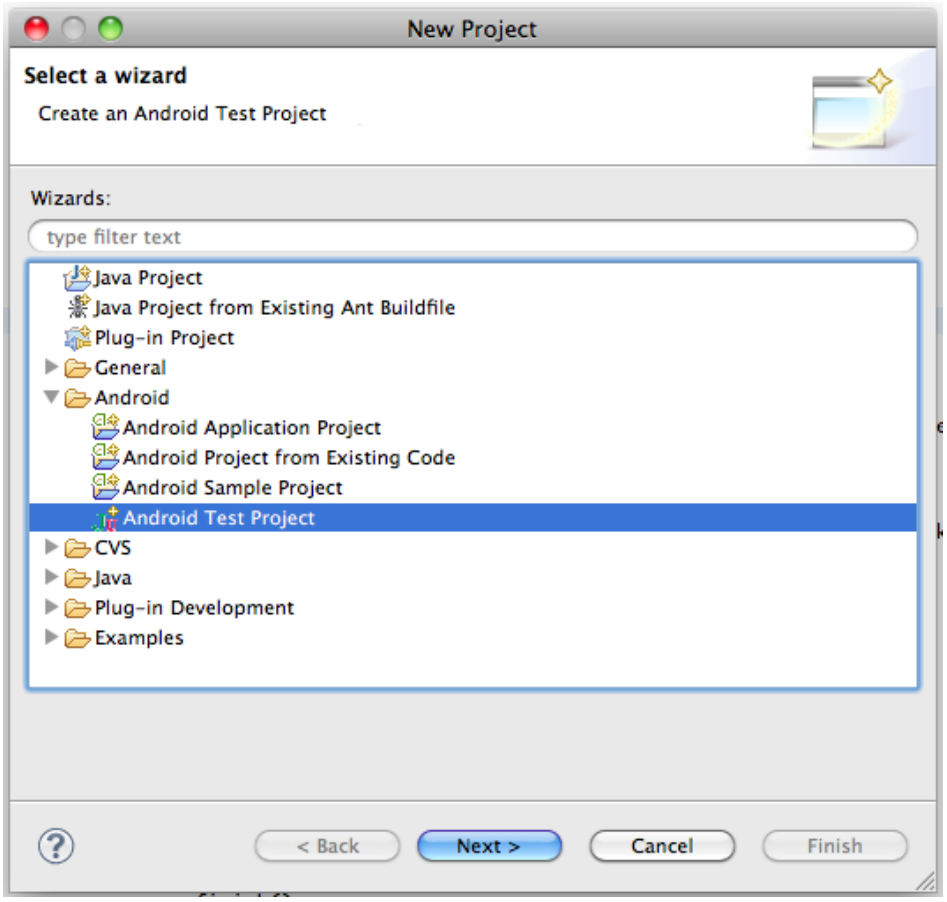

Figura 3.2: Creazione progetto di test in Eclipse

- ContentResolver
- Service
- Alcuni casi di test forniscono anche mock di Intent

Queste classi si trovano nel package android.test e in android.test.mock.

I mock sono in grado di isolare i test di un sistema in esecuzione spegnendo o ignorando le normali operazioni di Android. Ad esempio, MockContentResolver[23] espone i metodi:

- addProvider(String name, ContentProvider provider)

  Sostituisce il resolver di Android con quello proprio, isolato dal resto del sistema.

- notifyChange(Uri uri, ContentObserver observer, boolean syncToNetwork)

  Permette di isolare gli *Observer* di una risorsa dal sistema mock, in modo che non vi sia il rischio di attivarli accidentalmente.

Il Framework JUnit, inoltre, mette a disposizione la classi che permettono di creare mock con molta semplicita:

- MockApplication
- MockContext
- MockContentProvider
- MockCursor
- MockDialogInterface
- MockPackageManager
- MockResources

Queste classi, sono la *versione stub* dei corrispondenti oggetti di Android. Tutti i metodi di questi stub, nativamente, lanciano l'eccezione *UnsupportedOperationException*. L'unica operazione richiesta al programmatore è quella di implementare il comportamento dei metodi di cui necessita.

Testing delle Activity

Una Activity risponde a tre tipi di input:

1. Eventi generati dal ciclo di vita.
2. Eventi generati da input dell'utente.
3. Eventi generati dal sistema.

È consigliato, quindi, studiare dei *Test cases* che mettano alla prova il comportamento dell'App in queste situazioni (*Black box*). Se invece, si vogliono studiare casi di test a *basso livello* (*White box*), si possono utilizzare i tradizionali *Assert* di JUnit. Si potrebbe, per esempio, usare una serie di *Assert* per controllare le proprietà della UI durante l'esecuzione del test.

Come mostrato in figura 1.2 a pagina 15, il ciclo di vita di un'Activity comprende cinque stati e sette metodi che scatenano la transizione da uno stato all'altro. Grazie ai metodi:

1. callActivityOnCreate()
2. callActivityOnStart()
3. callActivityOnResume()
4. callActivityOnRestart()
5. callActivityOnPause()
6. callActivityOnStop()
7. callActivityOnDestroy()

messi a disposizione dalla *Classe Instrumentation*[19] è possibile testare il comportamento dell'App durante tutti i passaggi di stato.
Inoltre vanno studiati casi di test specifici per l'interazione dell'utente con la UI, in modo da testare che anche i questi casi l'App si comporti in modo corretto. Una possibile strategia per questi test è quella di stimolare tutti i possibili input messi a disposizione dalla UI; bisogna, inoltre, testare le sequenze di eventi, che coprono le funzionalità dell'Activity. Per eseguire questa tipologia di test è possibile utilizzare la *classe ActivityInstrumentationTestCase2*, questa, infatti, permette di effettuare test funzionali sulle Activity e di sollecitarle isolandola dagli altri componenti.
Tra le funzionalità non supportate da *JUnit* c'è il testing dei *Toast*, infatti, anche se largamente usati durante lo sviluppo delle *App*, non è possibile testarlo con quanto messo a disposizione dal *Framework*.
È evidente quanto sia difficoltoso lo studio dei casi di test in presenza di tutte le possibili interazioni dell'App con i *Sistemi estreni*, si dovrà quindi spendere una considerevole quantità di tempo per studiare ed implementare una valida *Suite di test* per una Activity.

Testing dei Service

Per il testing dei Service si possono fare discorsi analoghi rispetto al testing delle Activity, ovvero, bisogna concentrarsi sugli aspetti:

- Ciclo di vita
- Funzionalità

Il ciclo di vita di un Service (mostrato in figura 1.3 a pagina 17) può essere testato grazie alla classe *Contex*[18], che con i suoi metodi *startService*, *bindService* e *stopService* permette di testare il ciclo di vita del service e di monitorare la copertura delle funzionalità richieste al Service stesso. Bisogna prestare particolare attenzione al ciclo di vita dei services, in quanto è possibile invocare più volte, all'interno dell'App i comandi di start e di bind del service. Per ogni invocazione, però, non saranno create più istanze dello stesso service, bensì, verranno chiamati più volte i metodi caratterizzanti il ciclo di vita del service stesso, ovvero:

- startService invocherà *onStartCommand*.
- bindService invocherà *onBind*.

Testing dei Content Provider

Come già discusso nel paragrafo 1.2.4 a pagina 18, un Content Provider, mette a disposizione dei componenti non appartenenti all'App dati, a cui questi non potrebbero accedere.

Visto che l'accesso ai dati fisici avviene con delle API di Android, per automatizzare i test si creano dei Mock che svolgano le funzioni del SO (lettura dei dati fisici dal *disco*). Per fare ciò si usano le classi *IsolatedContext*[20] e *MockContentResolver*[23].

Per eseguire questa tipologia di test è possibile utilizzare la *classe ProviderTestCase2*, questa, infatti, permette di effettuare test funzionali sui *Content Provider* e di sollecitare tutti i metodi esposti.

Testing dei Broadcast Receiver

I Broadcast Receiver, come già discusso nel paragrafo 1.2.3 a pagina 18, sono dei componenti che permettono la ricezione di eventi di sistema o applicativi. Il *Framework* non mette a disposizione una apposita classe per testare questo componente, tuttavia, nella documentazione ufficiale di Android[13] è consigliato scrivere test per i componenti che inviano messaggi in broadcast, così da studiare il comportamento dei *Broadcast Receiver* atti a ricevere tali messaggi. È comunque possibile sfruttare la classe *AndroidTestCase* per verificare il corretto comportamento del *Broadcast Receiver*.

Test delle UI

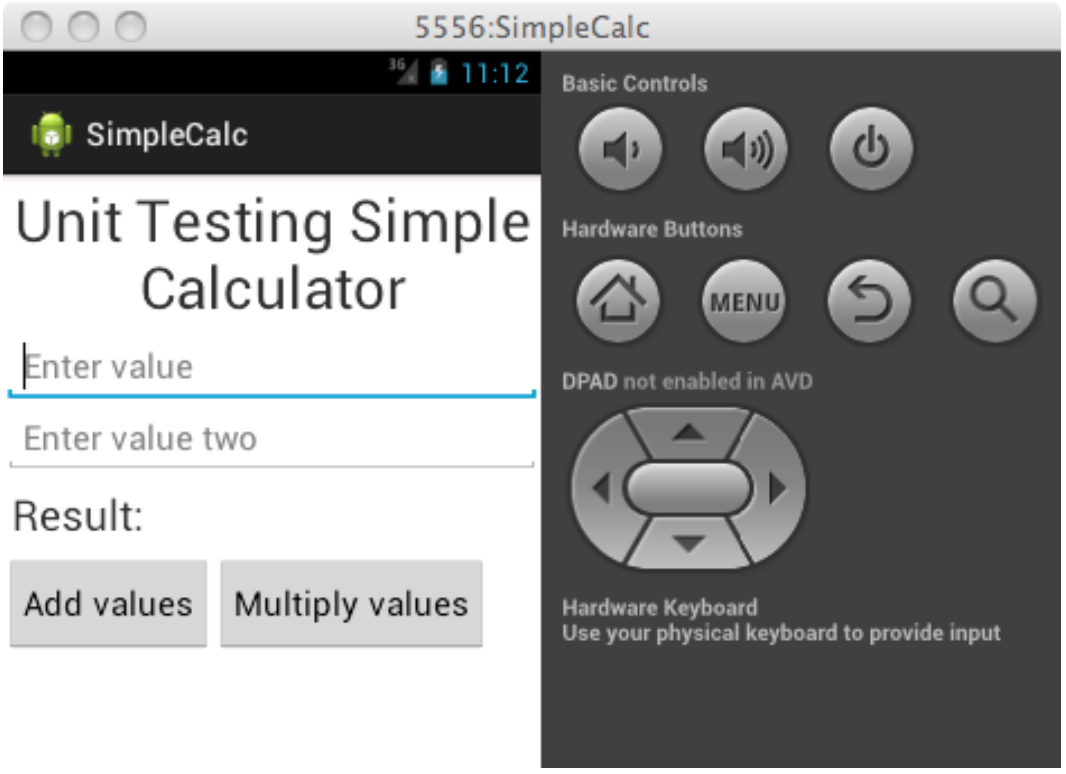

Figura 3.3: SimpleCalc in Portrait

Uno dei test più importanti da fare per la UI è quello di assicurarsi che tutti gli elementi grafici siano *interamente* contenuti all'interno del display. Questo è un test che bisogna ripetere per tutte le risoluzioni di cui si è interessati. Nel paragrafo A.3.1 a pagina 115 è presente un un esempio di test della UI applicato all'esempio riportato nell'appendice A a pagina 109 che

evidenzia detta problematica. Questo test non verrà sempre eseguito con successo:

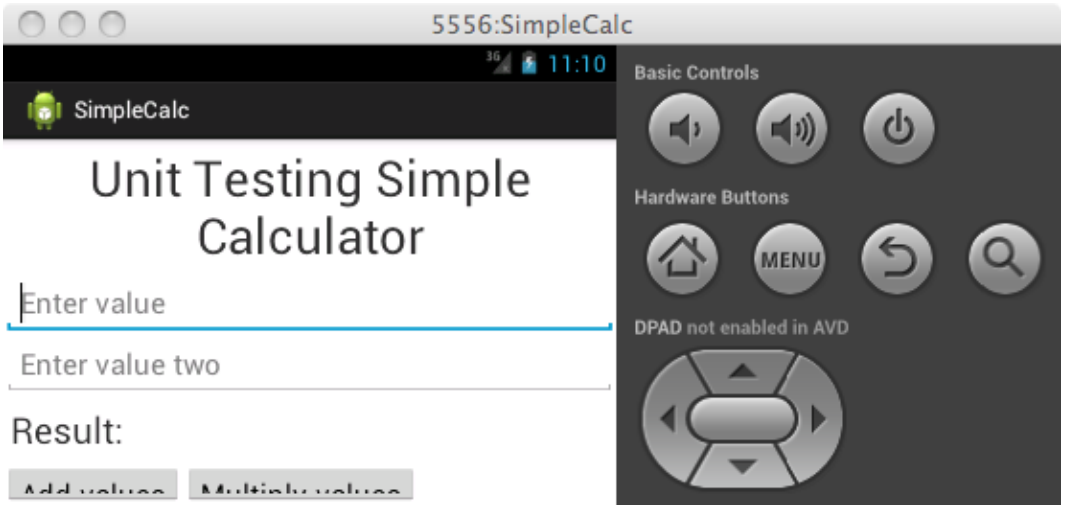

Figura 3.4: SimpleCalc in Landscape

1. 480x800, portrait mode (passato)
2. 800x480, landscape mode (fallito)
3. 320x480, portrait mode (fallito)
4. 480x320, landscape (fallito)
5. 480x854, portrait mode (passato)
6. 854x480, landscape mode (fallito)

Come è possibile vedere nella lista sopra riportata, per alcune risoluzioni il test fallisce (le immagini 3.3 a pagina 35 e 3.4 a pagina 36 sono un chiaro esempio del problema), ovvero, non tutti i componenti grafici dell'activity sono completamente visualizzabili. Da ciò si capisce che l'App non è utilizzabile su tutti i dispositivi in commercio.

3.1.4 Test delle classi java

Essendo il framework basato su JUnit, il test delle classi "puramente java", ovvero, delle classi che non utilizzano le librerie proprie di Android, si riducono a dei semplici test JUnit, che esulano dalla stesura di questo testo.

3.2 Robotium

3.2.1 Overview

Robotium è un *Framework* che sta riscuotendo un notevole successo grazie alla capacità che permette di creare sia test di tipo *Black-box* che di tipo *White-box*. Fornito con licenza Apache License 2.0, *Robotium* offre svariate *features*[27]:

1. Semplificare la scrittura di test.
2. Permettere la creazione di test anche con una conoscenza minima dell'App sotto test.
3. Il framework gestisce automaticamente molteplici attività Android.
4. Tempo minimo necessario per scrivere i casi di test solidi.
5. Leggibilità dei casi di test notevolmente migliorata, rispetto ai test scritti con strumentazione standard.
6. Casi di test più robusti grazie al binding a run-time dei componenti della GUI.
7. Esecuzione più rapida dei casi di test.
8. Si integra con Maven, Gradle o Ant per eseguire i test anche in continuous integration.

Robotium, infatti, permette di scrivere casi di test che siano:

- Black-box
- Robusti
- Automatici

Robotium può anche essere usato per testare le applicazioni di cui non sia disponibile il codice sorgente, o anche applicazioni installate in precedenza.
Robotium supporta il testing di:

- Activities

- Dialogs
- Toasts
- Menus
- Menu contestuali

3.2.2 Creazione di un progetto di Test

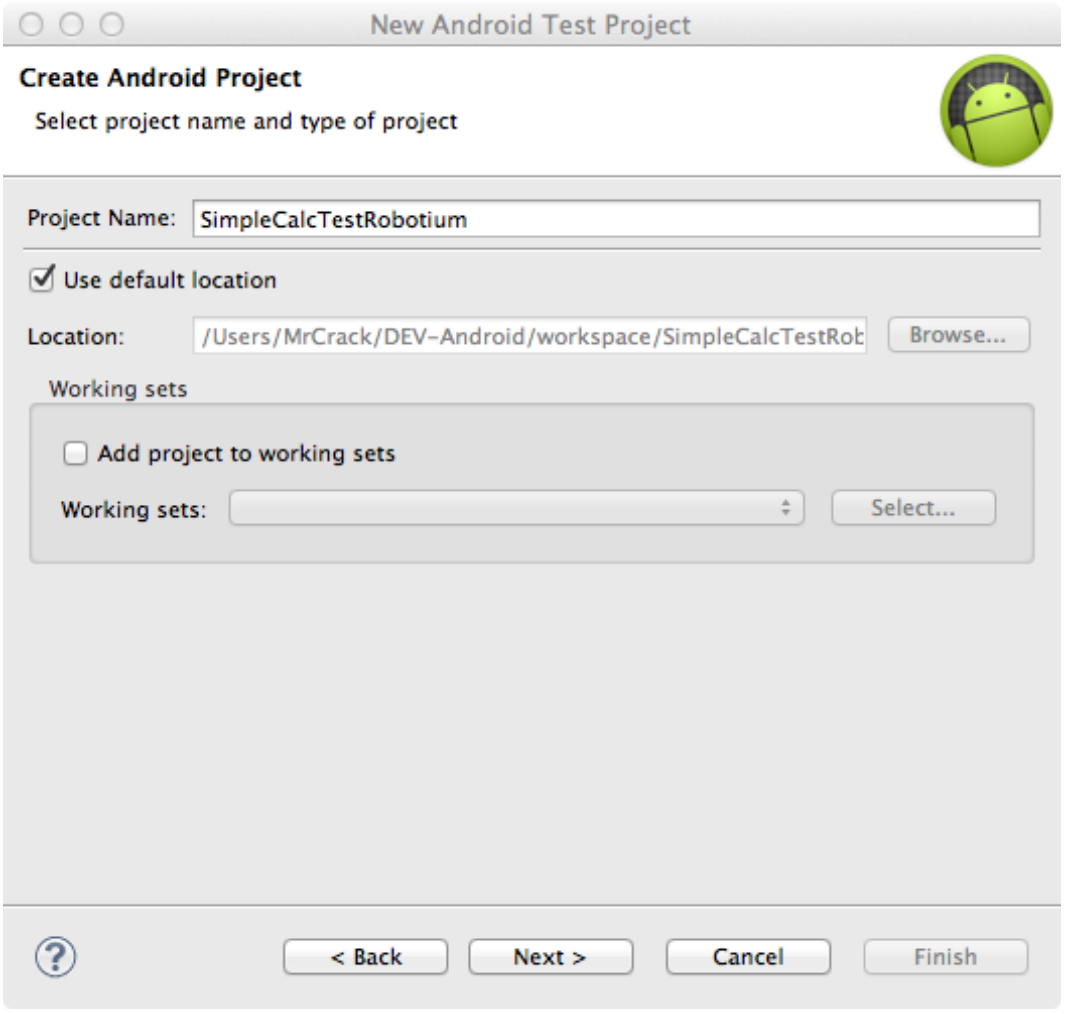

Figura 3.5: Nuovo Test Project Robotium

La creazione di un progetto di test Robotium e molto simile a quella di un progetto JUnit, infatti, di questo ne condivide la base del progetto. Una volta terminata la creazione del

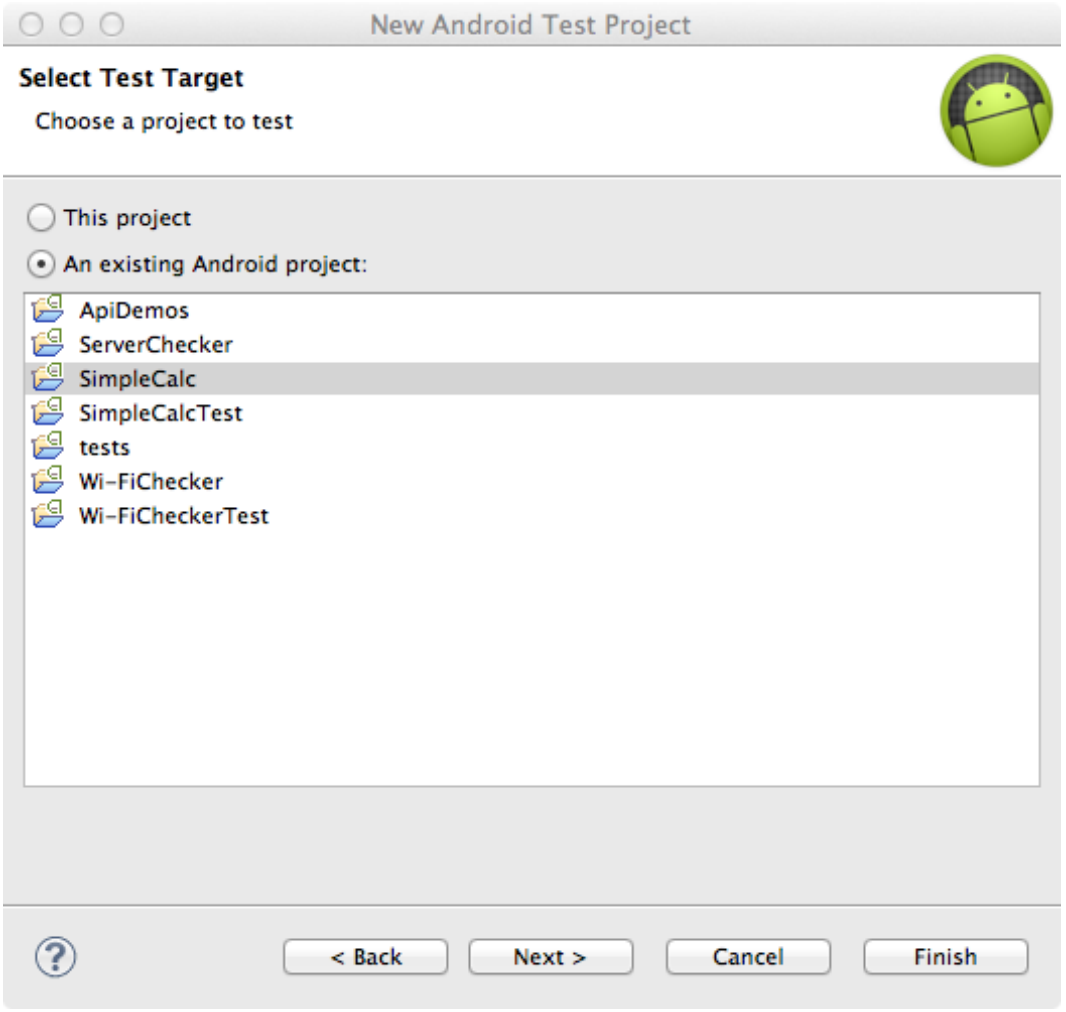

Figura 3.6: Nuovo Test Project Robotium - Selezione progetto target

progetto di test JUnit (come mostrato nelle figure 3.5 e 3.6) è necessario aggiungere le librerie Robotium Solo al *Build path* del progetto:

1. Scaricare le librerie dal sito di riferimento di Robotium
2. Click destro sul nome del progetto → Build Path → Configure Build Path
3. Spostarsi su *Libraries*
4. Add External JARs...
5. Selezionare il jar scaricato al punto 1 e confermare (mostrato in figura 3.8)
6. Spostarsi nella scheda *Order and Export*

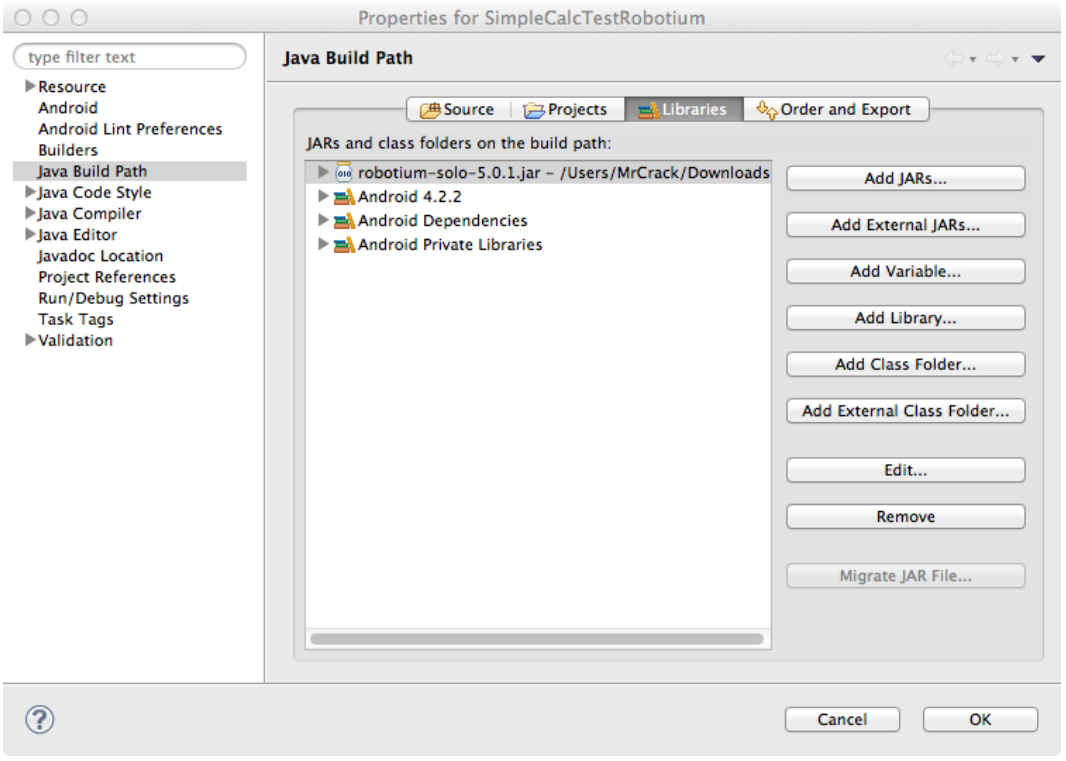

Figura 3.7: Nuovo Test Project Robotium - Aggiunta libreria Robotium

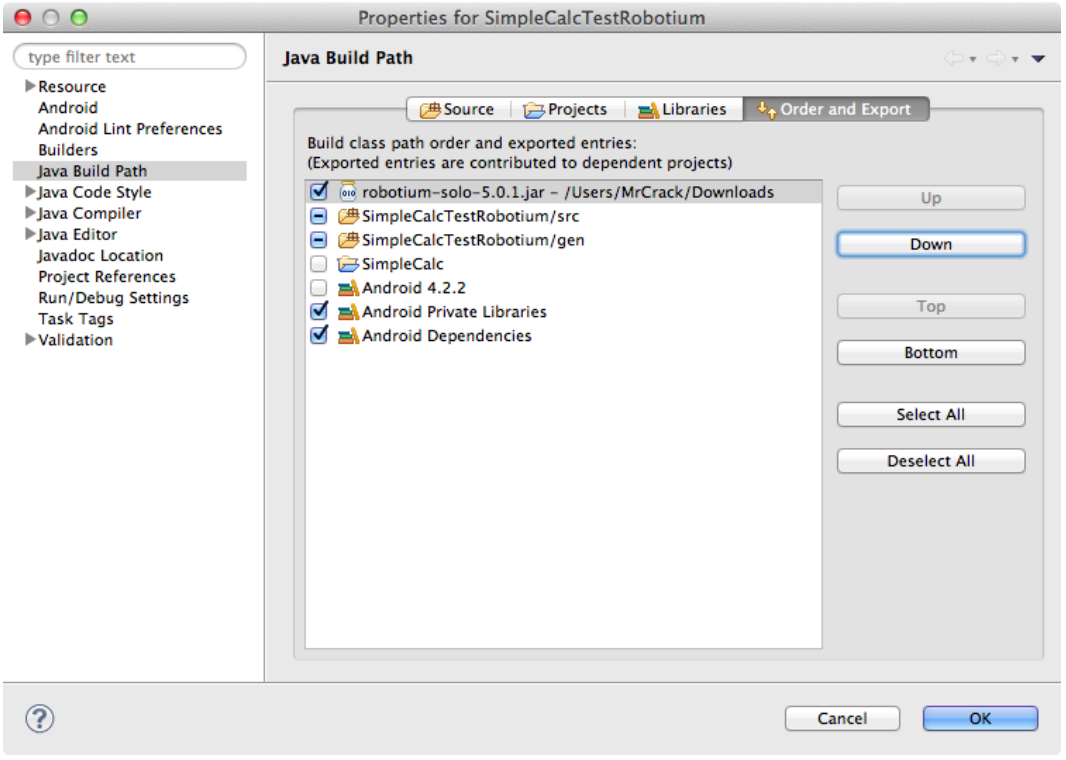

Figura 3.8: Nuovo Test Project Robotium - Export ed ordinamento libreria Robotium

7. Abilitare l'export della libreria appena aggiunta

8. Spostare la libreria in cima all'elenco

Anche se *Robotium* permette di effettuare il testing di App di cui non si dispongono i sorgenti, tutti gli esempi mostrati di seguito, sfrutteranno le librerie che prevedono il possesso dei sorgenti. Una volta creato il progetto, si potrà procedere con l'aggiunta delle classi di test. Le classi di test create con *Robotium* sono analoghe a quelle usate con JUnit, in aggiunta possono sfruttare le funzionalità messe a disposizione dalla libreria.

3.2.3 Test di componenti specifici di Android

Mock

Come indicato nei paragrafi precedenti, *Robotium* non permette la creazione di Mock, però, è possibile sfruttare le peculiarità messe a disposizione da JUnit, mostrate nel paragrafo 3.1.3.

Testing delle Activity

Il testing delle *Activity*, grazie a questa libreria diventa più agevole e rapido rispetto a quanto visto nel capitolo 3.1.3 con *JUnit*. Con poche righe di codice è possibile recuperare elementi della view che, altrimenti, con JUnit, sarebbe stato più complesso e laborioso estrarre. Di seguito è possibile vedere un confronto tra il recupero di una TextView con JUnit

```
final TextView pauseText = (TextView) getActivity().findViewById(
   it.mrcrack.wi_fichecker.R.id.onpause);
int pause = Integer.parseInt(pauseText.getText().toString());
getInstrumentation().callActivityOnPause(getActivity());
assertEquals(pause + 1,
   Integer.parseInt(pauseText.getText().toString()));
```

Sotto, la stessa operazione ottenuta sfruttando le librerie *Robotium*.

```
int pause = Integer.parseInt(solo.getText(10).getText().toString())
   ;
getInstrumentation().callActivityOnPause(getActivity());
assertEquals(pause + 1,
   Integer.parseInt(solo.getText(10).getText().toString()));
```

L'esempio precedente mostra come sia possibile recuperare un elemento della view conoscendone la posizione. È ancora più interessante vedere che *Robotium* permette anche il recupero di elementi grafici a partire da testo che questo visualizza.

```
public void testStartButt() {
  solo.sleep(5000);
  solo.waitForActivity(MainActivity.class, 5000);
  solo.clickOnButton("List networks");
  assertTrue(solo.waitForActivity(ListActivity.class, 5000));
}
```

Confrontando il precedente blocco di codice con quello usato per il test *JUnit* è possibile vedere che *Robotium* permette l'uso di tali elementi anche senza l'esplicita richiesta di utilizzare tale elemento all'interno del *Threade* della UI.

```
public void testStartButt() {
  Instrumentation.ActivityMonitor listAct = getInstrumentation()
     .addMonitor(ListActivity.class.getName(), null, false);

  final Button startButt = (Button) getActivity().findViewById(
     it.mrcrack.wi_fichecker.R.id.button1);

  getActivity().runOnUiThread(new Runnable() {
    @Override
    public void run() {
      // click button and open next activity.
      startButt.performClick();
    }
  });

  Activity currAct = getInstrumentation().waitForMonitorWithTimeout(
     listAct, 10000);
  assertNotNull(currAct);
  currAct.finish();
}
```

Il precedente confronto riesce a far comprendere le grandi potenzialità messe a disposizione da questa libreria. Le API di questo *Framework* offrono innumerevoli metodi per la scrittura di *Test Android*, di seguito se ne analizzeranno i principali:

- *void clickOnXxx(int index)*: Effettua il click sull'elemento del tipo Xxx a partire dalla sua posizione (rappresentata dall'indice accettato come parametro di ingresso).
- *void clickOnXxx(int index)*: Effettua il click sull'elemento del tipo Xxx a partire dalla sua posizione (rappresentata dall'indice accettato come parametro di ingresso).

- *void drag(float fromX, float toX, float fromY, float toY, int stepCount)*: Simula il tocco dalla punto specificato, con successivo trascinamento fino al nuovo punto specificato.
- *void enterText(android.widget.EditText editText, String text)*: Inserisce il testo richiesto all'interno della EditText richiesta.
- *void enterText(int index, String text)*: Inserisce il testo richiesto all'interno della EditText richiesta a partire dalla sua posizione (rappresentata dall'indice
- *void enterTextInWebElement(By by, String text)*: Inserisce il testo richiesto all'interno della WebElement richiesta.
- *Xxx getXxx(int index)*: Permette il recupero di un elemento della view a partire dalla sua posizione (rappresentata dall'indice accettato come parametro di ingresso).
- *Xxx getXxx(String text)*: Permette il recupero di un elemento della view a partire dalla stringa in esso visualizzato (questo metodo non è disponibile per tutti i tipi di view, ma solo per le view che mostrano del testo).

Testing di Service - Content Provider - Broadcast Receiver

Robotium è da considerarsi, più che come uno strumento indipendente, come un estensione di *JUnit*, infatti, è possibile scrivere classi di test che sfruttino sia le funzionalità di *JUnit*, che quelle di *Robotium*.

Per il testing di:

- Service
- Content Provider
- Broadcast Receiver

Robotium non espone metodi particolari che aiutino il programmatore, sarà quindi, necessario sfruttare quanto visto nel paragrafo precedente per risolvere tali problematiche.

3.3 Calabash

Calabash è uno strumento *OpenSource*, coperto da *Eclipse Public License 1.0*, che consente di scrivere ed eseguire *Acceptance tests* automatizzati di App. *Calabash* supporta applicazioni native *Android* e *iOS* e consiste in una serie librerie, che consentono di interagire con una App, in modo *Programmatico*, sia essa nativa o ibrida. L'interazione che è possibile programmare dovrebbe rispecchiare un certo numero di azioni di un utente comune, quindi le azioni possono consistere in[16]:

- Gestures
- Touches o gesti (es. *tappare*, scorrere e ruotare)
- asserzioni (es.: ci dovrebbe essere un pulsante Login o la visualizzazione web deve contenere un elemento <h1> con il testo Ciao)
- Screenshots della vista corrente del modello di dispositivo corrente

I test *Calabash* possono essere eseguiti sia su *Device emulati* che su *Devices fisici*.

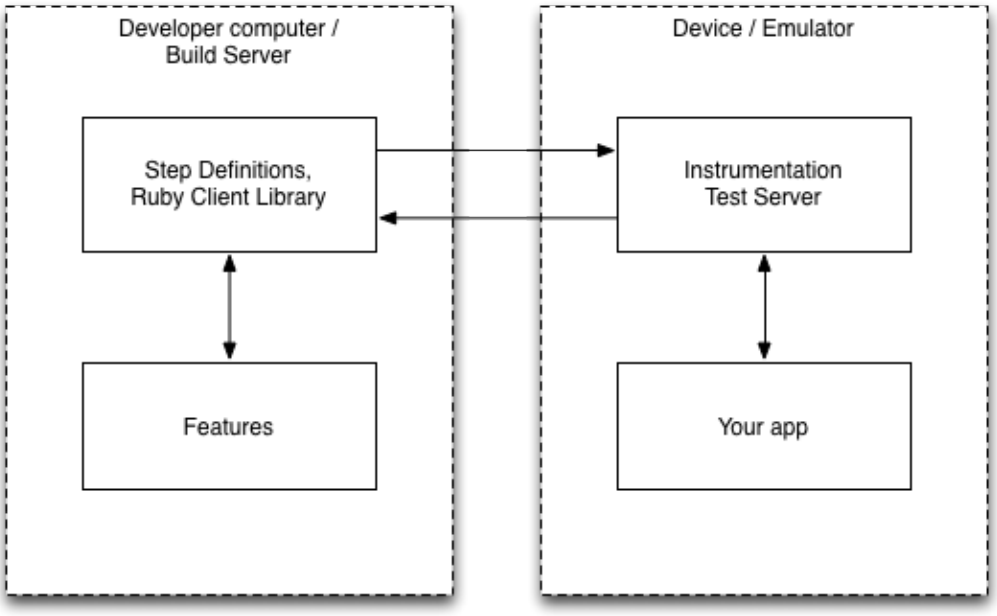

Figura 3.9: Calabash - Android Architecture[17]

Una delle principali peculiarità di *Calabash* è quella di basarsi su *Cucumber*, tool scritto in *Ruby* che permette la scrittura di *Acceptance test* in *linguaggio naturale*. *Cucumber*, a sua volta si basa sul Behavior-Driven Development (BDD), processo di sviluppo software che unisce le peculiarità di:

- *Test-Driven Development (TDD)*: Processo di sviluppo software che si basa sulla scrittura preventiva dei test. Una volta scritti i test, è necessaria la scrittura della *minima quantità* di codice per il superamento del test, infine bisogna rivisitare il codice e renderlo *compliant* agli standard di qualità.

- *Domain-Driven Design (DDD)*: Processo di sviluppo software che pone come centro focale del progetto il *Dominio* e le sue logiche di funzionamento e spingendo per una forte collaborazione tra i progettisti del sistema e gli esperti del dominio.

- *Object-Oriented Analysis and Design (OOAD)*: È un approccio tecnico molto diffuso per l'analisi e la progettazione di un'applicazione o un sistema. Questo sfrutta il paradigma *object-oriented* ed il paradigma di *modellazione visuale* durante i cicli di vita dello sviluppo, per favorire una migliore qualità del prodotto.

facendo in modo di fornire strumenti condivisi tra *Analisti funzionali* e *Sviluppatori*, che gli permettano di collaborare fattivamente alla creazione dell'App, per esempio, permettendo agli *Analisti* di scrivere test a monte degli sviluppi, facendo così emergere le anomalie già durante i primi sviluppi.
In figura 3.9 a pagina 44, è possibile vedere l'interazione che c'è tra il PC ed il *Device* durante i test. In particolare possiamo riconoscere:

- Features

  Il file *feature* contiene tutti i passaggi che devono essere effettuati durante il test. Durante una singola esecuzione sarà possibile testare più caratteristiche.

- Step Definitions

In *Calabash Android* è già presente un set di step predefiniti. Questi step sono generici e pronti per essere utilizzati rapidamente. È comunque possibile definire nuovi step, sfruttando una terminologia progettuale, propria del *dominio del problema*. Ad esempio, un'applicazione orientata al mondo *financial* potrebbe contenere uno step del tipo *Trasferire i contanti al mio account*.

- Instrumentation Test Server

  App installata ed eseguita da *Calabash Android*, contiene *ActivityInstrumentationTestCase2* dell'Software Development Kit (SDK) di *Android*.

- Your app

  App sotto test, grazie all'*Instrumentation Test Server* non è necessario apportarvi alcuna modifica per l'esecuzione dei test.

3.3.1 Esempio di utilizzo

Il test è stato effettuato sia sfruttando l'emulatore offerto dall'ambiente di sviluppo di *Android*, sia sfruttando un *Device hardware*. L'ambiente di test prevede l'utilizzo di Ruby, quindi, come riportato sul sito ufficiale del progetto, sarà necessario installare *Ruby*, e poi installare il pacchetto *gem Calabash*.
A valle dell'installazione bisogna:

- Creare un folder da utilizzare per i test
- Lanciare il comando *calabash-android gen*
- Modificare il file *my_first.feature* presente nel folder *features*
- Lanciare l'emulatore o collegare il *Device* in porta USB
- Lanciare il comando *calabash-android run <apk>*

Dopo aver lanciato lo script, come indicato sopra, vengono installati l'*Instrumentation Test Server* e l'App da testare, poi vengono eseguiti tutti i passi presenti nel file *my_first.feature*, e viene prodotto l'output disponibile nel prossimo sottoparagrafo.

Funzionalità da testare

- Moltiplicazione numeri positivi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = 4
  - Output atteso: 8
  - Output ottenuto: 8
- Somma numeri positivi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = 4
  - Output atteso: 6
  - Output ottenuto: 6
- Controllo Input
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = a

- Output atteso: Apertura nuova *Activity* con istruzioni dell'App
- Output ottenuto: Apertura nuova *Activity* con istruzioni dell'App

- Moltiplicazione numeri negativi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = -1
  - Output atteso: -2
  - Output ottenuto: 4

- Somma numeri negativi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = -1
  - Output atteso: 1
  - Output ottenuto: 1

Test

Lo script necessario ad effettuare i test dell'App *SimpleCalc* è:

```
Feature: Somma e moltiplicazione

Scenario: vengono inseriti 2 e 4
  Then I enter text "2" into field with id "value1"
  Then I enter text "4" into field with id "value2"
  Then I press view with id "multiplyValues"
  Then I see the text "8"
  Then I press view with id "addValues"
  Then I see the text "6"
```

```
  Then I enter text "a" into field with id "value2"
  Then I press view with id "multiplyValues"
  Then I see the text "Instructions"

  Then I press view with id "back"
  Then I press view with id "addValues"
  Then I see the text "Instructions"

  Then I press view with id "back"
  Then I enter text "-1" into field with id "value2"
  Then I press view with id "multiplyValues"
  Then I see the text "-2"

  Then I press view with id "addValues"
  Then I see the text "1"
```

L'esecuzione del test ha generato il seguente output:

```
c:\DEV\calabash>calabash-android run
        C:\DEV\svnRaspi\sources\java\android\simple
calc\bin\SimpleCalc.apk
*** WARNING: You must use ANSICON 1.31 or
higher (https://github.com/adoxa/ansicon/)
 to get coloured output on Windows
Feature: Somma e moltiplicazione

  Scenario: vengono inseriti 2 e 4
                     # features\my_first.feature:3
731 KB/s (556651 bytes in 0.743s)
1391 KB/s (199476 bytes in 0.140s)
    Then I enter text "2" into field with id "value1"
        # calabash-android-0.4.20
/lib/calabash-android/steps/enter_text_steps.rb:25
    Then I enter text "4" into field with id "value2"
        # calabash-android-0.4.20
/lib/calabash-android/steps/enter_text_steps.rb:25
    Then I press view with id "multiplyValues"
        # calabash-android-0.4.20
/lib/calabash-android/steps/press_button_steps.rb:13
    Then I see the text "8"
        # calabash-android-0.4.20
/lib/calabash-android/steps/assert_steps.rb:1
    Then I press view with id "addValues"
        # calabash-android-0.4.20
/lib/calabash-android/steps/press_button_steps.rb:13
```

Figura 3.10: Calabash - Screenshot generato dal test

```
    Then I see the text "6"
        # calabash-android-0.4.20
/lib/calabash-android/steps/assert_steps.rb:1
    Then I enter text "a" into field with id "value2"
         # calabash-android-0.4.20
/lib/calabash-android/steps/enter_text_steps.rb:25
    Then I press view with id "multiplyValues"
         # calabash-android-0.4.20
/lib/calabash-android/steps/press_button_steps.rb:13
    Then I see the text "Instructions"
         # calabash-android-0.4.20
/lib/calabash-android/steps/assert_steps.rb:1
    Then I press view with id "back"
         # calabash-android-0.4.20
/lib/calabash-android/steps/press_button_steps.rb:13
    Then I press view with id "addValues"
```

```
          # calabash-android-0.4.20
/lib/calabash-android/steps/press_button_steps.rb:13
    Then I see the text "Instructions"
          # calabash-android-0.4.20
/lib/calabash-android/steps/assert_steps.rb:1
    Then I press view with id "back"
          # calabash-android-0.4.20
/lib/calabash-android/steps/press_button_steps.rb:13
    Then I enter text "-1" into field with id "value2"
          # calabash-android-0.4.20
/lib/calabash-android/steps/enter_text_steps.rb:25
    Then I press view with id "multiplyValues"
          # calabash-android-0.4.20
/lib/calabash-android/steps/press_button_steps.rb:13
    Then I see the text "-2"
          # calabash-android-0.4.20
/lib/calabash-android/steps/assert_steps.rb:1
      Action 'assert_text' unsuccessful: Text'-2' was
             not found (RuntimeError)
      C:/Ruby/Ruby193/lib/ruby/1.9.1/timeout.rb:69:in `timeout'
      features\my_first.feature:23:in `Then I see the text "-2"'
    Then I press view with id "addValues"
             # calabash-android-0.4.20
/lib/calabash-android/steps/press_button_steps.rb:13
    Then I see the text "1"
             # calabash-android-0.4.20
/lib/calabash-android/steps/assert_steps.rb:1

Failing Scenarios:
cucumber features\my_first.feature:3
             # Scenario: vengono inseriti 2 e 4

1 scenario (1 failed)
18 steps (1 failed, 2 skipped, 15 passed)
0m48.916s
```

Inoltre, sono stati generati due screenshot (uguali), rappresentanti le activity *navigate*, visibile in figura 3.10 a pagina 50.

3.4 Selenium Android WebDriver

Selenium Android WebDriver è uno strumento OpenSource coperto da licenza *Apache License 2.0* per l'*Automation testing* di *WebApp*. Questo strumento è indicato per essere utilizzato

da sviluppatori web che ottimizzano siti per i *Browser Mobili*, facendo quindi in modo che il sito sia correttamente consultabile anche dal browser di un *Device*.
I test di *Selenium Android WebDriver* sono test end-to-end che esercitano la *WebApp* come farebbe un utente reale:

- Tocchi brevi
- Tocchi prolungati
- Scroll
- Rotazione dello schermo
- etc. etc.

Inoltre, possono sfruttare, anche le caratteristiche messe a disposizione dall'*HTML5*, come:

- Local storage
- Session storage
- Application cache

WebDriver è composto da due componenti principali:

- server
- i test

Il server è un'applicazione che gira sul *Device* ed ascolta richieste in arrivo. Esegue i test in una WebView configurata come un browser. I test vengono eseguiti sul client e possono essere scritti in tutte le lingue supportate da WebDriver, tra cui Java e Python. I test WebDriver comunicano con il server attraverso richieste RESTful JSON su protocollo HTTP. I test e i componenti del server non devono obbligatoriamente trovarsi sulla stessa macchina. Per i test su *Devices* Android è anche possibile sostituire il framework di test Android al server WebDriver remoto.

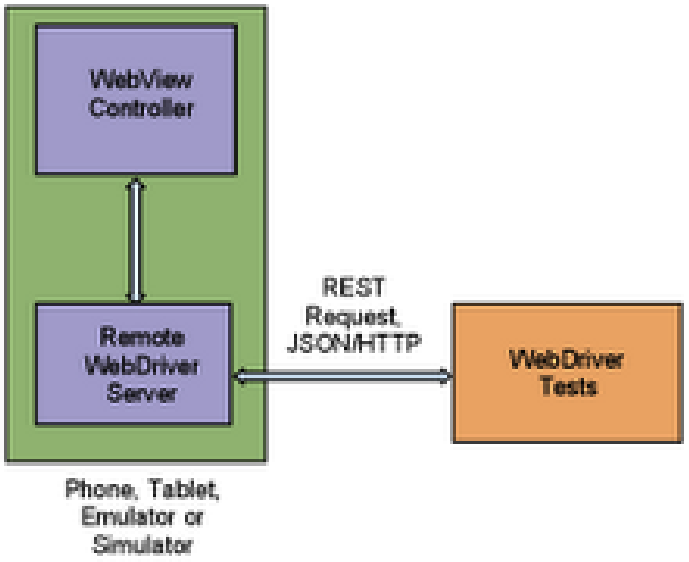

Figura 3.11: Componenti di Selenium Android WebDriver[30]

Questo strumento, come la maggior parte dei tool testuali, si basa sui Junit. I test prodotti, potranno essere lanciati da Eclipse o riga di comando. I test possono essere integrati in un sistema di CI, oppure, possono essere eseguiti da un *Device*. Per eseguire il test, il tool apre una *WebView* configurata come *Browser Android* e vi apre la *WebApp* sotto test.
La creazione di un progetto *Selenium Android WebDriver* consiste nella creazione, attraverso l'ambiente di sviluppo nativo di *Android*, di un *progetto Android* senza interfaccia grafica[29]:

```
public class SimpleAppActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Creare un progetto di test *Android*, *Selenium*, in maniera del tutto trasparente, creerà una *WebView* inserendola automaticamente all'interno della *Main Activity*. La creazione dei test, anche se è testuale, è resa semplice ed intuitiva. Vediamo un esempio, dove:

- Si effettuerà accesso ad un motore di ricerca.
- Ricerchi una frase.

- Controlli che si vedano correttamente i vari componenti della pagina.

```
public class SimpleGoogleTest extends
   ActivityInstrumentationTestCase2<SimpleAppActivity> {

    public void testGoogleShouldWork() {
      // Create a WebDriver instance with the activity in
      // which we want the test to run
      WebDriver driver = new AndroidDriver(getActivity());
      // LetÕs open a web page
      driver.get("http://www.google.com");

      // Lookup for the search box by its name
      WebElement searchBox = driver.
            findElement(By.name("q"));

      // Enter a search query and submit
      searchBox.sendKeys("weather in san francisco");
      searchBox.submit();

      // Making sure that Google shows 11 results
      WebElement resultSection = driver.findElement(By.id("ires"));
      List<WebElement> searchResults = resultSection.
            findElements(By.tagName("li"));
      assertEquals(11, searchResults.size());

      // Let's ensure that the first result shown is the
            weather widget
      WebElement weatherWidget = searchResults.get(0);
      assertTrue(weatherWidget.getText().
            contains("Weather for San Francisco, CA"));
    }
}
```

Eseguendo il precedente esempio, come già precedentemente accennato, verrà creata una WebView con la stessa configurazione del browser Android. Così, sarà possibile visualizzare il comportamento del test direttamente dal display del *Devices*, o della macchina virtuale se non si sta utilizzando un *Device fisico*.

3.5 Appium

Appium è un *Framework OpenSource* coperto da *Apache License v. 2.0*, che supporta l'*Automation testing* di App native ed *Ibride* per i SO:

- *Android*
- *iOS*
- *FirefoxOS*

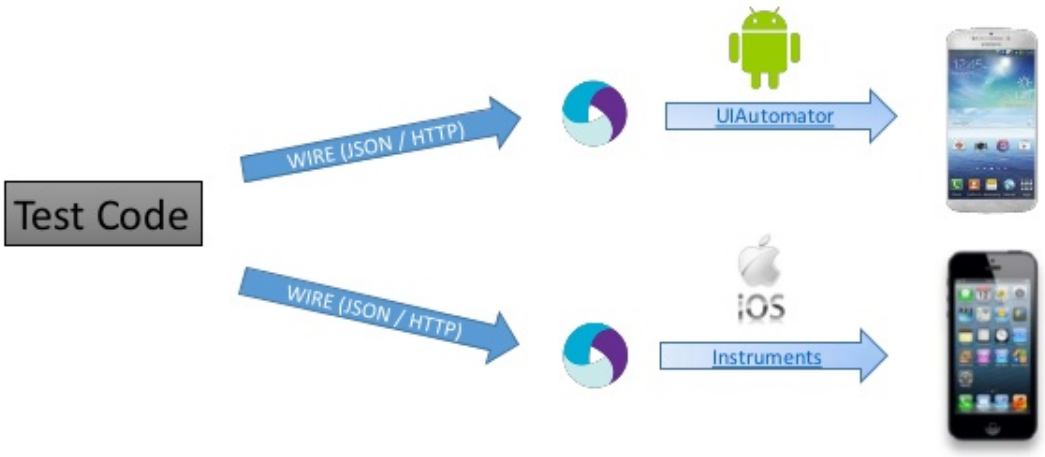

Figura 3.12: Appium - Logica di funzionamento[15]

Appium è stato pensato e sviluppato con specifiche peculiarità:

- Testare App senza apportarvi alcuna modifica
- Permettere di scrivere test in qualsiasi linguaggio e su qualsiasi piattaforma
- Supportare le specifiche delle *WebDriver API*

L'API WebDriver mira a fornire delle API sincrone che possono essere utilizzate per svariati casi d'uso, anche se, inizialmente progettate per supportare l'automation testing di applicazioni web.[33]
Appum, quindi, permette l'utilizzo di qualsiasi linguaggio che compatibile con le specifiche *WebDriver* come:

- Java
- Objective-C
- JavaScript

- Node.js
- PHP
- ...

Inoltre, è possibile utilizzare questo strumento anche in ambiente di CI, rendendolo ideale per questo ambito. Con l'utilizzo di WebDriver, i progettisti hanno fatto in modo di non vincolare lo strumento a nessuna tecnologia specifica, bensì, hanno demandato la scelta ai tester, che, in base alle proprie esigenze/esperienze sono liberi di scegliere quale strumento utilizzare.

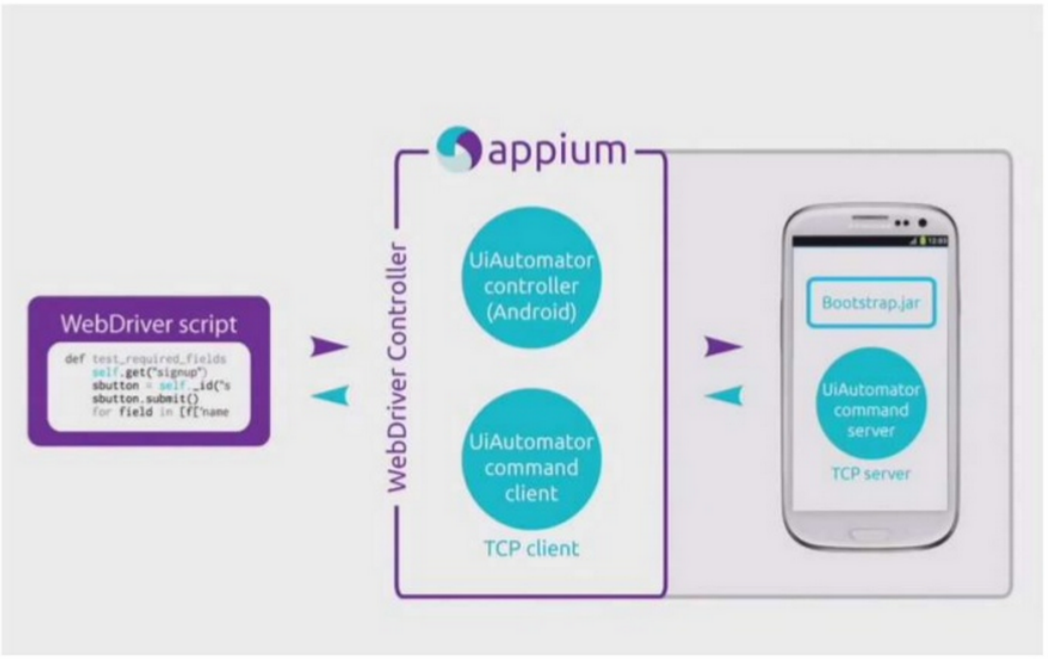

Figura 3.13: Appium - Architettura[15]

3.5.1 Esempio di utilizzo

Per utilizzare *Appium* sono necessari

- Linux/OSX (l'ambiente per Windows è al momento in beta)
- Android SDK > 15
- node.js con npm

Una volta in possesso di tutti i requisiti, è possibile installare *Appium* lanciando da riga di comando:

```
> npm install -g appium
> npm install wd
> appium &
```

Funzionalità da testare

- Moltiplicazione numeri positivi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = 4
  - Output atteso: 8
  - Output ottenuto: 8
- Somma numeri positivi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = 4
  - Output atteso: 6
  - Output ottenuto: 6
- Controllo Input
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = a

  - Output atteso: Apertura nuova *Activity* con istruzioni dell'App
  - Output ottenuto: Apertura nuova *Activity* con istruzioni dell'App

- Moltiplicazione numeri negativi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = -1
  - Output atteso: -2
  - Output ottenuto: 4

- Somma numeri negativi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = -1
  - Output atteso: 1
  - Output ottenuto: 1

Test

Lo script necessario ad effettuare i test dell'App *SimpleCalc* sono:

```
@Test
public void apiDemo() {
   WebElement value1 = driver.findElement(By.name("value1"));
   WebElement value2 = driver.findElement(By.name("value2"));
   WebElement multiplyValues = driver.findElement(By
     .name("multiplyValues"));
   WebElement addValues = driver.findElement(By.name("addValues"));
   WebElement result = driver.findElement(By.name("result"));
   WebElement back = null;
```

```
    value1.sendKeys("2");
    value2.sendKeys("4");

    multiplyValues.click();
    assertEquals("8", result.getText());
    addValues.click();
    assertEquals("6", result.getText());

    value2.clear();
    value2.sendKeys("a");
    multiplyValues.click();

    // Se non trovato genera una NoSuchElementException
    // cosi' fallisce il test
    back = driver.findElement(By.name("back"));
    back.click();

    multiplyValues = driver.findElement(By.name("multiplyValues"));
    multiplyValues.click();

    back = driver.findElement(By.name("back"));
    back.click();

    value2 = driver.findElement(By.name("value2"));
    multiplyValues = driver.findElement(By.name("multiplyValues"));
    addValues = driver.findElement(By.name("addValues"));

    value2.clear();
    value2.sendKeys("-1");
    multiplyValues.click();
    assertEquals("-2", result.getText());
    addValues.click();
    assertEquals("1", result.getText());
}
```

3.6 Selendroid

Selendroid è un *Framework OpenSource* coperto da *Apache License v. 2.0*, che supporta l'*Automation testing* di App *Android*:

- Native
- Ibride

- Web

Selendroid è un progetto che si basa su *Selenium*, o, per meglio dire, è un *Fork* di quest'ultimo per il solo testing di App *Android*; è particolarmente adatto ad essere utilizzato durante gli sviluppi, infatti, è facile inserirlo all'interno di un contesto di CI . Le principali funzionalità messe a disposizione da *Selendroid* sono:

- Piena compatibilità con il JSON Wire Protocol (API WebDriver, sul modello di quelle esposte da *Appium*, che espongono *WebServices RESTful* ed accettano oggetti JSON)
- Non è richiesta alcuna modifica alle App da testare
- Permette di verificare *WebApp*, sfruttando, come fatto da *Selenium Android WebDriver* una WebView
- Supporto delle *Gesture*
- È possibile testare l'App su più *Devices* (sia reali che emulati) contemporaneamente

Selendroid contiene quattro componenti principali:

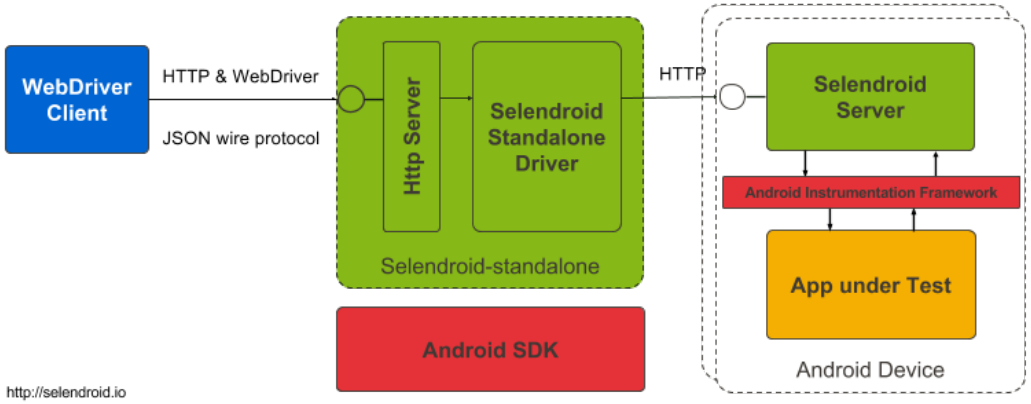

Figura 3.14: Selendroid - Architettura del Framework[28]

- *Selendroid-Client*: la libreria java del client (basata sul client java di *Selenium*).
- *Selendroid-Server*: è un modulo che verrà eseguito contemporaneamente all'App da testare.

- *AndroidDriver-App*: al il compito di testare *WebApp* richiamando il sito all'interno di una WebView.
- *Selendroid-Standalone*: ha il compito di gestire simultaneamenteo più dispositivi Android e di installarvi l'App sotto test.

3.6.1 Esempio di utilizzo

L'esempio di utilizzo di *Selendroid* risulta essere lo stesso visto per *Appium*, infatti, come per questo, il server installato sul *Device* utilizza il *JSON Wire Protocol* protocollo utilizzato dalle librerie Java già utilizzate nel capitolo precedente: *org.openqa.selenium.WebDriver*.

3.7 Android GUI Ripper

http://wpa
http://wpa

Android GUI Ripper è un tool di *Automation Testing* che, automatizza il testing delle App sfruttando la Graphical User Interface (GUI) esposta dall'App stessa[11].

Il funzionamento di *Android GUI Ripper* si basa sull'esplorazione automatica della GUI dell'App, con lo scopo di esercitare una serie di possibili input in modo strutturato, strategia, molto simile a quella utilizzata per il *Web crawling*. Oltre ad effettuare l'esplorazione della GUI, il tool:

- Ricerca di *Crash*
- GUI model reverse engineering
- Generazione di *Test cases JUnit*

Questo tipo di approccio permette di scoprire malfunzionamenti che, potrebbero altrimenti, non essere trovati. Infatti, un *operatore umano*, potrebbe non pensare a tali casistiche. D'altronde, adottando solo questa strategia potrebbero non essere trovati altri tipi di malfunzionamento.

TODO

3.7.1 Esempio di utilizzo

3.8 A^{3}E

A^3E è uno strumento distribuito sotto *licenza BSD con 3 clausole*, che aiuta gli sviluppatori ed i tester a scrivere test di App *Android*. Questo strumento, infatti, facilita gli sviluppatori a produrre una sequenza di interazioni con l'App, simile a quella prodotta da un normale utente. I progettisti di A^3E hanno pensato il prodotto in modo che avesse le seguenti peculiarità[9]:

- Non essere vincolato all'esclusivo utilizzo con i simulatori, ma, anche con diversi telefoni *Android* presenti sul mercato.
- Dare la possibilità di utilizzare il tool anche in assenza di codice sorgente.
- Cercare di scoprire le *Activities* nascoste per un normale esplorazione utente e di lanciarle per testarne il comportamento.

La strategia di esplorazione delle *Activities* implementata in A^3E sfrutta due tecniche:

- Targeted Exploration (TE)
- Depth-First Exploration (DFE)

3.8.1 Static Activity Transition Graph

A^3E è stato progettato sfruttando il *concetto* di Static Activity Transition Graph (SATG). Il principio su cui si basa l'SATG afferma che qualora ci fosse una transizione da A a B, potenzialmente ci potrebbe essere un flusso di dati che viaggia nello stesso verso. Questa relazione è ulteriormente verificata da un'attenta analisi effettuata con *SCanDroid*.

Il SATG è un grafo di transizione che mostra le relazioni che legano tra loro le *Activity*. Ogni nodo del grafico è un'*Activity* all'interno dell'applicazione. Si tratta di un grafo orientato, dove la freccia dall'*Activity* A all*Activity* B indica una transizione nello stesso verso. Per validare una transazione viene effettuata una accurata analisi per verificare se un determinato percorso da un'*Activity* di origine a una di destinazione esiste realmente o no. Questo

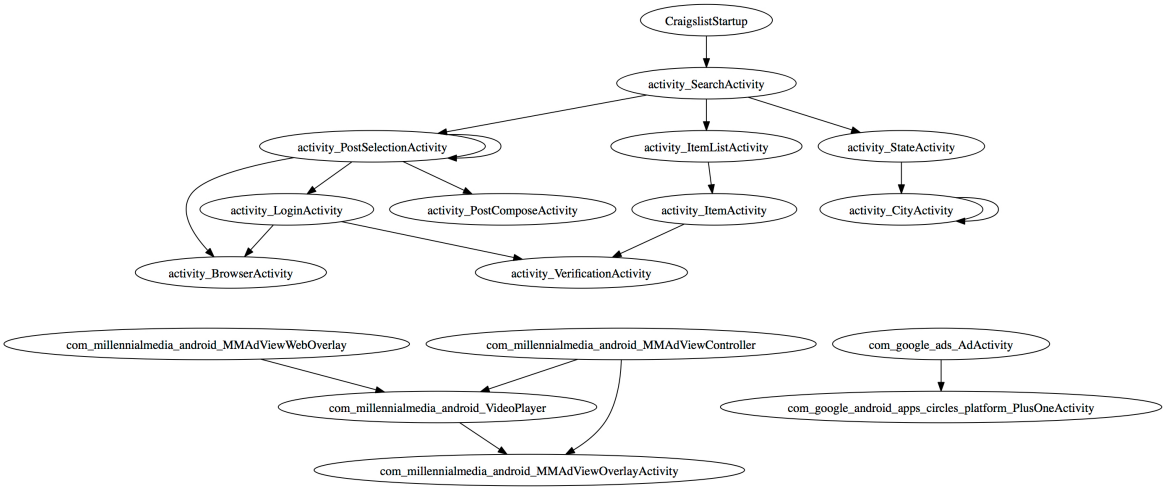

Figura 3.15: Esempio di SATG per l'App *craiglist* presente nel *Market di Google*

studio, è stato condotto dagli sviluppatori di A^3E sfruttando una versione personalizzata di *SCanDroid*. In figura 3.15 è possibile vedere l'SATG per l'App *Craiglist*.
A^3E, dopo aver ottenuto l'SATG, può iniziare ad *esplorare* le *Activity* con la stessa tecnica usata per il DFE, alla ricerca, in questa fase, delle informazione di transizione. Questo permette di ottenere dei vantaggi aggiuntivi:

1. È possibile invocare anche *Activity esportabili* e seguirne il percorso dall'SATG, così da coprire più *Activity*.

2. Esplorazione più rapida. Avendo a disposizione le informazioni di transizione, si necessita di minor tempo, non dovendo obbligatoriamente attendere per coprire tutte le *Activity*.

Il compromesso richiesto da questa strategia è quello di avere dovuto rinunciare ad un tipo di copertura che potesse essere:

- Copertura funzionale
- Copertura di metodi
- Copertura di istruzioni

Con queste rinunce si otterrà, sicuramente, la mancata copertura totale del codice sorgente da parte dei test.

In figura 3.16, è mostrato un flusso di esplorazione standard di A^3E.

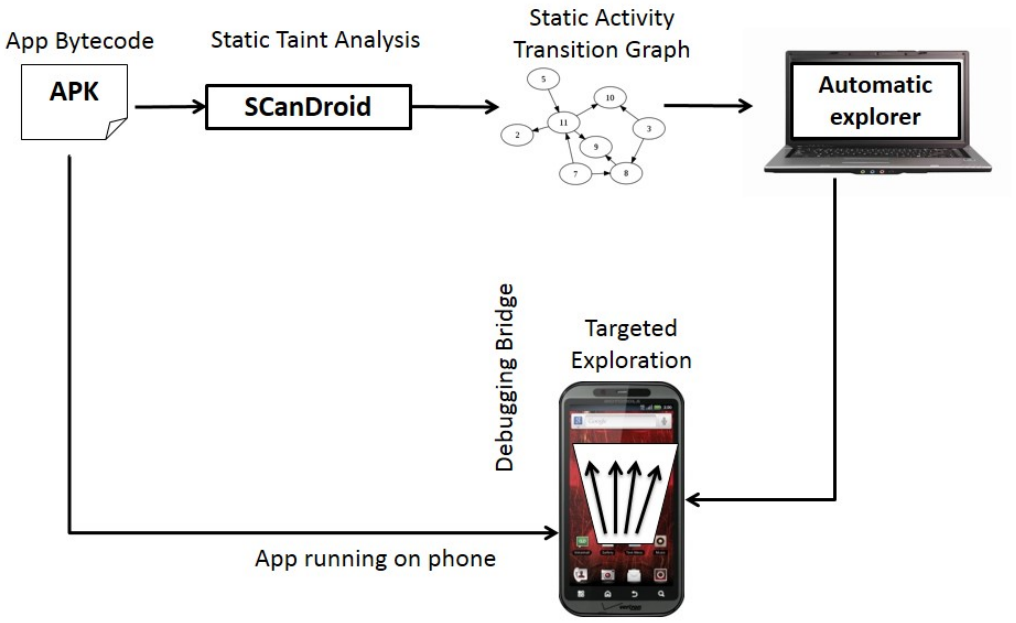

Figura 3.16: Flusso di esecuzione di una esplorazione standard

3.8.2 Targeted Exploration

TE raccoglie una prima serie di informazioni dalle *Activities*, dividendo le stesse in due categorie:

1. *Activities esportate*: Activities con *Intent filter* che possono essere lanciate dall'interno o dall'esterno dell'applicazione; alcune App trasferiscono il controllo ad App esterne, un semplice esempio potrebbe essere quello di un'App che ha la necessità di inviare e-mail. Lo sviluppatore, però, non può essere certo che l'utlizzatore dell'App invii mail sfruttando il client di posta predefinito, ed ancora, non è detto che l'*App* usata dall'utente per inviare mail abbia un flusso standard di esecuzione (ovvero, che emuli

il comportamento del gestore predefinito di posta). È infatti possibile, ad esempio, che questa App necessiti di una prima fase di accesso in cui richieda la validazione delle credenziali ad un servizio esterno (es. Facebook, Twitter, ...). È quindi chiaro che debba esserci un *Entrypoint* (o più) per richiamare l'*Activity* che sfrutta tali meccanismi.

2. *Activities interne*: *Activities* che possono essere richiamate solo attraversando un flusso standard di utilizzo.

Per analizzare le anomalie che potrebbero essere presenti nelle *Activities esportabili* A^3E costruisce un SATG da diversi *entrypoints* attraverso un'attenta analisi statica della App prima dell'esecuzione vera e propria della stessa.

3.8.3 Depth-First Exploration

Questa modalità, predefinita in A^3E, tenta di imitare il modo in cui un utente comune approccia ed interagisce con un sistema. Questo, risulta anche essere comportamento di interazione *imposto da Android*, ovvero, quello di passare di schermata in schermata e di costruire uno *Stack* con le *Activity* visualizzate. Così, ogni qualvolta un utente ritorna alla schermata precedentemente visitata, l'ultima schermata inserita viene eliminata.
La figura 3.17 mostra il comportamento della DFE.

3.8.4 Esempio di utilizzo

Come già mostrato nel precedente paragrafo, questo strumento permette di effettuare due tipi di esplorazioni

- DFE
- SATG

Depth-First Exploration

Per sfruttare questo tool di testing c'è bisogno di:

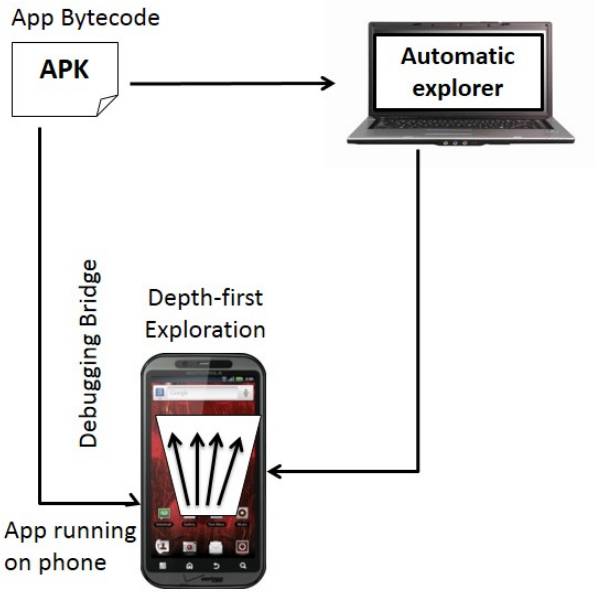

Figura 3.17: Flusso di esecuzione di una esplorazione standard

1. Installare *Ruby* sul sistema *Target*

2. Scaricare gli strumenti necessari dal sito: *https://github.com/tanzirul/a3e*

Una volta eseguite le due operazioni riportate sopra:

1. Spostarsi nel folder *troyd/bin* presente nel pacchetto scaricato

2. Lanciare il comando

```
ruby bin/rec.rb aut/your_application.apk
                    --no-rec -loop
```

Lo script eseguirà una serie di passi standard atti a:

1. Lancio *Activity* principale

2. Analisi dei componenti contenuti nell'*Activity*

3. Inserimento di input per la validazione dei campi editabbili

Gli input utilizzati negli *EditText* possono essere personalizzati; di default A^3E inserisce la stringa *test*, se si vuole modificare la stringa è possibile farlo editando il file *a3e.conf*. Se, invece, dovesse essere necessario utilizzare input specifici per uno o più *EditText*, è possibile creare un file xml con nome *activities.input.xml*, che contenga gli input con una sintassi del tipo:

```
<activity name="com.org.foo" viewid="4523651">
                        John</activity>
<activity name="com.org.foo" viewid="4523652">
                        john@example.com</activity>
```

A^3E tenterà di fornire questi input alle *Activity* indicate. Questo risulta, per esempio, utile nel fornire credenziali di accesso durante il test. The View Id di elementi può essere acquisita cercando di lanciare l'*Activity* seguita dal comando GetViews:

```
ruby bin/rec.rb aut/your_application.apk
                --no-rec -noloop com.org.foo getViews
```

3.8.5 Difficoltà di utilizzo

La principale difficoltà che si incontra utilizzando questo *tool* è quella di preparare l'ambiente di utilizzo, infatti, dato che sfrutta *Ruby*, bisogna installare tale strumento sull'ambiente *target*. Ci sono SO dove questo può essere più difficoltoso, prendiamo il caso di *OSX*: dato che è già presente *Ruby 1.8.7* e che tale versione non ha permesso l'esecuzione del test, è necessario effettuare il non facile aggiornamento di *Ruby* ad una versione più recente.

3.9 MonkeyTalk

MonkeyTalk è una piattaforma completa per la creazione di test funzionali per App:

- Native
- *WebApp*
- Ibride

MonkeyTalk in versione *Community Edition* è fornito con licenza *GNU Affero General Public License v3 (AGPL v3)*. *CloudMonkey LLC* (Software house che sviluppa il tool) sviluppa anche la *Professional Edition*, che offre un maggior numero di funzionalità rispetto alla *Community Edition*. Questa piattaforma è studiata e sviluppata con l'intento di supportare gli sviluppatori *Android* ed *iOS* nel testing delle proprie App, infatti, è necessario, per utilizzare questo strumento, effettuare alcune modifiche al progetto, che non dovranno essere presenti sull'*apk* da pubblicare sul market. Questo strumento è composto da tre parti fondamentali:

- *MonkeyTalk IDE*: applicazione desktop per la registrazione / riproduzione / creazione di script di test
- *MonkeyTalk Agent*: libreria che deve essere aggiunto app per abilitare il test
- *Script MonkeyTalk*: componente che permette la creazione di script di test mantenibili

Una volta inserito il *MonkeyTalk Agent* corretto (ne esistono due diversi, uno per *Android* ed un altro per *iOS*) all'interno dellApp è possibile sfruttare l'IDE per *registrare* una sessione di test. È anche possibile eseguire i task *MonkeyTalk* con *Ant*, gli sviluppatori, poi, suggeriscono l'uso di Hudson o Jenkins come *CI Server*.
MonkeyTalk Agent supporta testing di App *Android* a partire dalla versione 2.2 (Froyo).
Oltre alla *registrazione* di test con il supporto di *MonkeyTalk IDE*, *MonkeyTalk* offre la possibilità di creare test con l'ausilio di un proprio linguaggio di scripting. Gli script creati con questo linguaggio potranno poi, essere esportati in *Javascript* ed essere eseguiti anche in questo linguaggio.
Gli sviluppatori hanno voluto inserire questa funzionalità per introdurre costrutti di programmazione piÌ comuni in *MonkeyTalk*, come cicli, istruzioni di controllo generazione di numeri casuali...
Un esempio di un semplice script *MonkeyTalk* può far capire le potenzialità di questa scelta[24]:

```
Input username EnterText joe
Input password EnterText "my secret password"
Button LOGIN Tap
```

Questo semplice script inserisce delle credenziali all'interno di un form ed effettua il click sul bottone *LOGIN*. Dopo la conversione il precedente script diventa[24]:

```
app.input("username").enterText("joe");
app.input("password").enterText("my secret password");
app.button("LOGIN").tap();
```

Partendo dal precedente script è possibile elaborare script più complessi che possano eseguire test più accurati, un esempio è mostrato di seguito[25]:

```
if (typeof Test == "undefined") {
  load("libs/Test.js");
};

Test.Login.prototype.run = function(usr, pwd) {
  usr = usr || randStr();
  pwd = pwd || randStr();

  this.app.input("username").enterText(usr);
  this.app.input("password").enterText(pwd);
  this.app.button("LOGIN").tap();
};

function randStr()
{
  var text = "";
  var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
      + "abcdefghijklmnopqrstuvwxyz0123456789";

  for( var i=0; i < 5; i++ )
    text += possible.charAt(Math.floor(Math.random()
        * possible.length));

  return text;
}
```

Nel precedente esempio è possibile vedere come da un semplice esempio sia stato costruito un test più complesso con la sola conoscenza di *javascript*. Questo rende la velocità di apprendimento dello scripting di *MonkeyTalk* molto breve per tutti coloro che già conoscono *javascript*.

Altre *features* di questo tool sono:

- Possibilità di parametrizzazione degli Script
- Data-driving
- Verifica dei risultati attesi
- Screenshots

3.9.1 Esempio di utilizzo

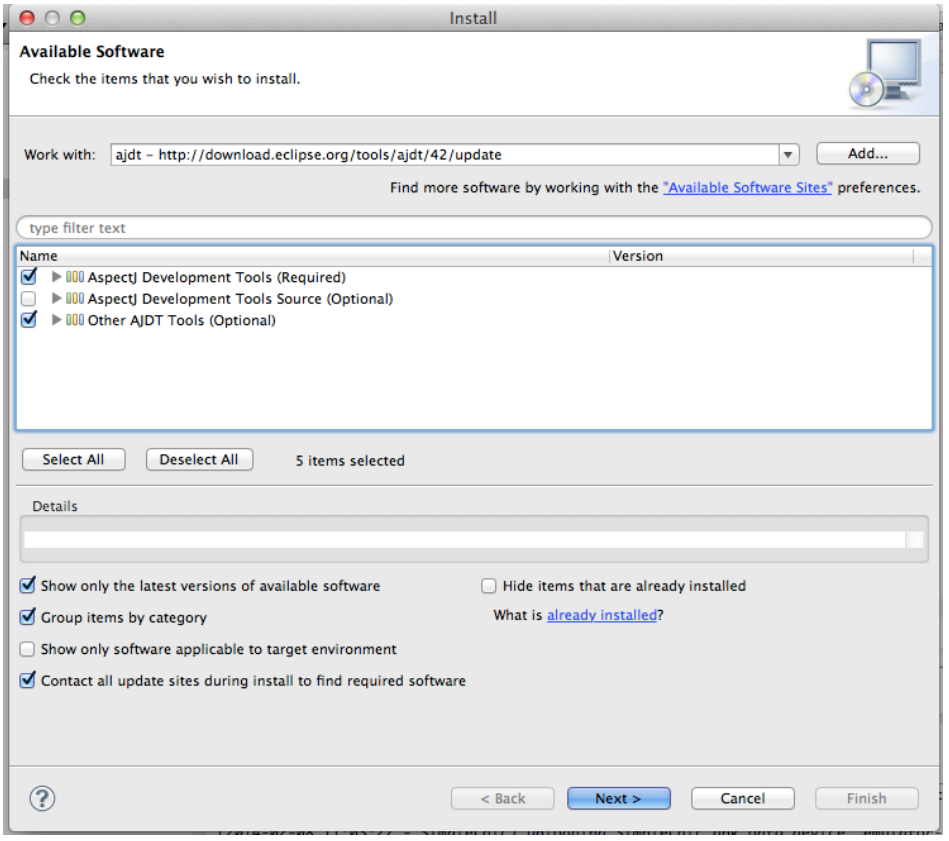

Figura 3.18: Installazione MonkeyTalk in Eclipse

Figura 3.19: Conversione da Progetto *Android* a *MonkeyTalk*

Per effettuare test con *MonkeyTalk* è necessario in primo luogo installare e configurare l'ambiente di testing. Questo richiede alcuni passi:

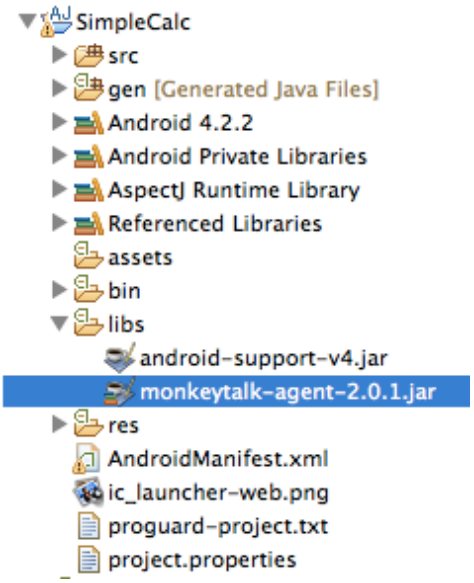

Figura 3.20: Inserimento librerie

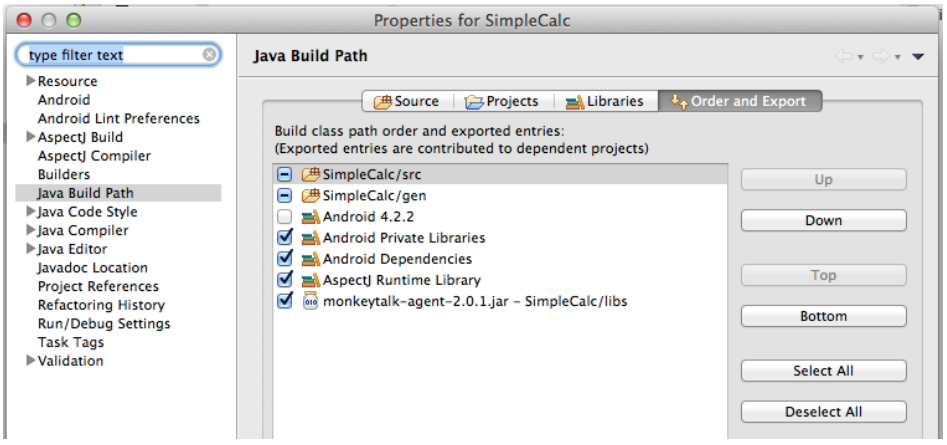

Figura 3.21: Inserimento nel buildpath ed export delle librerie

1. Scaricare l'ambiente MonkeyTalk (per i test presenti in questo testo è stato utilizzato l'ambiente offerto dalla *Community Edition*)

2. Installare in *Eclipse* l'*AJDT*, come mostrato in figura 3.18 a pagina 70

3. Convertire il progetto *Android* in *AspectJ Project*, come mostrato in figura 3.19 a pagina 70

4. Inserire le librerie di *MonkeyTalk* nel progetto *AspectJ Project*, come mostrato in figura 3.20 a pagina 71

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.mrcrack.android.simplecalc"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="14" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="it.mrcrack.android.simplecalc.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.GET_TASKS"/>
    <uses-permission android:name="android.permission.INTERNET"/>

</manifest>
```

Figura 3.22: Aggiunta permissions al *Manifest*

5. Abilitare l'export delle librerie necessarie in fase di creazione del file *apk* come mostrato in figura 3.21 a pagina 71

6. Aggiungere i permessi *GET_TASKS* e *INTERNET* al file *Manifest.xml* del progetto da testare, come mostrato in figura 3.22 a pagina 72

Finita la fase di configurazione del progetto è possibile, dopo aver inviato ed avviato la nuova versione dell'App, effettuare la registrazione del test:

1. Avviare l'*IDE* di *MonkeyTalk*

2. *Agganciare* l'*IDE* al device che esegue l'App da testare

3. Avviare la registrazione dei test dell'App

4. Eseguire sull'App tutte operazioni che si vogliono registrare per le future sessioni di test, come mostrato in figura 3.23 a pagina 73

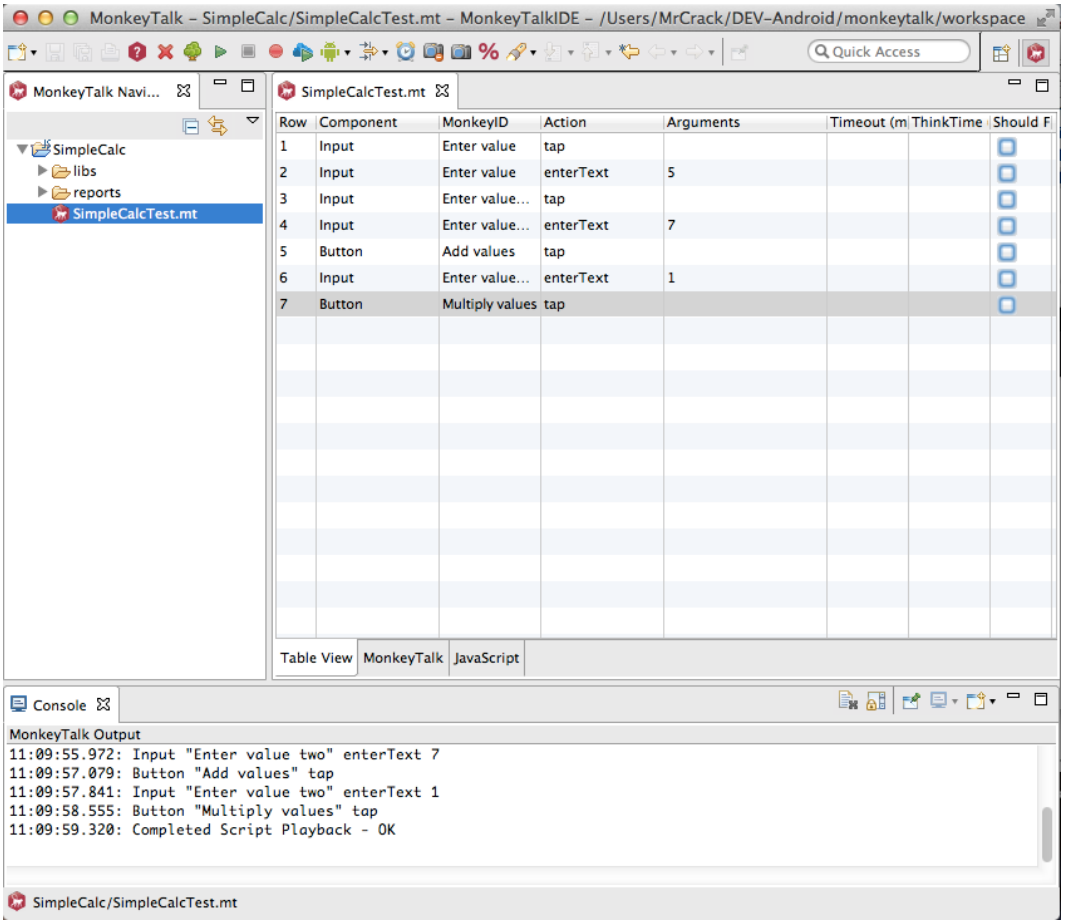

Figura 3.23: Registrazione test

L'*IDE* produce un elenco di step che corrispondono alle operazioni effettuate durante la registrazione (un esempio è mostrato in figura 3.24 a pagina 74). A partire dalla tabella è possibile vedere e modificare sia lo script *MonkeyTalk*:

```
Input "Enter value" tap
Input "Enter value" enterText 5
Input "Enter value two" tap
Input "Enter value two" enterText 7
Button "Add values" tap
Button "Multiply values" tap
Input "Enter value" tap
Input "Enter value" enterText a
Button "Add values" tap
Button Back tap \%thinktime=3000
Button "Multiply values" tap \%thinktime=3000
Button Back tap \%thinktime=3000
Input "Enter value" tap \%thinktime=3000
Input "Enter value" enterText 5
Input "Enter value two" tap
```

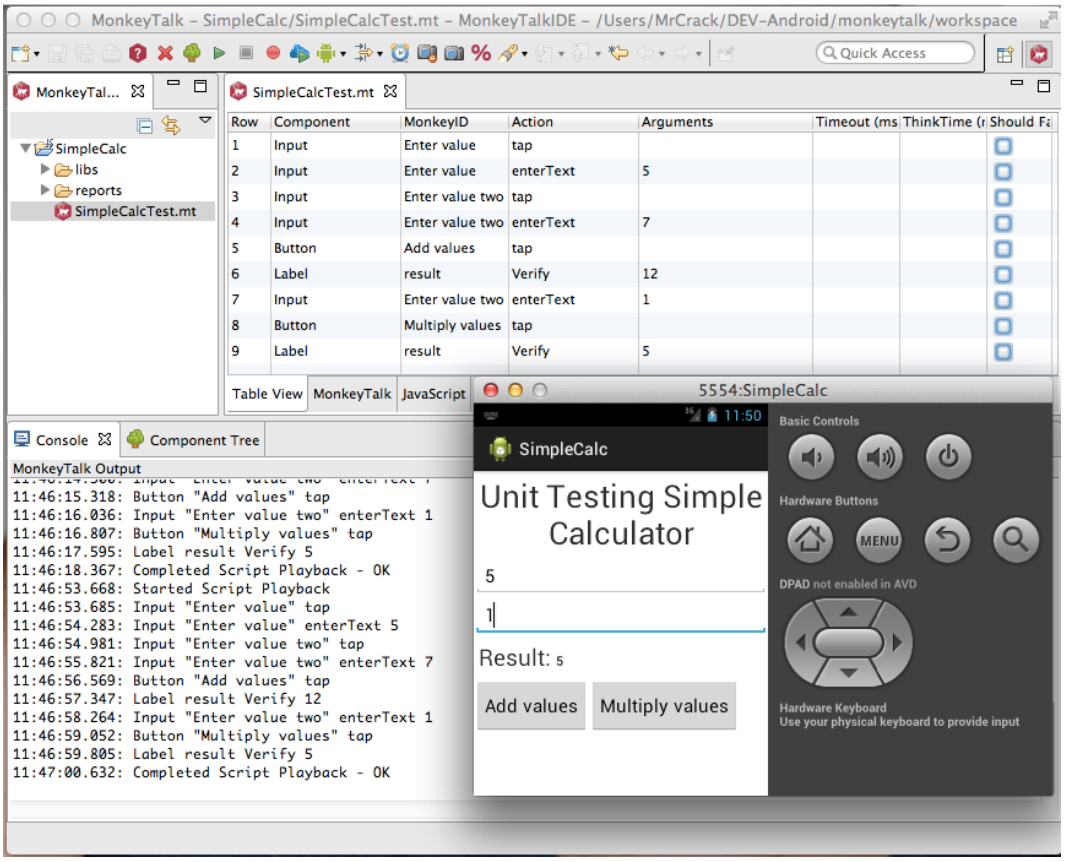

Figura 3.24: Aggiunta manuale delle verifiche ed esecuzione del test

```
Input "Enter value two" enterText -1
Button "Multiply values" tap
Button "Add values" tap
```

che lo script *Javascript*:

```
load("libs/SimpleCalc.js");
SimpleCalc.SimpleCalcTest.prototype.run = function() {
/**
 * @type MT.Application
 */
var app = this.app;
app.input("Enter value").tap();
app.input("Enter value").enterText("5");
app.input("Enter value two").tap();
app.input("Enter value two").enterText("7");
app.button("Add values").tap();
app.button("Multiply values").tap();
app.input("Enter value").tap();
app.input("Enter value").enterText("a");
app.button("Add values").tap();
```

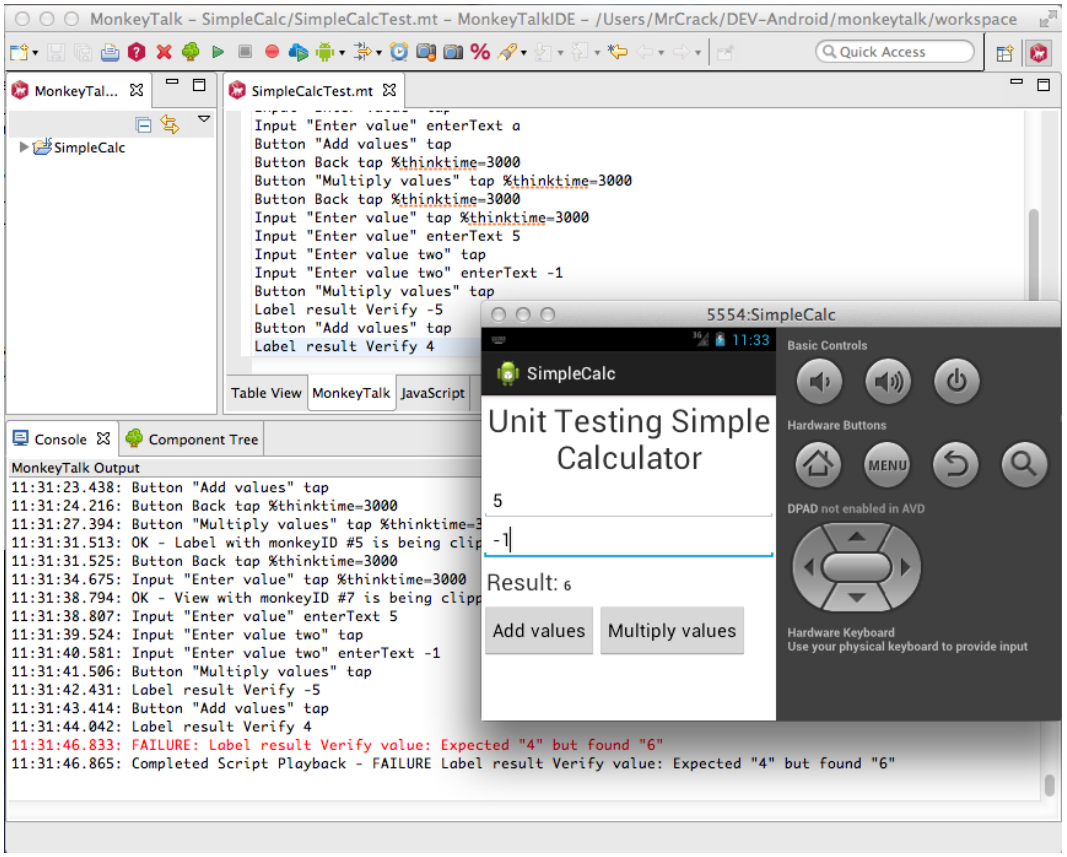

Figura 3.25: Aggiunta manuale delle verifiche ed esecuzione del test

```
app.button("Back").tap({thinktime:"3000"});
app.button("Multiply values").tap({thinktime:"3000"});
app.button("Back").tap({thinktime:"3000"});
app.input("Enter value").tap({thinktime:"3000"});
app.input("Enter value").enterText("5");
app.input("Enter value two").tap();
app.input("Enter value two").enterText("-1");
app.button("Multiply values").tap();
app.button("Add values").tap();
};
```

È anche possibile modificare gli script, aggiungendo, ad esempio, dei controlli:

```
Input "Enter value" tap
Input "Enter value" enterText 5
Input "Enter value two" tap
Input "Enter value two" enterText 7
Button "Add values" tap
Label result Verify 12
Button "Multiply values" tap
Label result Verify 35
Input "Enter value" tap
```

```
Input "Enter value" enterText a
Button "Add values" tap
Button Back tap \%thinktime=3000
Button "Multiply values" tap \%thinktime=3000
Button Back tap \%thinktime=3000
Input "Enter value" tap \%thinktime=3000
Input "Enter value" enterText 5
Input "Enter value two" tap
Input "Enter value two" enterText -1
Button "Multiply values" tap
Label result Verify -5
Button "Add values" tap
Label result Verify 4
```

```
load("libs/SimpleCalc.js");
SimpleCalc.SimpleCalcTest.prototype.run = function() {
/**
 * @type MT.Application
 */
var app = this.app;
app.input("Enter value").tap();
app.input("Enter value").enterText("5");
app.input("Enter value two").tap();
app.input("Enter value two").enterText("7");
app.button("Add values").tap();
app.label("result").verify("12");
app.button("Multiply values").tap();
app.label("result").verify("35");
app.input("Enter value").tap();
app.input("Enter value").enterText("a");
app.button("Add values").tap();
app.button("Back").tap({thinktime:"3000"});
app.button("Multiply values").tap({thinktime:"3000"});
app.button("Back").tap({thinktime:"3000"});
app.input("Enter value").tap({thinktime:"3000"});
app.input("Enter value").enterText("5");
app.input("Enter value two").tap();
app.input("Enter value two").enterText("-1");
app.button("Multiply values").tap();
app.label("result").verify("-5");
app.button("Add values").tap();
app.label("result").verify("4");
};
```

Nell'esempio in figura 3.25 a pagina 75, sfruttando l'anomalia presente nell'App di esempio viene mostrato il comportamento del tool in caso di errore.

Da quanto mostrato in questo paragrafo è abbastanza chiaro che l'utilizzo di questo strumento non genera troppi problemi. Durante i test, le principali difficoltà si sono avute per

- Recupero dei *MonkeyID*
- Potenza di calcolo del *Sistema target*
- Necessità di modificare il progetto

Infatti, il recupero dei *MonkeyID*, anche se viene messo a disposizione il *Component Tree*, non è molto semplice per i componenti che non si trovano sull'*Activity* principale.
Per quanto concerne, invece, il *Sistema target*, è necessario che questo abbia una discreta potenza di calcolo, in particolar modo, se si utilizzeranno *devices virtuali*, infatti, è necessario lanciare contemporaneamente:

- L'ambiente di sviluppo di *Android* (nel nostro caso *Eclipse*
- *Device virtuale* che faccia girare l'App
- L'ambiente di sviluppo di *MonkeyTalk* (versione custom di *Eclipse*

I test sono stati effettuati su un sistema con 4GB di Ram, che ha impiegato svariati minuti per avere un ambiente *Up&Running*.
La principale problematica, potrebbe essere, la necessità di dover effettuare modifiche al progetto:

- Conversione da progetto *Android* in progetto *AspectJ Project*
- Inclusione dalla libreria *monkeytalk-agent* nel *BuildPath* dell'App.

Dato che gli stessi sviluppatori sconsigliano vivamente di pubblicare l'App contenente tali librerie, si dovrà avere l'accortezza di evitare che la libreria venga utilizzata al di fuori delle fasi di test.

3.10 SeeTestMobile

La *Experitest Ltd* ha sviluppato una serie di strumenti per aiutare gli sviluppatori di App (*iOS*, *Android, Blackberry* e *WindowsPhone*) a *scrivere* casi di test. Gli strumenti sviluppati sono:

- SeeTest Automation
- SeeTest Manual
- SeeTest Cloud

tutti coperti da *Licenza Commerciale*, gli strumenti prodotti dalla *Experitest Ltd* funzionano con *Devices reali*, *Devices emulati* e con *Devices cloud*. Una delle principali peculiarità di questi strumenti è la possibilità data, di poter collegare simultaneamente più *Devices*, in modo da poter eseguire contemporaneamente gli stessi test su più tutti i dispositivi. È facile capire che questo permette di risparmiare una gran quantità di tempo.

3.10.1 SeeTest Automation

Questo *Tool*, disponibile solo a pagamento, permette di:

- *Registrare* sessioni di test
- Convertire la *registrazione* in uno script in UFT (QTP), MSTestVisualStudioTFS, RFT, TestComplete, C#, Java, Perl e Python
- Eseguire la sessione di test su uno o più *Devices* contemporaneamente

Gli script prodotti da *SeeTest Automation* possono essere eseguiti senza alcuna modifica su *Devices* di qualsiasi grandezza e tipologia (anche con diversi *SO*). Una volta prodotti gli script di test, sarà possibile correggerli e/o modificarli per un successivo utilizzo.

3.10.2 SeeTest Manual

Di tutti i componenti prodotti dalla *Experitest Ltd* questo è l'unico, che, anche se coperto da *Licenza Commerciale*, è messo a disposizione in versione gratuita. La principale peculiarità di questo strumento è quella di permettere l'esecuzione dei test con l'ausilio di mouse e tastiera e di registrare dei report che mostrano, per ogni singolo step una descrizione dell'operazione e lo *screenshot* del dispositivo, nell'istante in cui è stata effettuata l'operazione operazione. È inoltre disponibile (a pagamento) un plug-in, che permette l'esecuzione parallela di test, su diversi *Devices*. Quando si parla di *diversi Devices*, si intende parlare, non solo di dispositivi di diversa forma e grandezza, bensì, anche di dispositivi che adottano SO diversi, dato, che come già detto in precedenza, questi tools permettono di eseguire test non solo su dispositivi *Android*.

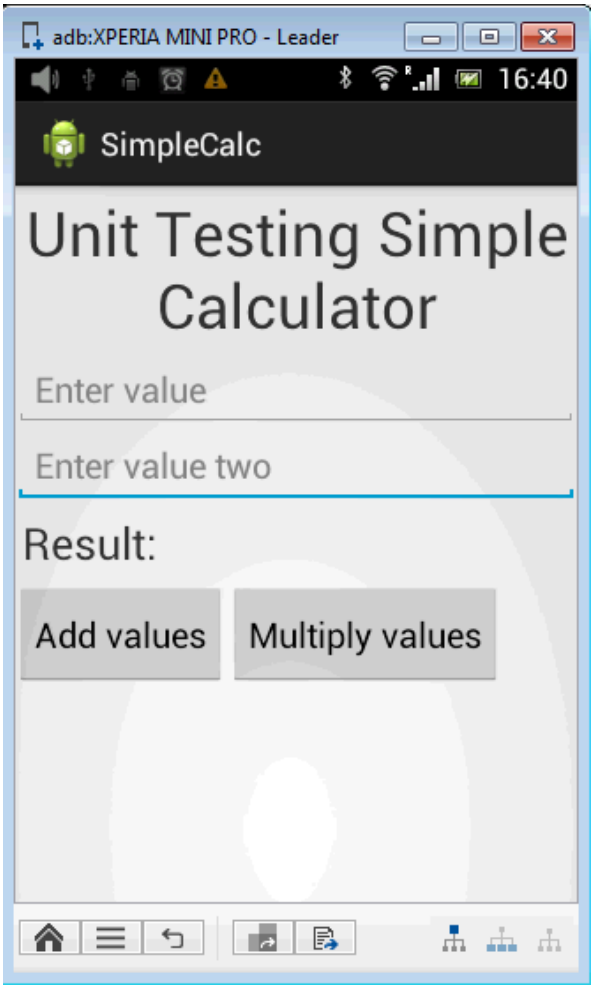

Figura 3.26: Smartphone collegato a SeeTest Manual

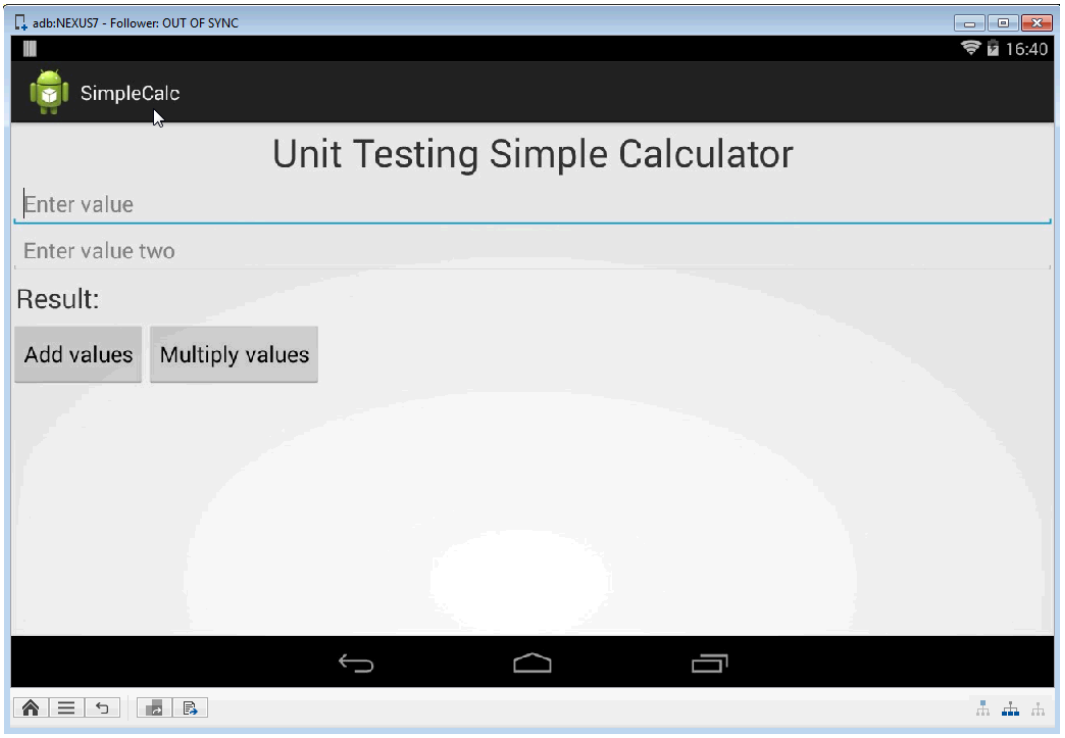

Figura 3.27: Tablet collegato a SeeTest Manual

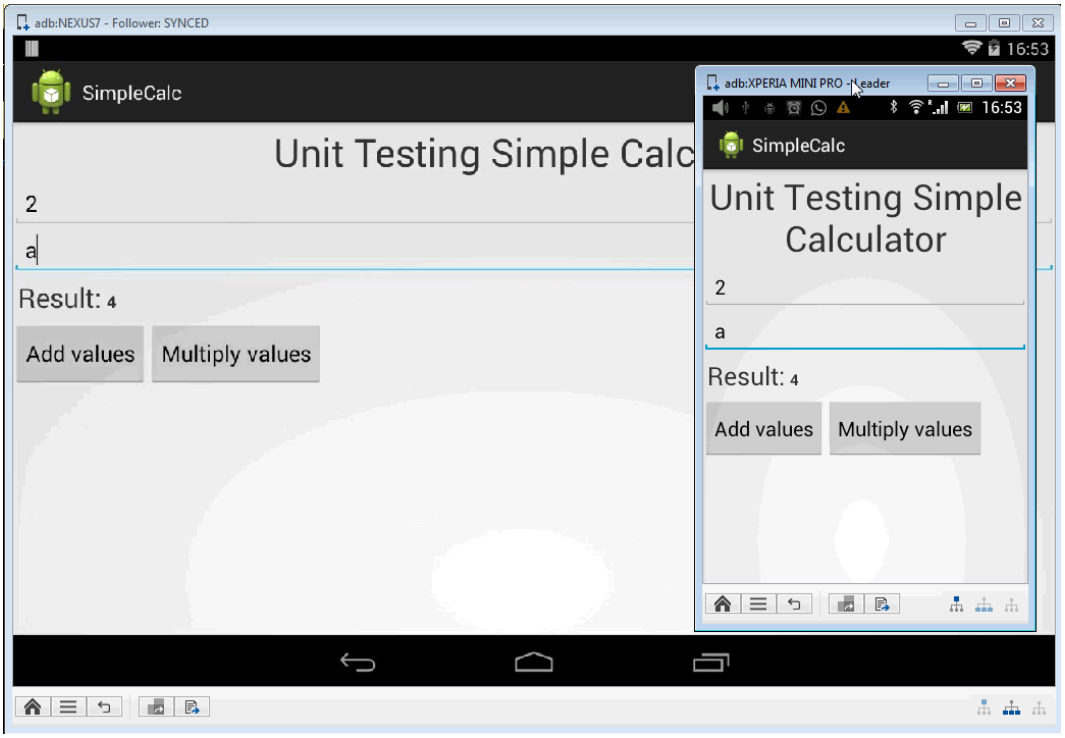

Figura 3.28: SeeTest Manual - Test che evidenzia la gestione degli input non validi

Come tutti gli altri strumenti della *Experitest Ltd*, anche questo strumento permette di collegare *Devices* multipli. Una volta collegati tutti i dispositivi al *Sistema target*, saranno

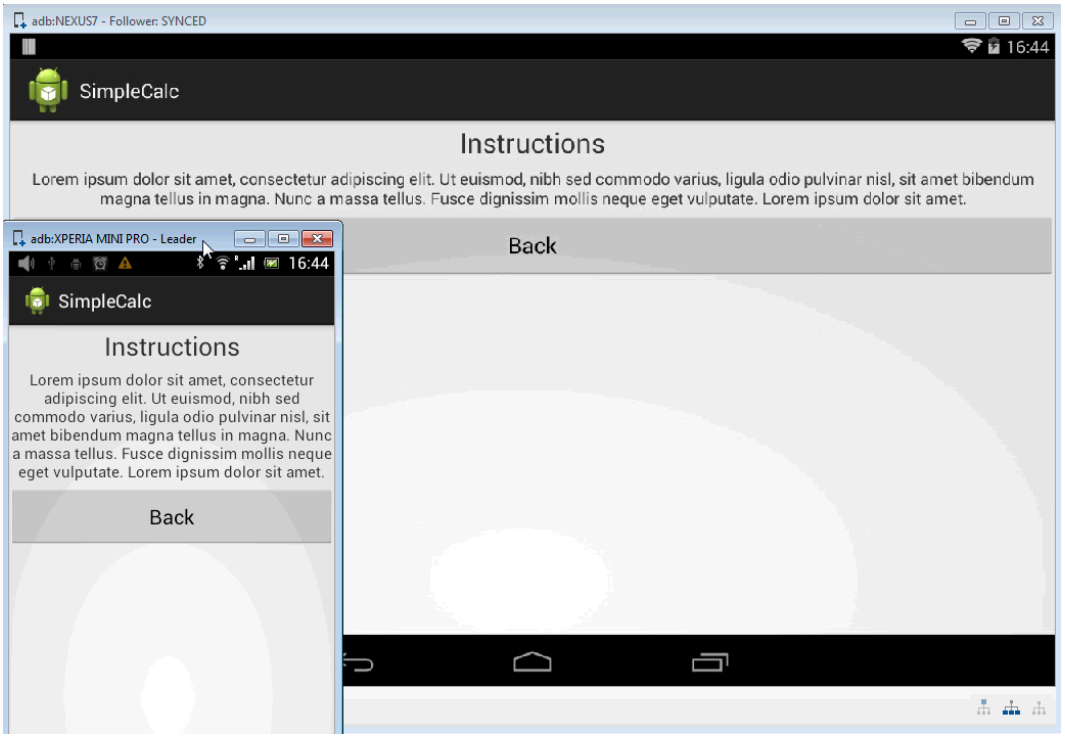

Figura 3.29: SeeTest Manual - Test che evidenzia la gestione degli input non validi

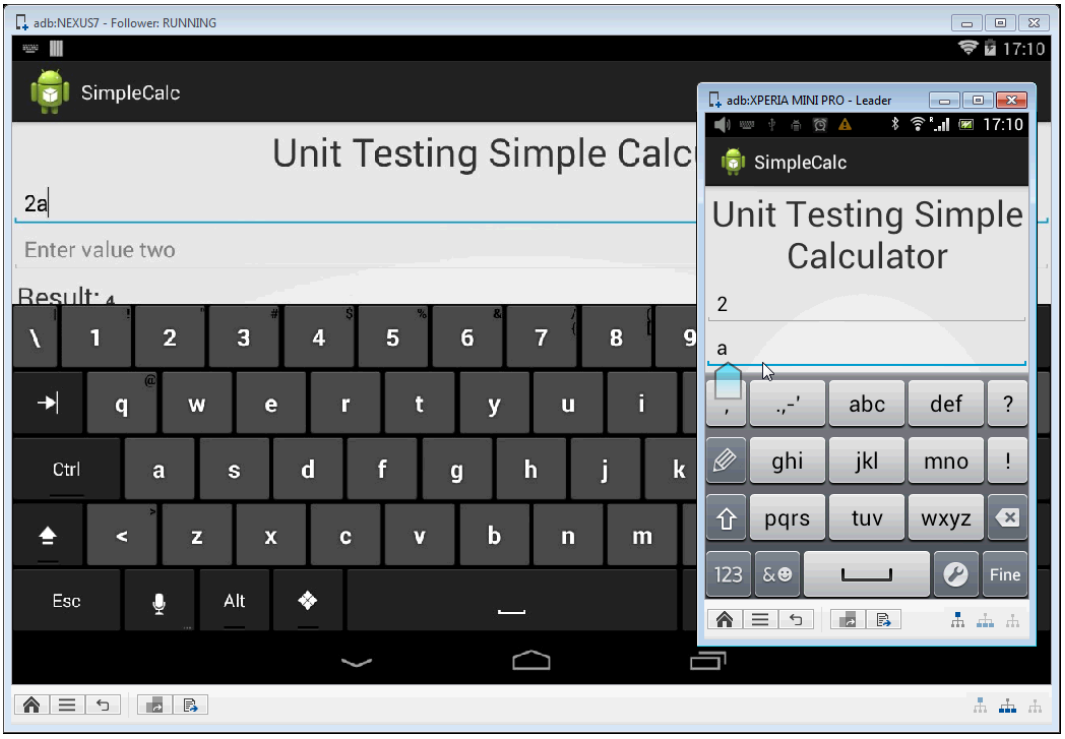

Figura 3.30: SeeTest Manual - Evidenza del disallineamento tra Devices avuto durante i test

visualizzate delle finestre che conterranno quanto mostrato dal *Device* originale e su cui sarà possibile operare con l'ausilio di mouse e tastiera. Ogni operazione effettuata sul *Main*

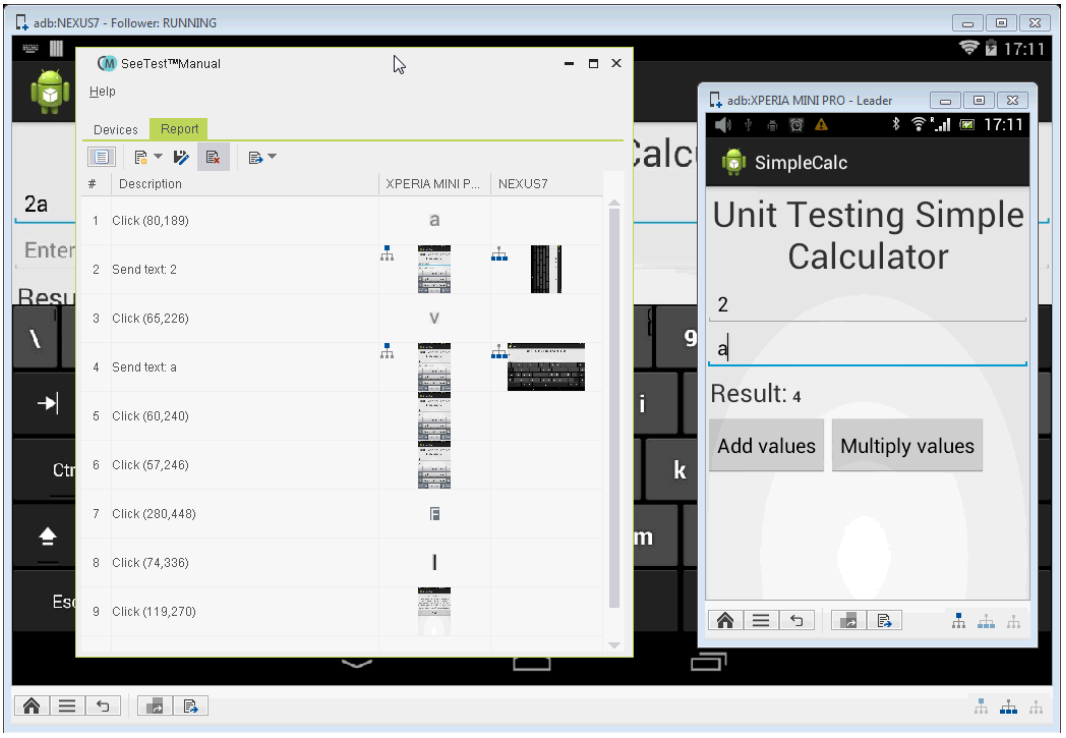

Figura 3.31: SeeTest Manual - Report del test

device, ovvero, sul dispositivo che si è deciso utilizzare quale dispositivo principale, sarà replicata su tutti gli altri dispositivi collegati al *Sistema target*. Non è possibile *registrare* la sessione di test per una nuova esecuzione, in quanto, questa funzionalità è presente su *SeeTest Automation*. È quindi lasciato all'addetto ai test ripetere nuovamente tutti gli step nelle successive sessioni di test e verificare che tutto funzioni correttamente.

3.10.3 SeeTest Cloud

Questo, più che essere un vero e proprio tool, è un servizio aggiuntivo che è possibile utilizzare con *SeeTest Automation* o *SeeTest Manual*. Questo servizio permette di *noleggiare Cloud devices*, ovvero, dei dispositivi reali fisicamente attaccati alle infrastrutture di *Experitest Ltd*. Questo servizio permette agli sviluppatori di testare le App sviluppate anche con Hardware non disponibile, sfruttando tutte le peculiarità che si hanno con i servizi cloud. Una volta acquistato il servizio, si dovranno *agganciare* i dispositivi cloud agli strumenti analizzati nei paragrafi precedenti, effettuata la connessione attraverso i parametri messi a disposizione, il funzionamento degli strumenti sarà lo stesso già analizzato.

3.10.4 Esempio di utilizzo

Da quanto detto già in precedenza, di tutti i tools sviluppati dalla *Experitest Ltd* l'unico utilizzabile gratuitamente è il *SeeTest Manual*, questo, per un periodo di tempo molto limitato (circa dieci minuti) permette di utilizzare anche il plug-in *Multiplier Add-on*. È così stato possibile effettuare dei test sull'App di esempio *SimpleCalc*. Effettuata l'installazione del tool e collegati due dispositivi molto diversi tra loro:

- Smartphone Sony Ericson mini pro
- Tablet Nexus7

si è proceduto con dei test analoghi a quelli visti già in precedenza. Dato che la verifica del corretto funzionamento dell'App è lasciata all'operatore, il test si è particolarmente concentrato sul funzionamento del *Multiplier Add-on*, che in un primo momento, come mostrato dall'immagini 3.29 a pagina , ha funzionato correttamente, poi, dopo alcuni giri di test ha mostrato alcuni problemi di disallineamento tra i *devices* (figura 3.30).

Di tutti i tools messi a disposizione dalla *Software Factory*, l'unico *FREE* è *SeeTest Manual*, questo però non permette l'automazione del testing delle App, bensì supporta il testing da PC e la parallelizzazione dei test su più *devices* contemporaneamente, anche se diversi tra loro. Sarà quindi compito dell'operatore sia effettuare tutti i casi di test, che verificare su ogni singolo *device* gli esiti ed il corretto avanzamento. Durante i test, il tool ha evidenziato alcuni problemi su quello che dovrebbe essere il suo punto di forza, ovvero, l'esecuzione parallela di test su piİ periferiche, infatti, dopo alcuni utilizzi, il *Device slave* non era più allineato con quanto fatto dal *device master*, generando quindi errori durante l'esecuzione.

3.11 Sikuli

SiculiX è un progetto *OpenSource* coperto da licenza *MIT*, è quindi liberamente utilizzabile sia per il testing di software sviluppati per uso privato, che per software commerciali.

Sikuli Script si basa su una libreria Jython e Java che automatizza l'interazione GUI usando i modelli di immagine per dirigere gli eventi di tastiera / mouse. Il nucleo di Sikuli Script

è una libreria Java che si compone di due parti: java.awt.Robot, che offre eventi da tastiera e mouse per la giusta *location* sul display, e un motore C++ basato su OpenCV, che cerca i *pattern* delle immagini sullo schermo. Il motore C++ è collegato a Java tramite JNI, deve essere quindi compilato per ogni piattaforma. Sulla libreria Java, uno *strato* Jython offre agli utenti finali un insieme di semplici comandi da utilizzare[31].
Uno script *Sikuli* (.sikuli) consiste in una directory che contiene

- un file sorgente Python (.py)
- tutti i file di immagine (.png) utilizzata dal file sorgente

Tutte le immagini utilizzate in uno script *Sikuli* sono dei semplici percorsi ai files png presenti nel pacchetto. Pertanto, il file sorgente *Python* può essere modificato con qualsiasi editor di testo.

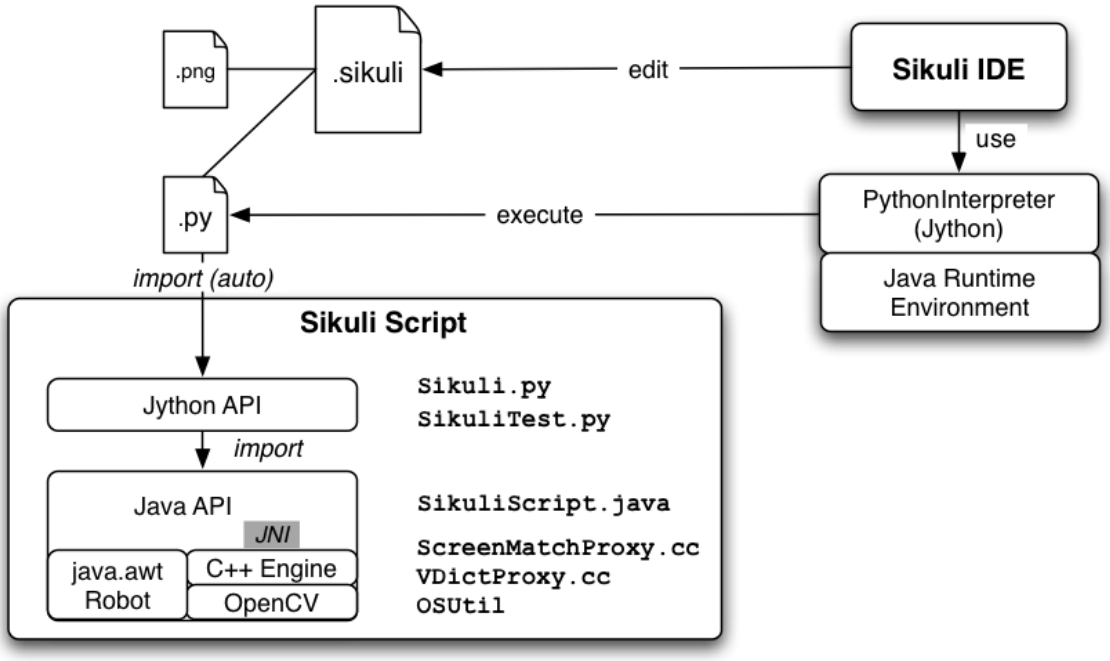

Figura 3.32: Sikuli System Design[31]

Durante il salvataggio di uno script, utilizzando Sikuli IDE, viene creato anche un file HTML aggiuntivo, questo permette agli utenti di condividere gli script sul web con facilità.

Uno script eseguibile *Sikuli* (.slk) è semplicemente un file compresso, contenente tutti i file contenuti nella directory *.sikuli*. Quando uno script viene passato a *Sikuli IDE* come argomento del comando (mostrato in figura 3.33 a pagina 85), *Sikuli IDE* riconosce il tipo del file passatogli analizzandone l'estensione

- Se l'estensione è ".skl", *Sikuli IDE* non visualizzerà la finestra IDE.
- Se l'estensione è un ".sikuli", *Sikuli IDE* aprirà in un editor per il codice sorgente.

Sikuli IDE aiuta a modificare e gestire gli script sorgenti *Sikuli*. Infatti, questo integra:

- La cattura dallo schermo
- Un editor di testo personalizzato *SikuliPane*

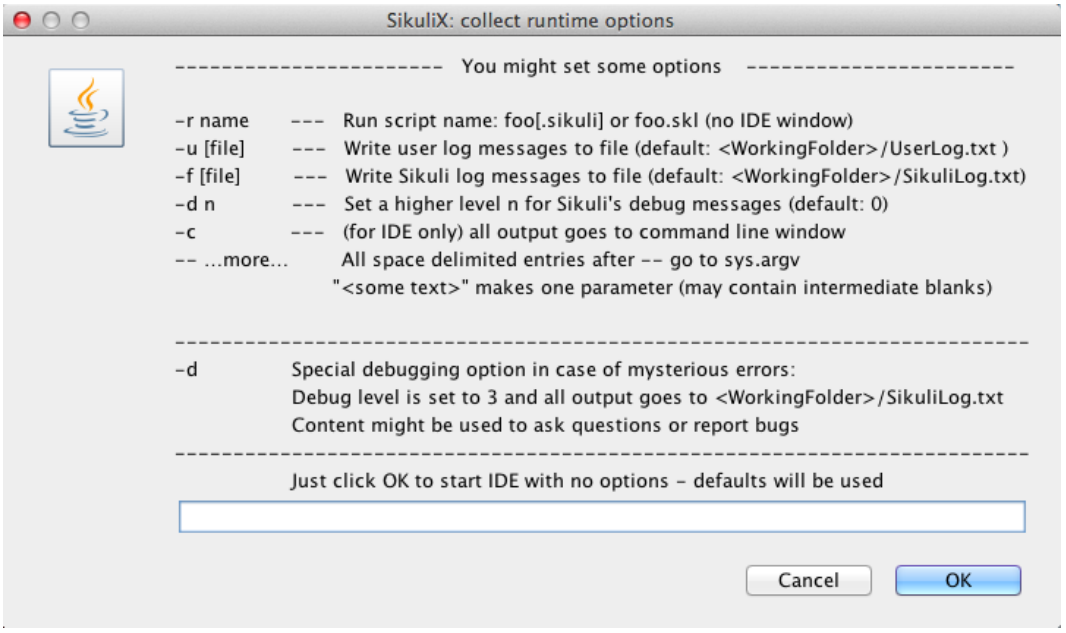

Figura 3.33: Sikuli - Richiesta parametri per lo start dell'IDE

così da ottimizzare l'usabilità della scrittura di script Sikuli. Per visualizzare le immagini incorporate nel *SikuliPane*, tutte le stringhe letterali che terminano con ".png" sono sostituite da un oggetto *JButton* personalizzato con un ImageButton. Se un utente regola la somiglianza del modello di immagine (grazie agli strumenti mostrati nelle immagini 3.34, 3.34 e 3.34), un *Pattern()* viene costruito automaticamente sulla parte superiore dell'immagine.

Per eseguire uno script *Sikuli*, *Sikuli IDE* crea un *org.python.util.PythonInterpreter* e passa alcune righe di intestazioni (ad esempio, per importare i moduli Jython di *Sikuli*, e per impostare il percorso di directory. Sikuli) all'interprete automaticamente. Una volta che queste intestazioni sono impostate, lo script ".py" viene semplicemente eseguito da *PythonInterpreter.execfile()*.

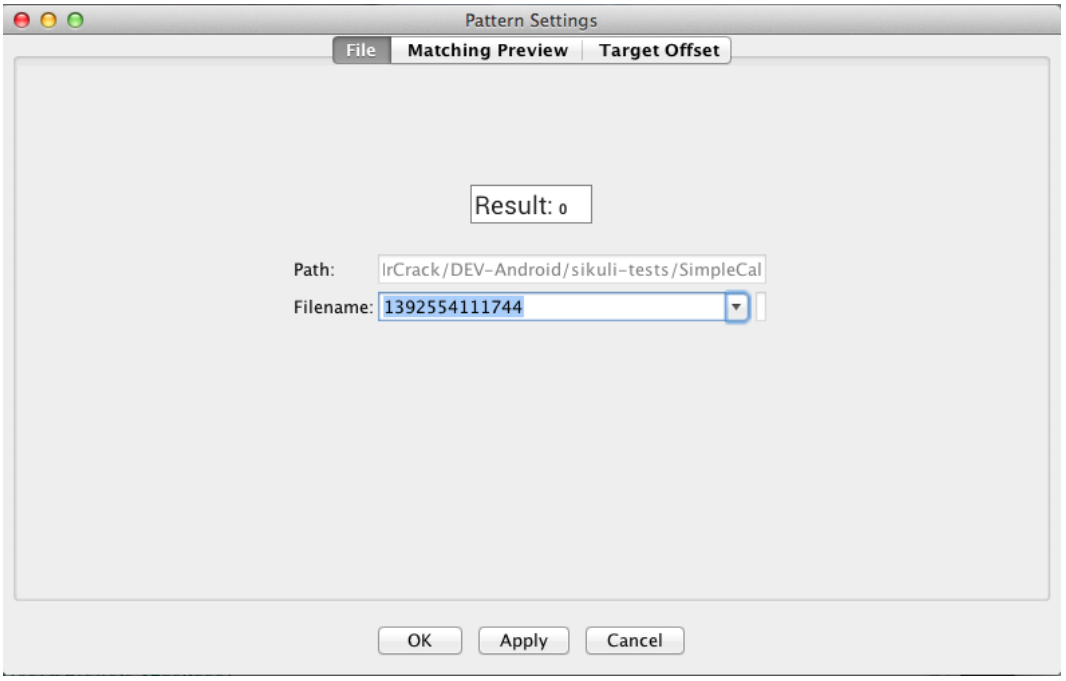

Figura 3.34: Sikuli - Regolazione del pattern *Tab File*

Da quanto appena detto, si evince, che *SikuliX* è uno strumento pensato e progettato per automatizzare tutto quello che viene mostrato sullo schermo dei Personal Computer (PC) su cui girano SO come *Windows*, *OSX* o *Linux/Unix*[32].

Poichè *Sikuli* non è pensato per testare direttamente App *Android*, è facile capire che, se l'addetto al testing volesse usare un *Device reale* per effettuare i test, dovrebbe installare sul dispositivo un *Server VNC*, che, per essere installato ha bisogno dei diritti di *root*.

3.11.1 Esempio di utilizzo

Il test è stato effettuato sfruttando l'*emulatore* messo a disposizione dall'ambiente di sviluppo di *Android*.

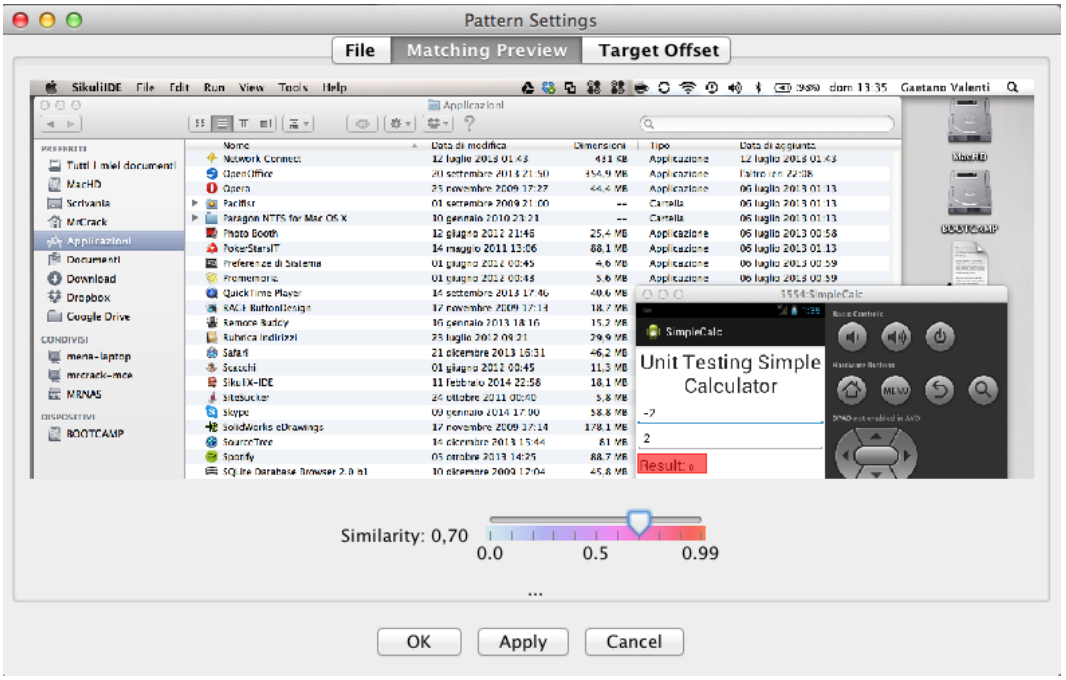

Figura 3.35: Sikuli - Regolazione del pattern *Tab Matching Preview*

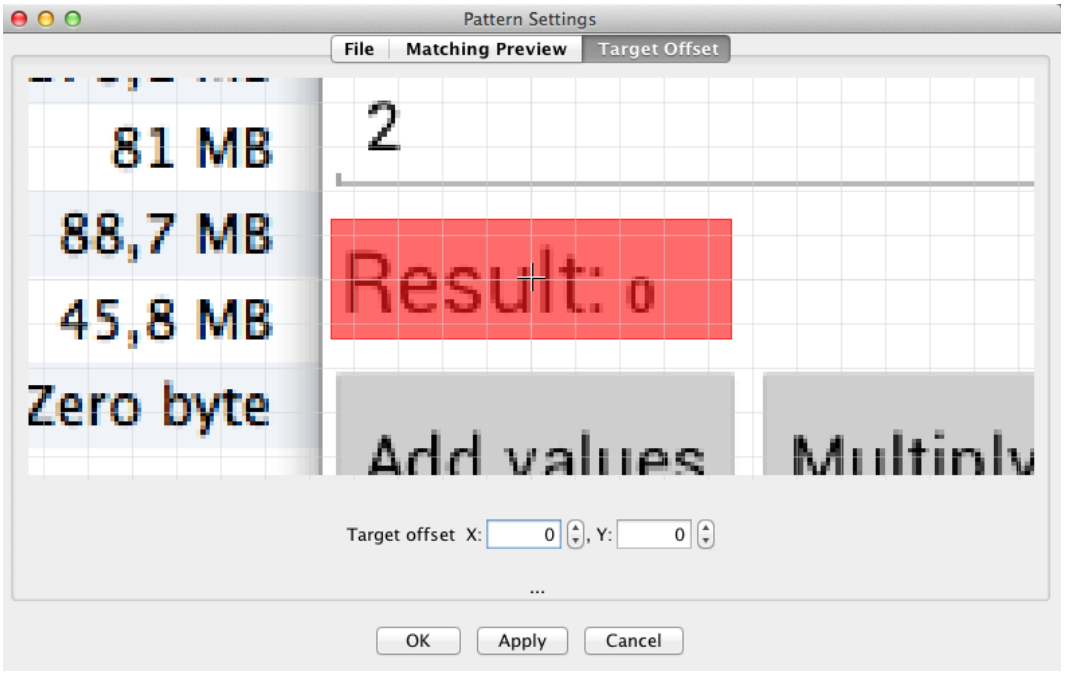

Figura 3.36: Sikuli - Regolazione del pattern *Tab Target Offset*

L'ambiente di test è installabile con pochi semplici passaggi. Installato quest'ultimo, è possibile avviare l'*IDE* e cominciare a sviluppare i test. I principali comandi da utilizzare negli script sono disponibili con su una barra degli strumenti (visibile nell'immagine 3.37 a pagina 88). Selezionato il comando da utilizzare, l'ambiente di sviluppo permetterà di selezionare

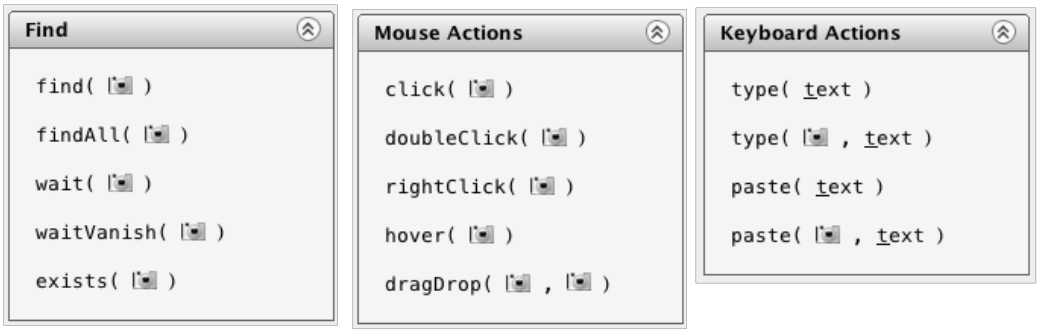

Figura 3.37: Sikuli - Strumenti dell'IDE

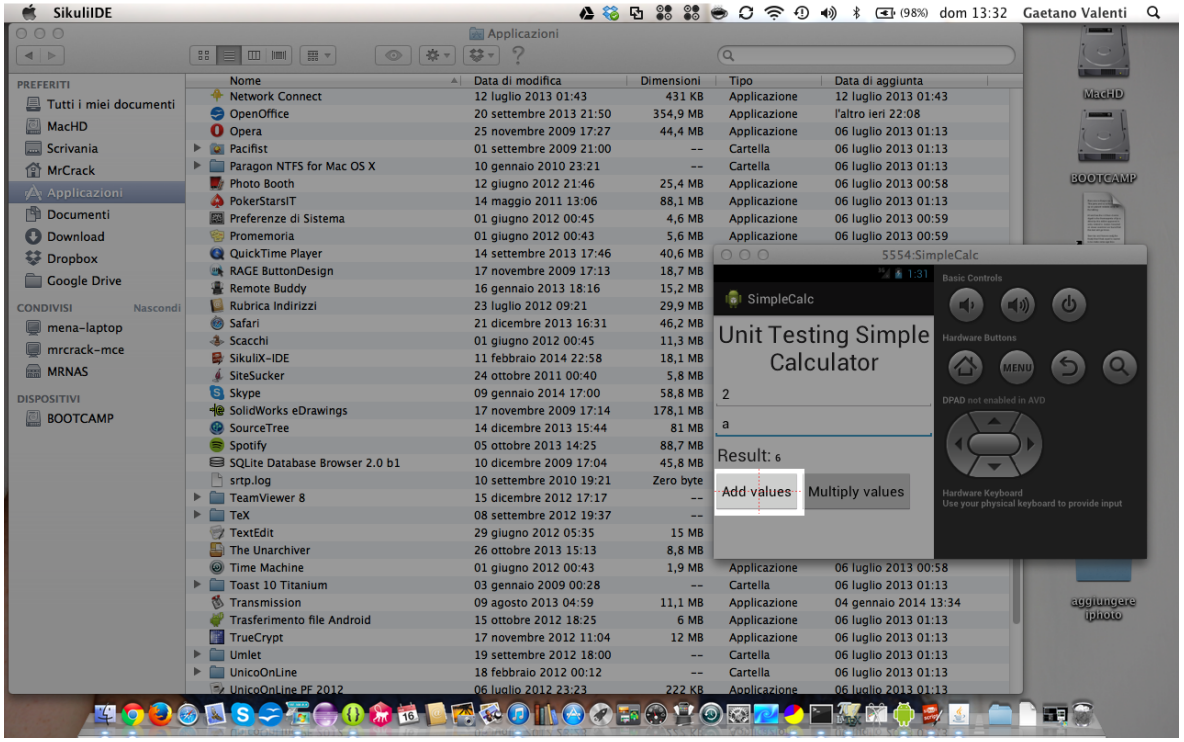

Figura 3.38: Sikuli - Selezione dell'area di interesse

l'area da cui prelevare il *pattern* per il confronto (come mostrato nell'immagine 3.38 a pagina 88) . Lo script visualizzato dall'*IDE* mostra anche le anteprime delle aree selezionate durante la stesura (come mostrato nell'immagine 3.39 a pagina 89), sicuramente molto importante, altrimenti sarebbe molto difficile mantenere il codice, infatti, lo script realmente memorizzato nel file ".py" è mostrato di seguito.

```
click("1392552655842.png")
type("2")
click("1392552701618.png")
type("3")
```

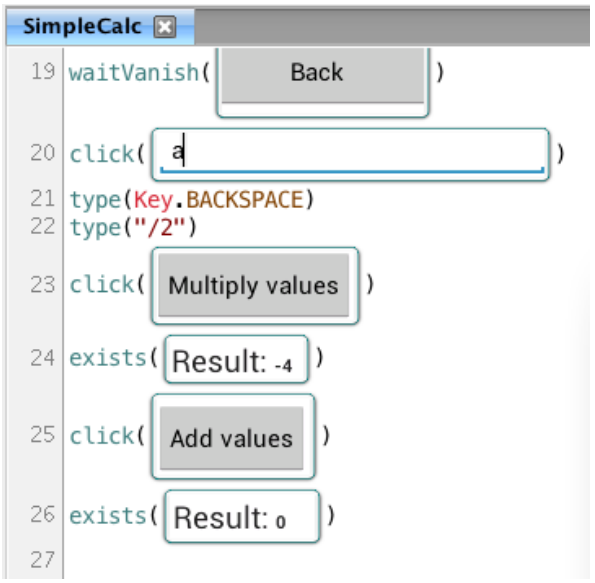

Figura 3.39: Sikuli - Script visto dall'IDE

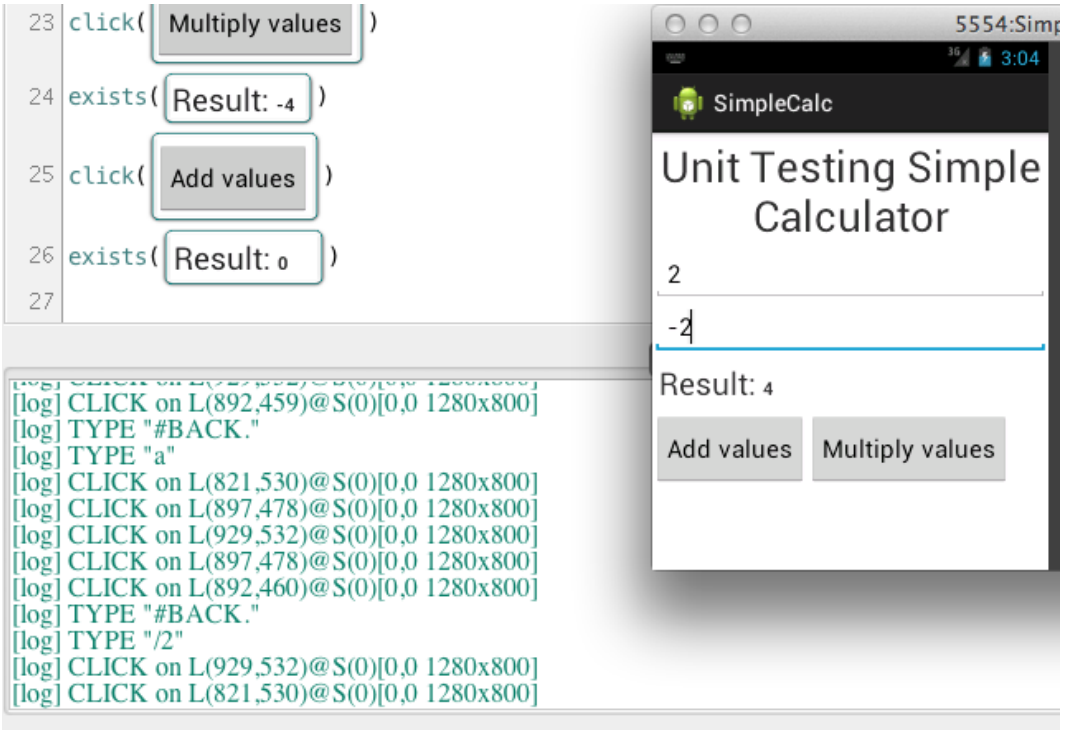

Figura 3.40: Sikuli - Errore di valutazione del risultato

```
click("1392552808914.png")
exists("1392552875227.png")
click("1392553281410.png")
exists("1392553314959.png")
click("1392553339824.png")
type(Key.BACKSPACE)
type("a")
```

```
click("1392552808914.png")
wait("1392553443821.png")
click("1392553456671.png")
waitVanish("1392553456671.png")
click("1392553281410.png")
wait("1392553443821.png")
click("1392553456671.png")
waitVanish("1392553456671.png")
click("1392554909372.png")
type(Key.BACKSPACE)
type("/2")
click("1392553281410.png")
exists("1392554056794.png")
click("1392552808914.png")
exists("1392554111744.png")
```

Per eseguire il test è sufficiente lanciarlo cliccando sul *Play* presente in cima all'*IDE*, oppure, se si volesse lanciarlo in un secondo momento, è possibile sfruttare la funzionalità già mostrata in precedenza (figura 3.33 a pagina 85). Durante i test si sono notati due limiti di questo strumento:

- Layout della tastiera

  Dato che durante l'esecuzione del test, viene simulata l'input da tastiera, ovvero, la reale pressione del tasto e dato che, il tool è stato sviluppato solo in base al layout *US*, si potrebbero avere problemi per i caratteri speciali. Ad esempio, il test sviluppato aveva la necessità di *digitare* il carattere "-", ma al suo posto veniva digitato l'apostrofo. Per *aggirare* il problema si possono usare due soluzioni:

  - Usare il comando *Paste* e non il *Type* (non utilizzabile in questa situazione)
  - Sostituire il carattere voluto con quello omologo sul layout US (nel caso specifico "/")

- Approssimazione eccessiva nella ricerca dei pattern

  Dato che i caratteri che rappresentano il risultato sull'*Activity* sono piccoli, il tool non riesce a distinguerli correttamente, infatti, nell'immagine 3.40 a pagina 89, è possibile vedere che dal test non è emerso l'errore introdotto nell'App.

Anche se *Sikuli* ha dimostrato notevoli qualità quali:

- comodità di utilizzo
- rapidità di scrittura dei test
- rapidità di apprendimento

Ha anche dimostrato notevoli limiti per l'utilizzo con *Android*, infatti, chi sviluppa App, ha la necessità di testarle su dispositivi diversi, ma dato che i test scritti con *Sikuli* potrebbero avere problemi riconducibili alla grandezza delle immagini, o dei testi visualizzati, è facile capire che questo potrebbe creare non pochi problemi con l'utilizzo di *Devices* di piccole dimensioni, ovvero, i cui display hanno una grandezza particolarmente ridotta. Rimane comunque un ottimo strumento per la scrittura dei primi casi di test, per esempio, da utilizzare durante lo sviluppo dell'App stessa, in modo da cercare di intercettare problemi di *regression*, problema che gli sviluppatori hanno spesso in fase di evoluzione del *Software*.

3.12 eggPlant

EggPlant consente di automatizzare l'esecuzione dei test funzionali in modo abbastanza rapido ed intuitivo, questo grazie ad un sistema brevettato per il *GUI testing*. L'interazione con l'utente è uniforme qualsiasi sia il dispositivo o l'applicazione sotto test, infatti, questo si basa sulle normali azioni effettuata dall'utente quando si trova avanti al PC. Questo tipo di approccio è fondamentale quando si parla di *curva di apprendimento*, infatti, risulta essere relativamente breve, mettendo in condizione chi lo utilizza di poter divenire produttivo in poche ore[34].

EggPlant utilizza una tecnologia di analisi avanzata delle immagini per *guidare* e validare i sistemi sotto test. Ad esempio, se si desidera fare clic sul pulsante OK, *EggPlant*, utilizzando algoritmi di riconoscimento delle immagini, analizza le immagini mostrate sullo schermo, in cerca del pulsante OK, al suo ritrovamento genera degli eventi a livello di sistema per cliccare su quel pulsante.

Questa tipologia di approccio, oltre ai vantaggi di cui si è parlato in precedenza, ha la caratteristica di riuscire ad astrarsi completamente dalla tecnologia usata per lo sviluppo del software sotto test, quindi l'utilizzo di questo tool, permette di testare applicazioni dal punto di vista dell'utente e non dal punto di vista del codice, siamo quindi in presenza di un testing di tipo *Black box*.

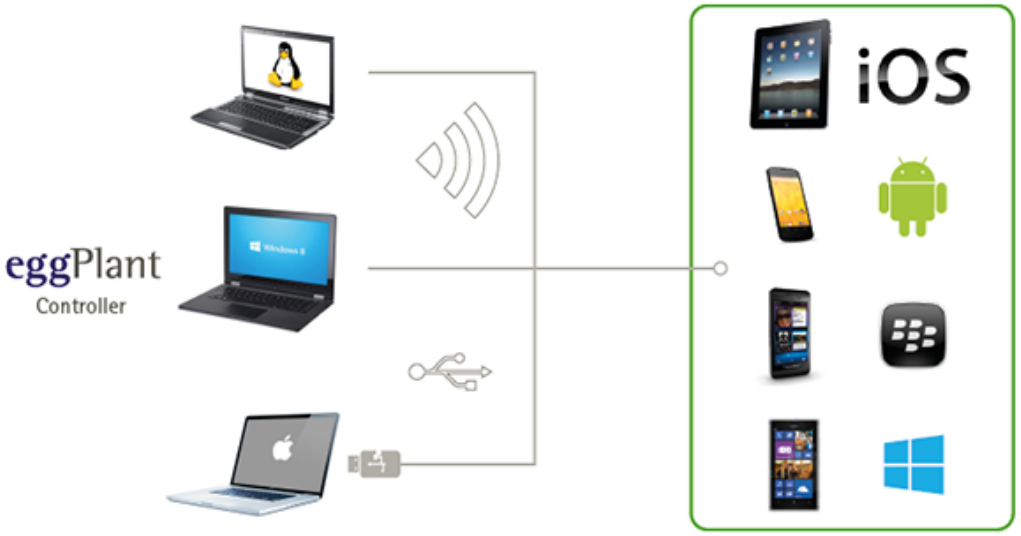

Figura 3.41: eggPlant - Interazione con devices esterni

Per l'utilizzo di questo strumento sui dispositivi mobili, vengono messe a disposizione delle App da installare sui *dispositivi fisici*, che permettono di visualizzare il display del *Device* sul monitor del PC e quindi, di testare le App, come un qualsiasi altro software.
Generalmente, le interfacce grafiche delle App, su dispositivi mobili diversi (anche con SO diversi), condividono una grafica abbastanza simile. Se l'App da testare è disponibile su più piattaforme e queste condividono l'interfaccia grafica, potrebbe essere possibile utilizzare gli stessi casi di test anche per *Devices* diversi, ottenendo quindi una netta riduzione dei tempi di sviluppo dei test.
eggPlant di base è un software Virtual Network Computing (VNC), ovvero, un software di controllo remoto che permette di amministrare il proprio computer a distanza (o nel nostro caso *Device*): installando un server VNC sulla propria macchina ed impostando una oppor-

tuna password si consente ai client VNC di ricevere una immagine dello schermo e di inviare input di tastiera e mouse al server VNC tramite una connessione remota.
Da quanto appena detto, si evince che:

1. L'ambiente di test è anche un client VNC
2. Il sistema da testare è un server VNC, quindi bisogna installarvi
    - Sui sistemi desktop un qualsiasi *Server VNC* già presenti sul mercato
    - Nei *Devices* una apposita App sviluppata dalla stessa Software House che ha sviluppato *eggPlant*

3.12.1 Esempio di utilizzo

Il test è stato effettuato sia sfruttando l'emulatore offerto dall'ambiente di sviluppo di *Android*, sia sfruttando un *Device hardware*. Per utilizzare l'emulatore non è possibile utilizzare l'App messa a disposizione dagli sviluppatori, questa richiede una connessione USB, non ottenibile in ambiente emulato. È stato quindi necessario installare un Server VNC sul PC su cui è installato l'emulatore; ulteriore vincolo a questa *architettura* è la necessità che il Server VNC e l'ambiente di test *eggPlant* non risiedano sulla stessa macchina, ma siano collegate tramite rete. In un secondo momento è stato comunque testato l'utilizzo di un *Device Hardware*, attività che ha evidenziato l'assoluta compatibilità dello script generato per l'emulatore con il *Device Hardware*.
L'installazione dell'ambiente, consiste in:

- Installazione sul PC dell'ambiente di test *eggPlant*
- Installazione di un Server VNC sul PC su cui risiede l'emulatore (macchina diversa da quella del punto precedente) oppure installazione dell'App su un *Device reale*

Funzionalità da testare

- Moltiplicazione numeri positivi

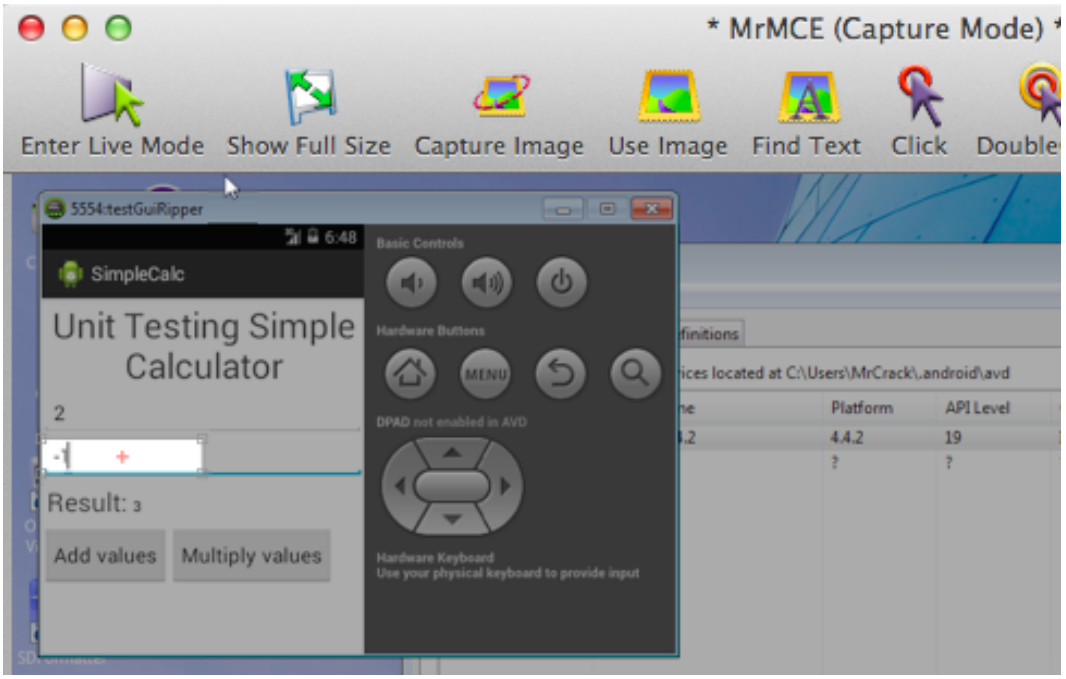

Figura 3.42: eggPlant - Selezione di un'area di interesse

- Precondizioni: L'App deve trovarsi sulla'*Activity* principale
- Input:

  Primo campo = 2

  Secondo campo = 4
- Output atteso: 8
- Output ottenuto: 8

- Somma numeri positivi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = 4
  - Output atteso: 6
  - Output ottenuto: 6

- Controllo Input
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = a
  - Output atteso: Apertura nuova *Activity* con istruzioni dell'App
  - Output ottenuto: Apertura nuova *Activity* con istruzioni dell'App
- Moltiplicazione numeri negativi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = -1
  - Output atteso: -2
  - Output ottenuto: 4
- Somma numeri negativi
  - Precondizioni: L'App deve trovarsi sulla'*Activity* principale
  - Input:

    Primo campo = 2

    Secondo campo = -1
  - Output atteso: 1
  - Output ottenuto: 1

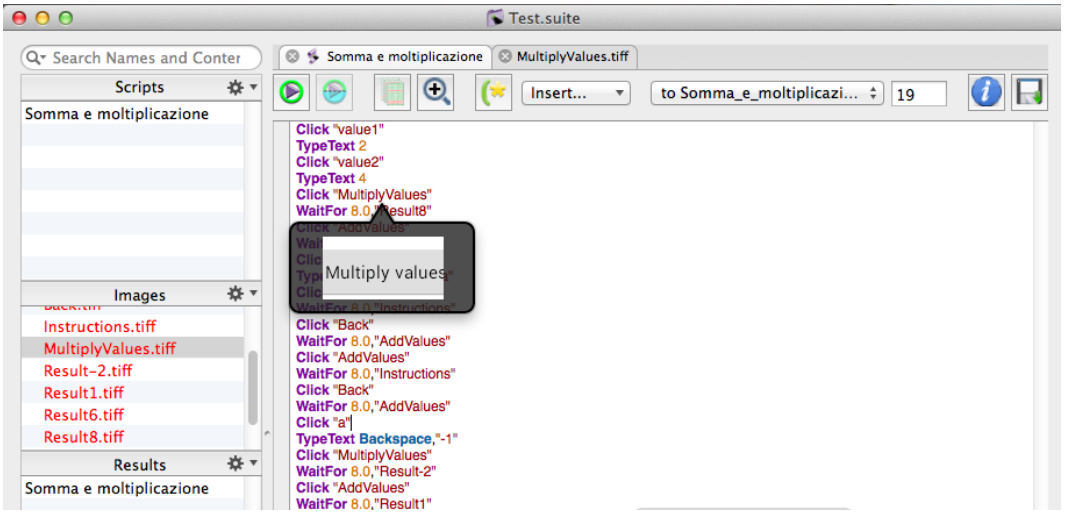

Figura 3.43: eggPlant - Interfaccia ambiente di sviluppo *eggPlant*

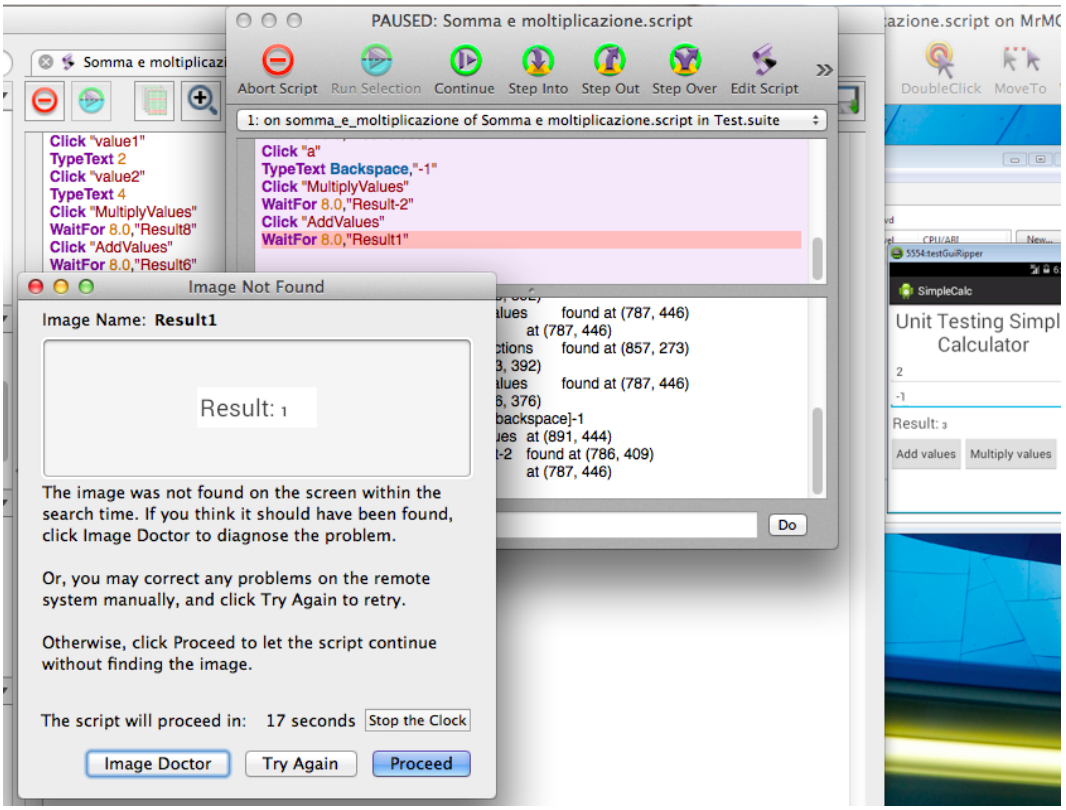

Figura 3.44: eggPlant - Errore durante l'esecuzione di un test

Test

Script prodotto dall'ambiente eggPlant

```
Click "value1"
TypeText 2
Click "value2"
TypeText 4
```

```
Click "MultiplyValues"
WaitFor 8.0,"Result8"
Click "AddValues"
WaitFor 8.0,"Result6"
Click "4.tiff"
TypeText Backspace,"a"
Click "MultiplyValues"
WaitFor 8.0,"Instructions"
Click "Back"
WaitFor 8.0,"AddValues"
Click "AddValues"
WaitFor 8.0,"Instructions"
Click "Back"
WaitFor 8.0,"AddValues"
Click "a"
TypeText Backspace,"-1"
Click "MultiplyValues"
WaitFor 8.0,"Result-2"
Click "AddValues"
WaitFor 8.0,"Result1"
```

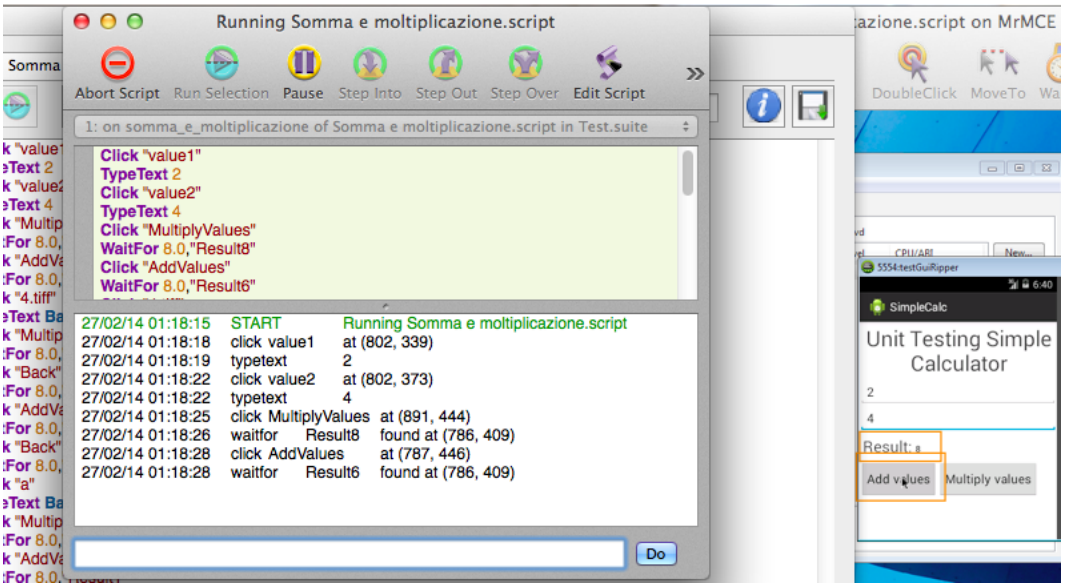

Figura 3.45: eggPlant - Esecuzione test con evidenza grafica dei controlli

Output

```
27/02/14 01:18:15 START Running Somma e moltiplicazione.script
27/02/14 01:18:18 click value1 at (802, 339)
27/02/14 01:18:19 typetext 2
27/02/14 01:18:22 click value2 at (802, 373)
27/02/14 01:18:22 typetext 4
```

```
27/02/14 01:18:25 click MultiplyValues at (891, 444)
27/02/14 01:18:26 waitfor Result8 found at (786, 409)
27/02/14 01:18:28 click AddValues at (787, 446)
27/02/14 01:18:28 waitfor Result6 found at (786, 409)
27/02/14 01:18:29 click 4.tiff at (796, 376)
27/02/14 01:18:30 typetext [backspace]a
27/02/14 01:18:33 click MultiplyValues at (891, 444)
27/02/14 01:18:42 waitfor Instructions found at (857, 272)
27/02/14 01:18:42 click Back at (853, 392)
27/02/14 01:18:47 waitfor AddValues found at (787, 446)
27/02/14 01:18:48 click AddValues at (787, 446)
27/02/14 01:18:56 waitfor Instructions found at (857, 273)
27/02/14 01:19:00 click Back at (853, 392)
27/02/14 01:19:06 waitfor AddValues found at (787, 446)
27/02/14 01:19:06 click a at (796, 376)
27/02/14 01:19:07 typetext [backspace]-1
27/02/14 01:19:08 click MultiplyValues at (891, 444)
27/02/14 01:19:09 waitfor Result-2 found at (786, 409)
27/02/14 01:19:09 click AddValues at (787, 446)
27/02/14 01:19:42 Exception Result1 Unable To Find Any Image
  On Screen "Result1" within 8.00 seconds
27/02/14 01:19:42 EndTestCase (Duration:"86.981874",
  Errors:"1", Exceptions:"1", StartTime:"2014-02-27 01:18:15
   +0100", Successes:"0",
   TestCase:"Somma e moltiplicazione.script", Warnings:"0")
27/02/14 01:19:43 FAILURE Image Not Found
  waitfor Error - Unable To Find Any Image On Screen
  "Result1" within 8.00 seconds\nExecution
  Time 0:01:26 Somma e moltiplicazione.script
```

3.13 Dollop

Il progetto *Dollop* coperto da licenza *Creative Commons Attribution-NoDerivs 3.0 Unported license*. Gli ideatori di questo tool hanno progettato ed implementato un'applicazione desktop che permette di creare, o per meglio dire "registrare", sessioni di test per App *Android*, grazie ad un'interfaccia totalmente grafica. Alcuni dei punti di forza di questo strumento sono:

- Non è richiesta la creazione di scripting in alcun linguaggio
- Non è richiesta la presenza dei sorgenti dell'App sotto test

- Supporta il test di:
  - App Native
  - App Ibride
  - WebApp
- Gestione di varie *Gesture*:
  - Tocchi brevi
  - Tocchi prolungati
  - Scroll
  - Testo
  - Immissione del codice.
- *Image-based*, supporto sia alle immagini che al testo
- Effettua *Screenshot* sia durante la riproduzione che durante la registrazione, dando quindi, maggiori informazioni in caso di errori durante il test.

Dollop è scritto totalmente in *Python* e sfrutta le librerie *wxPython* per la costruzione dell'interfaccia grafica. Il funzionamento di questo strumento si basa sulla traduzione delle operazioni effettuate durante la *registrazione* della sessione di test, in linguaggio *Python*, per una successiva ri-esecuzione del test. Questo tipo di test possono essere molto utili, sia a valle di modifiche effettuate all'App (aggiunta di nuove funzionalità, bug-fixing, ...), sia per studiare il comportamento dell'App su *Devices* diversi. Quanto detto ci porta a constatare che questo strumento è particolarmente adatto ad essere utilizzato da parte di *analisti funzionali*, che, non avendo un forte background tecnico, potrebbero avere difficoltà a scrivere script di test. TODO

3.13.1 Esempio di utilizzo

Prerequisiti

Per configurare l'ambiente di test (PC) sono necessari:

- Ambiente Windows (l'autore riporta che con alcune modifiche al codice potrebbe essere usato anche su Linux)
- wxPython
- OpenCV
- Configurazione delle variabili d'ambiente (*PATH* deve contenere il percorso contenete *adb*

Inoltre, questo tool funziona solo con *Devices* reali. Per il corretto funzionamento del *Device* bisogna:

- Abilitare l'*USB Debugging*
- Impostare il *Device* in *PC Mode*

Capitolo 4 Strumenti messi a confronto

In questo capitolo si vogliono confrontare i vari strumenti analizzati nel capitolo precedente per riassumerne i principali punti d'interesse. Il confronto conterrà:

- *Linguaggi supportati*: Contiene i linguaggi supportati dallo strumento di test. Se nella casella sarà presente *WebDriven* si vorrà intendere "tutti i linguaggi che offrono nativamente o che presentino librerie a supporto di tale protocollo"

- Supporto GUI:

  Questo campo informa se vi sono supporti grafici messi a disposizione dal tool, ad esempio: registrazione dei casi di test da GUI, scrittura di casi di test con il supporto del drug&drop, ...

- *Device di test supportati*: Contiene il tipo di *Device* supportato dallo strumento:

  - R: Reali
  - E: Emulati
  - C: Cloud

- *App supportate*: Contiene il tipo di App supportate:

  - N: Native
  - I: Ibride
  - W: WebApp

- *Continuous integration*: Indica se i test prodotti dallo strumento possono essere inseriti in un ambiente di *Continuous integration* quali *hudson*, *jenkins*, ...

- *Copertura del codice*: Indica se lo strumento fornisce la copertura del codice dei test creati.

- *Testing di Componenti Nativi*: Contiene informazioni sulla possibilità di creare test per componenti nativi di *Android*, quali: content provider, services, ...

- *OpenSource - Free*: Indica se lo strumento è OpenSource o Free, quindi se non vi sono costi da sostenere per il suo utilizzo.

	Linguaggi supportati	Supporto GUI	Device di test supportati	App supportate	Continuous integration	Copertura del codice	Testing di Componenti Nativi	OpenSource - Free
JUnit	Java	No	R - E	N	Si	No	Si	Si
Robotium	Java	No	R - E	N	Si	No	Si	Si
Calabash	Cucumber	No	R - E	N	Si	No	No	Si
Selenium Android WebDriver	WebDriver	No	R - E	W	Si	No	No	Si
Appium	WebDriver	No	R - E	N - I - W	Si	No	No	Si
Selendroid	WebDriver	No	R - E	N - I - W	Si	No	No	Si
Gui Ripper Android	Java	Si	E	N	No	Si	No	Si
A3E	-	Si	R - E	N	No	No	No	Si
MonkeyTalk	Proprietario	Si	R - E	N - I - W	Si	No	No	No
SeeTestMobile	-	Si	R - E - C	N - I - W	No	No	No	No
Sikuli	Proprietario	Si	E	N - I - W	No	No	No	Si
eggPlant	Proprietario	Si	R - E	N - I	No	No	No	No
Dollop	-	Si	R	N - I - W	No	No	No	Si

Conclusioni

Nel precedente capitolo sono stati analizzati numerosi strumenti che supportano il test di applicazioni *Android*, ognuno di questi offre una serie di funzionalità che possono essere più o meno utili a seconda dell'ambito progettuale. Sicuramente è difficile pensare che possa esistere un unico strumento di test che copra totalmente le complesse necessità di chi sviluppa App. È quindi opportuna una breve analisi delle necessità che si hanno nelle fasi principali della creazione di un'App:*Sviluppo*, *Integrazione* e *Collaudo*:

- *Sviluppo*: In questa fase, i test dovrebbero principalmente
  - Essere eseguiti in fase di *compilazione*
  - Essere dei *Test unitari*

  Perciò, non si ritiene rilevante se il tool offre supporti GUI per la scrittura di casi di test, ma sicuramente è importante fare considerazioni sui linguaggi supportati da tale tool, infatti, se tali linguaggi sono già conosciuti dagli sviluppatori, oppure, presentano una curva di apprendimento abbastanza rapida, il tool potrà rientrare tra quelli scelti per la data App, in questo frangente potrebbero essere particolarmente indicati i tools che offrono il supporto al *WebDriver*.

  Ulteriore punto da tenere sicuramente in considerazione è la *verbosità* del tool di test, prendendo come esempio JUnit, strumento offerto nativamente da *Android*, anche se molto potente richiede lunghi tempi di sviluppo, fattore fortemente penalizzante, dato che le attuali realtà progettuali non sempre possono permettersi tempi così lunghi. Nuovi strumenti presenti sul mercato risolvono in parte il problema, facilitando notevolmente l'attività degli sviluppatori, ma, al contempo, non offrono la stessa completezza offerta da *JUnit*. L'unico strumento che è riescito ad ovviare a questo problema sembra essere *Robotium*, questo grazie all'integrabilità offerta con *JUnit*.

Un altro punto di attenzione bisogna sicuramente farlo alla possibilità offerta da taluni tools di poter sfruttare emulatori per effettuare i test, infatti, sarebbe fortemente dispendioso fornire uno o più *Devices* ad ogni sviluppatore. Certamente i tempi utilizzando l'emulatore si allungano, ma utilizzando dei discreti PC, si può mitigare il problema.

- *Integrazione*: Fase fondamentale dei test, qui è possibile far emergere svariate tipologie di problemi: cattiva analisi dei componenti da sviluppare o delle loro interfacce, implementazione errata delle interfacce tra componenti, ...

  In questa fase di test, la presenza di supporti GUI potrebbe essere un fattore determinante, infatti, se il processo di sviluppo segue i principi del BDD in questa fase i test verrebbero "scritti a quattro mani", ovvero con il fondamentale supporto degli esperti del *Dominio del problema*.

  Di grande importanza dovrebbe essere ritenuta la possibilità di sfruttare la CI, infatti, adottare un sistema che effettua integrazione, buid e test periodicamente ed in modalità del tutto automatica permette di risparmiare una notevole quantità di tempo.

  Ulteriore fattore da tenere in considerazione è la possibilità di avere l'analisi della quantità di codice coperta dai test, infatti, tale informazione permette di prevenire eventuali problemi aumentando il numero di test case se non sufficienti.

  Infine, dato che il mercato stà portando a modelli di sviluppo distribuiti e che il mercato dei *Devices* è sempre in rapidissima evoluzione, potrebbe essere sentita l'esigenza di poter orientarsi verso strumenti che permettano l'utilizzo di *Device Cloud* (al momento, la maggior parte dei tools offre un servizio *HaaS*). Adottando una scelta di questo tipo, le *Software House* potrebbero:

  - Risparmiare sull'acquisto di numerosi *Devices*
  - Ottimizzare la gestione dei *Team distribuiti*

- *Collaudo*: Questa fase, generalmente svolta dal cliente, risente di parte delle esigenze già viste al punto precedente:

- Presenza di supporti GUI
- Utilizzo di *Device Cloud*

Ci sono anche delle esigenze di carattere più generale, che non sono dipendenti, quindi dalla fase di sviluppo in cui ci si trova, per esempio, la licenza offerta dallo strumento di test: per problemi di budget si potrebbe essere costretti a orientarsi verso soluzioni *Free*. Oppure, la tipologia di App supportata, infatti, è logico che se si fosse in presenza di una App *ibrida*, non sarebbe possibile orientarsi verso un prodotto che non la supporti.
Quindi, per decidere quali strumenti adottare all'interno di un progetto, è necessario prendere in considerazione tutte le necessità appena trattate e quelle più specifiche del progetto da sviluppare. Compito affidato principalmente ai progettisti.

Bibliografia

[1] IEEE Std. 610.12-1990. *Glossary of Software Engineering Terminology, in Software Engineering Standard Collection*. IEEE CS Press, Los Alamitos, California, 1990.

[2] A. K. Maji; K. Hao; S. Sultana; S. Bagchi. *Characterizing Failures in Mobile OSes: A Case Study with Android and Symbian*. IEEE 21st Int. Symposium on Software Reliability Engineering IEEE Comp. Soc. Press, pp. 249- 258., 2010.

[3] Dorota Huizinga; Adam Kolawa. *Automated Defect Prevention: Best Practices in Software Management*. John Wiley & Sons, 2007.

[4] Tao Xie; Kunal Taneja; Shreyas Kale; Darko Marinov. *Towards a Framework for Differential Unit Testing of Object-Oriented Programs*. Pubblicazione. Department of Computer Science, North Carolina State University USA; Department of Computer Science, University of Illinois at Urbana-Champaign USA, 2011.

[5] Diego Torres Milano. *Android Application Testing Guide*. Packt Publishing Ltd., 2011.

[6] Mark L. Murphy. *The Busy Coder's Guide to Advanced Android Development*. Commonsware, 2011.

[7] Domenico Amalfitano; Anna Rita Fasolino; Porfirio Tramontana; Bryan Robbins. *Testing Android Mobile Applications: Challenges, Strategies and Approaches*. Pubblicazione. Dipartimento di Informatica e Sistemistica, UniversitĹ di Napoli Federico II; Department of Computer Science, University of Maryland, 2011.

[8] Jerry Gao; Xiaoying Bai; Wei-Tek Tsai3; Tadahiro Uehara. *Mobile Application Testing - Research, Practice, Issues and Needs*. San Jose State University USA; Tsinghua University China; Arizona State University USA; Fujitsu Laboratories Ltd Japan, 2010.

Sitografia

[9] *A3E - about*. URL: http://spruce.cs.ucr.edu/a3e/about.html.

[10] *Activity lifecycle*. URL: http://marakana.com/bookshelf/main_building_blocks_tutorial/activity_lifecycle.html.

[11] *Android GUI Ripper - Presentazione*. URL: http://www.slideshare.net/PorfirioTramontana/using-gui-ripping-for-automated-testing-of-android-apps.

[12] *Android Service*. URL: http://funsung.blogspot.it/2012/05/android-service.html.

[13] *Android Testing Guide*. URL: http://developer.android.com/tools/testing/testing_android.html.

[14] *Android architecture*. URL: http://blog.zeustek.com/2010/11/11/android-architecture/.

[15] *Appium - Overview*. URL: http://www.slideshare.net/danielputerman/appium-overview-selenium-israel-2-feb-2014.

[16] *Calabash - Home*. URL: http://calaba.sh/.

[17] *Calabash - Overview*. URL: http://blog.lesspainful.com/2012/03/07/Calabash-Android/.

[18] *Contex Class*. URL: http://developer.android.com/reference/android/content/Context.html.

[19] *Instrumentation Class*. URL: http://developer.android.com/reference/android/app/Instrumentation.html.

[20] *IsolatedContext Class*. URL: http://developer.android.com/reference/android/test/IsolatedContext.html.

[21] *JUnit Testing Fundamentals*. URL: http://developer.android.com/tools/testing/testing_android.html.

[22] *Mock object*. URL: http://en.wikipedia.org/wiki/Mock_object.

[23] *MockContentResolver Class*. URL: http://developer.android.com/reference/android/test/mock/MockContentResolver.html.

[24] *MonkeyTalk - FAQ*. URL: https://www.gorillalogic.com/node/357.

[25] *MonkeyTalk - Scripting in JavaScript*. URL: https://www.cloudmonkeymobile.com/monkeytalk-documentation/monkeytalk-language-reference/scripting-javascript.

[26] *Palmare*. URL: http://it.wikipedia.org/wiki/Palmare.

[27] *Robotium homepage*. URL: https://code.google.com/p/robotium/.

[28] *Selendroid - Architecture*. URL: http://selendroid.io/architecture.html.

[29] *Selenium - Esempio*. URL: http://android-developers.blogspot.it/2011/10/introducing-android-webdriver.html.

[30] *Selenium - Overview*. URL: http://google-opensource.blogspot.it/2011/10/test-your-mobile-web-apps-with.html.

[31] *Sikuli Homepage*. URL: http://doc.sikuli.org/devs/system-design.html.

[32] *Sikuli Homepage*. URL: http://www.sikulix.com.

[33] *WebDriver - Specifiche*. URL: http://www.w3.org/TR/2013/WD-webdriver-20130117/.

[34] *eggPlant - Overview*. URL: http://www.testplant.com/eggplant.

Appendice A App "SimpleCalc"

A.1 Introduzione

La prima App di esempio non prevede l'interazione con l'hardware del device. Non vuole essere un esempio completo di calcolatrice, bensì, una semplice App, composta da due Activity, "capace" di effettuare la somma o la moltiplicazione di due operandi. La prima Activity ha il compito di gestire le operazioni, la seconda, invece, comparirà in caso di input errati all'interno dei campi di test. È stato inoltre introdotto un *Bug* nel codice, così da poter testare il comportamento degli strumenti analizzati in caso di errore. Il *Bug* è stato inserito nella somma, infatti, il valore inserito nel secondo campo viene preso in valore assoluto, quindi, ad esempio, la somma tra "1" e "-1", non darebbe come risultato "0", ma "2".

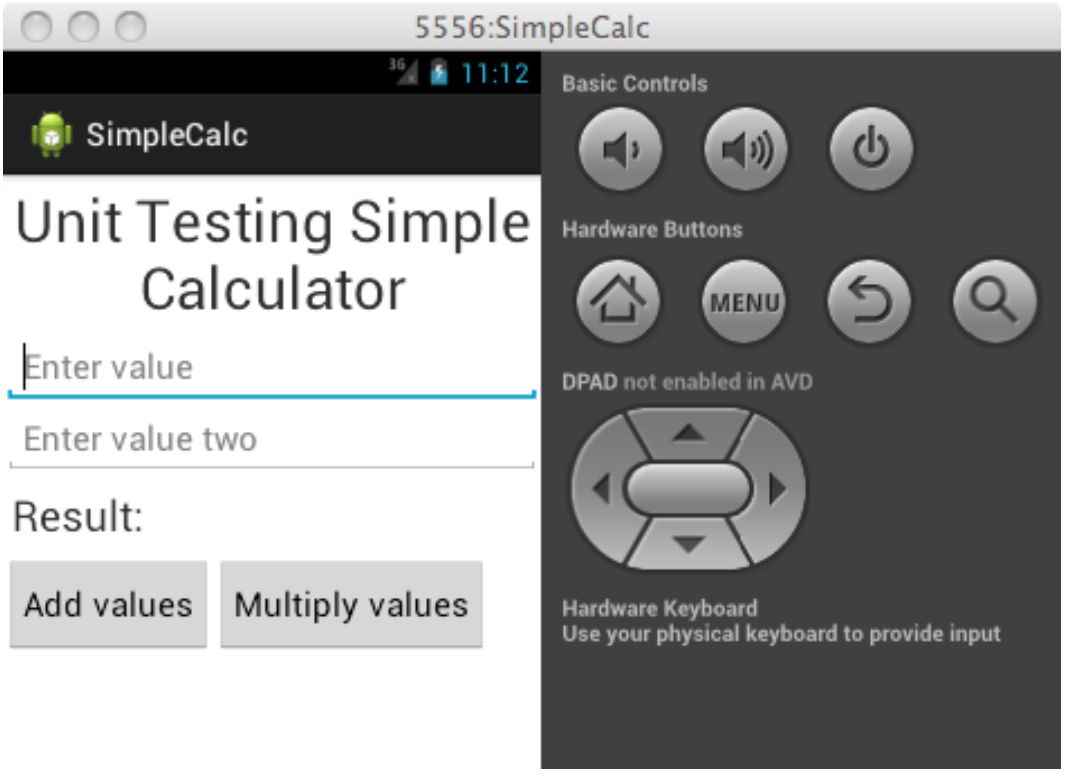

Figura A.1: SimpleCalc in Portrait

A.2 Sorgenti App

A.2.1 Manifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="it.mrcrack.android.simplecalc"
   android:versionCode="1"
   android:versionName="1.0" >

   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="14" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="it.mrcrack.android.simplecalc.MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
               />
         </intent-filter>
      </activity>
      <activity
         android:name="it.mrcrack.android.simplecalc.
            InstructionsActivity"
         android:label="@string/app_name" >
      </activity>

   </application>
   <uses-permission android:name="android.permission.GET_TASKS"/>
   <uses-permission android:name="android.permission.INTERNET"/>

</manifest>
```

A.2.2 mainactivity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
   android:id="@+id/MainLayout"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:gravity="center_horizontal"
```

```
   android:padding="3sp"
   android:text="@string/hello"
   android:textSize="35dp" />

<EditText
   android:id="@+id/value1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:hint="@string/hint1"
   android:inputType="text"
   android:textSize="20dp" >
</EditText>

<EditText
   android:id="@+id/value2"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:hint="@string/hint2"
   android:inputType="text"
   android:textSize="20dp" >
</EditText>

<FrameLayout
   android:id="@+id/FrameLayout01"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:padding="3sp" >

   <LinearLayout
      android:id="@+id/LinearLayout02"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:padding="2sp" >

      <TextView
         android:id="@+id/resultLabel"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/resultLabel"
         android:textSize="25dp" >
      </TextView>

      <TextView
         android:id="@+id/result"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_marginLeft="5sp"
         android:textSize="15sp"
         android:textStyle="bold" >
      </TextView>
   </LinearLayout>
</FrameLayout>

<LinearLayout
   android:id="@+id/LinearLayout03"
   android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content" >

        <Button
            android:id="@+id/addValues"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:minHeight="60dp"
            android:text="@string/add"
            android:textSize="20dp" >

        </Button>

        <Button
            android:id="@+id/multiplyValues"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:minHeight="60dp"
            android:text="@string/multiply"
            android:textSize="20dp" >

        </Button>
    </LinearLayout>

</LinearLayout>
```

A.2.3 MainActivity.java

```
package it.mrcrack.android.simplecalc;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    final String LOG_TAG = "MainScreen";
    super.onCreate(savedInstanceState);
    setContentView(R.layout.mainactivity);
    final EditText value1 = (EditText) findViewById(R.id.value1);
    final EditText value2 = (EditText) findViewById(R.id.value2);
    final TextView result = (TextView) findViewById(R.id.result);

    Button addButton = (Button) findViewById(R.id.addValues);
    addButton.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        try {
          int val1 = Integer.parseInt(value1.getText().toString());
```

```
            int val2 = Integer.parseInt(value2.getText().toString());
            val2 = val2 < 0 ? -val2 : val2;

            Integer answer = val1 + val2;
            result.setText(answer.toString());
          } catch (Exception e) {
            Log.e(LOG_TAG, "Failed to add numbers", e);
            Intent instr = new Intent(v.getContext(), InstructionsActivity.
                class);
            startActivity(instr);
          }
        }
      });
      Button multiplyButton = (Button) findViewById(R.id.multiplyValues);
      multiplyButton.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
          try {
            int val1 = Integer.parseInt(value1.getText().toString());
            int val2 = Integer.parseInt(value2.getText().toString());
            Integer answer = val1 * val2;
            result.setText(answer.toString());
          } catch (Exception e) {
            Log.e(LOG_TAG, "Failed to multiply numbers", e);
            Intent instr = new Intent(v.getContext(), InstructionsActivity.
                class);
            startActivity(instr);
          }
        }
      });
    }
}
```

A.2.4 instructions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:id="@+id/MainLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:padding="3sp"
        android:text="@string/instructions"
        android:textSize="25dp" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        android:padding="3sp"
```

```
        android:text="@string/instructions_lorem"
        android:textSize="15dp" />

    <Button
        android:id="@+id/back"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:minHeight="60dp"
        android:textSize="20dp"
        android:text="@string/back" />
</LinearLayout>
```

A.2.5 InstructionsActivity.java

```
package it.mrcrack.android.simplecalc;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class InstructionsActivity extends Activity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.instructions);

    Button addButton = (Button) findViewById(R.id.back);
    addButton.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        InstructionsActivity.this.finish();
      }
    });
  }
}
```

A.2.6 strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">SimpleCalc</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="hello">Unit Testing Simple Calculator</string>
    <string name="hint1">Enter value</string>
    <string name="hint2">Enter value two</string>
    <string name="resultLabel">Result:</string>
    <string name="add">Add values</string>
    <string name="multiply">Multiply values</string>
    <string name="instructions">Instructions</string>
```

```
    <string name="instructions_lorem">Lorem ipsum dolor sit amet,
       consectetur adipiscing elit. Ut euismod, nibh sed commodo varius
       , ligula odio pulvinar nisl, sit amet bibendum magna tellus in
       magna. Nunc a massa tellus. Fusce dignissim mollis neque eget
       vulputate. Lorem ipsum dolor sit amet.</string>
    <string name="back">Back</string>

</resources>
```

A.3 Sorgenti Test JUnit

A.3.1 MainActivityLayoutTest.java

```
package it.mrcrack.android.simplecalc.test;

import it.mrcrack.android.simplecalc.MainActivity;
import it.mrcrack.android.simplecalc.R;
import android.graphics.Rect;
import android.test.ActivityInstrumentationTestCase2;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivityLayoutTest extends
   ActivityInstrumentationTestCase2<MainActivity> {
  private View mainLayout;
  private MainActivity mainActivity;

  /**
   * Default constructor
   */
  public MainActivityLayoutTest() {
    super(MainActivity.class);
  }

  /**
   * Initialize result textview
   */
  protected void setUp() throws Exception {
    super.setUp();
    mainActivity = getActivity();
    mainLayout = (View) mainActivity.findViewById(R.id.MainLayout);
  }

  /**
   * Test Add button layout
   */
  public void testAddButtonOnScreen() {
    int fullWidth = mainLayout.getWidth();
    int fullHeight = mainLayout.getHeight();
    int[] mainLayoutLocation = new int[2];
    mainLayout.getLocationOnScreen(mainLayoutLocation);
    int[] viewLocation = new int[2];
    Button addButt = (Button) mainActivity.findViewById(R.id.addValues)
       ;
```

```
  addButt.getLocationOnScreen(viewLocation);
  Rect outRect = new Rect();
  addButt.getDrawingRect(outRect);
  assertTrue("Add button off the right of the screen", fullWidth
     + mainLayoutLocation[0] > outRect.width() + viewLocation[0]);
  assertTrue("Add button off the bottom of the screen", fullHeight
     + mainLayoutLocation[1] > outRect.height() + viewLocation[1]);
}

/**
 * Test Multiply button layout
 */
public void testMultiplyButtonOnScreen() {
  int fullWidth = mainLayout.getWidth();
  int fullHeight = mainLayout.getHeight();
  int[] mainLayoutLocation = new int[2];
  mainLayout.getLocationOnScreen(mainLayoutLocation);
  int[] viewLocation = new int[2];
  Button multiplyButt = (Button) mainActivity
     .findViewById(R.id.multiplyValues);

  multiplyButt.getLocationOnScreen(viewLocation);
  Rect outRect = new Rect();
  multiplyButt.getDrawingRect(outRect);
  assertTrue("Multiply button off the right of the screen", fullWidth
     + mainLayoutLocation[0] > outRect.width() + viewLocation[0]);
  assertTrue("Multiply button off the bottom of the screen",
      fullHeight
     + mainLayoutLocation[1] > outRect.height() + viewLocation[1]);
}

/**
 * Test Value1 input layout
 */
public void testValue1EditTextOnScreen() {
  int fullWidth = mainLayout.getWidth();
  int fullHeight = mainLayout.getHeight();
  int[] mainLayoutLocation = new int[2];
  mainLayout.getLocationOnScreen(mainLayoutLocation);
  int[] viewLocation = new int[2];
  EditText value1 = (EditText) mainActivity
     .findViewById(R.id.value1);

  value1.getLocationOnScreen(viewLocation);
  Rect outRect = new Rect();
  value1.getDrawingRect(outRect);
  assertTrue("Value1 EditText off the right of the screen", fullWidth
     + mainLayoutLocation[0] >= outRect.width() + viewLocation[0]);
  assertTrue("Value1 EditText off the bottom of the screen",
      fullHeight
     + mainLayoutLocation[1] >= outRect.height() + viewLocation[1]);
}

/**
 * Test Value2 input layout
```

```
     */
    public void testValue2EditTextOnScreen() {
      int fullWidth = mainLayout.getWidth();
      int fullHeight = mainLayout.getHeight();
      int[] mainLayoutLocation = new int[2];
      mainLayout.getLocationOnScreen(mainLayoutLocation);
      int[] viewLocation = new int[2];
      EditText value2 = (EditText) mainActivity
          .findViewById(R.id.value2);

      value2.getLocationOnScreen(viewLocation);
      Rect outRect = new Rect();
      value2.getDrawingRect(outRect);
      assertTrue("Value2 EditText off the right of the screen", fullWidth
          + mainLayoutLocation[0] >= outRect.width() + viewLocation[0]);
      assertTrue("Value2 EditText off the bottom of the screen",
          fullHeight
          + mainLayoutLocation[1] >= outRect.height() + viewLocation[1]);
    }
}
```

A.3.2 MainActivityMultiplyTest.java

```
package it.mrcrack.android.simplecalc.test;

import it.mrcrack.android.simplecalc.R;
import it.mrcrack.android.simplecalc.MainActivity;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

public class MainActivityMultiplyTest extends
    ActivityInstrumentationTestCase2<MainActivity> {
  private TextView result;
  private static final String N0 = "0";
  private static final String N1 = "1";
  private static final String N2 = "2";
  private static final String N4 = "4";
  private static final String N5 = "5";
  private static final String N7 = "7";
  private static final String ENTER = "ENTER";
  private static final String DOT = "NUMPAD_DOT";
  private static final String MINUS = "NUMPAD_SUBTRACT";
  private static final String RIGHT = "DPAD_RIGHT";

  private static final String N_24_E = enqueue(N2, N4, ENTER);
  private static final String N_74_E = enqueue(N7, N4, ENTER);
  private static final String N_5_DOT_5_E = enqueue(N5, DOT, N5, ENTER)
      ;
  private static final String MINUS_N10_E = enqueue(MINUS, N1, N0,
      ENTER);

  private static final String MULTIPLY_RESULT;
  private static final String MULTIPLY_DECIMAL_RESULT;
  private static final String MULTIPLY_MINUS_RESULT;
```

```
static {
  MULTIPLY_RESULT = String.valueOf(Integer.valueOf(N2 + N4)
     * Integer.valueOf(N7 + N4));

  MULTIPLY_DECIMAL_RESULT = String.valueOf(Double.valueOf(N5 + "." +
     N5)
     * Integer.valueOf(N7 + N4));

  MULTIPLY_MINUS_RESULT = String.valueOf(Integer.valueOf("-" + N1 +
     N0)
     * Integer.valueOf(N7 + N4));
}

private static String enqueue(String... args) {
  StringBuilder toRet = new StringBuilder();

  for (String s : args) {
    toRet.append(s);
    toRet.append(" ");
  }

  return toRet.toString();
}

/**
 * Default constructor
 */
public MainActivityMultiplyTest() {
  super(MainActivity.class);
}

/**
 * Initialize result textview
 */
protected void setUp() throws Exception {
  super.setUp();
  MainActivity mainActivity = getActivity();
  result = (TextView) mainActivity.findViewById(R.id.result);
}

/**
 * Test multiply integer
 */
public void testMultiplyValues() {
  // Send 24 to value1
  sendKeys(N_24_E);
  // Send 74 to value2
  sendKeys(N_74_E);
  // Send Right pad sig
  sendKeys(RIGHT);
  // Send Enter button
  sendKeys(ENTER);
  // Get result
  String mathResult = result.getText().toString();
  assertTrue("Multiply result should be " + MULTIPLY_RESULT + " but
     was "
```

```
        + mathResult, mathResult.equals(MULTIPLY_RESULT));
  }

  /**
   * Test multiply decimal number
   */
  public void testMultiplyDecimalValues() {
    // Send 5.5 to value1
    sendKeys(N_5_DOT_5_E);
    // Send 74 to value2
    sendKeys(N_74_E);
    // Send Right pad sig
    sendKeys(RIGHT);
    // Send Enter button
    sendKeys(ENTER);
    // Get result
    String mathResult = result.getText().toString();
    assertFalse("Multiply result should be " + MULTIPLY_DECIMAL_RESULT
        + " but was " + mathResult,
        mathResult.equals(MULTIPLY_DECIMAL_RESULT));
  }

  /**
   * Test multiply negative number
   */
  public void testMultiplyNegativeValues() {
    // Send -10 to value1
    sendKeys(MINUS_N10_E);
    // Send 74 to value2
    sendKeys(N_74_E);
    // Send Right pad sig
    sendKeys(RIGHT);
    // Send Enter button
    sendKeys(ENTER);
    // Get result
    String mathResult = result.getText().toString();
    assertFalse("Multiply result should be " + MULTIPLY_MINUS_RESULT
        + " but was " + mathResult,
        mathResult.equals(MULTIPLY_MINUS_RESULT));
  }
}
```

A.3.3 MainActivitySumTest.java

```
package it.mrcrack.android.simplecalc.test;

import it.mrcrack.android.simplecalc.R;
import it.mrcrack.android.simplecalc.MainActivity;
import android.test.ActivityInstrumentationTestCase2;
import android.widget.TextView;

public class MainActivitySumTest extends
    ActivityInstrumentationTestCase2<MainActivity> {
  private TextView result;
  private static final String N0 = "0";
```

```
private static final String N1 = "1";
private static final String N2 = "2";
private static final String N4 = "4";
private static final String N5 = "5";
private static final String N7 = "7";
private static final String ENTER = "ENTER";
private static final String DOT = "NUMPAD_DOT";
private static final String MINUS = "NUMPAD_SUBTRACT";

private static final String N_24_E = enqueue(N2, N4, ENTER);
private static final String N_74_E = enqueue(N7, N4, ENTER);
private static final String N_5_DOT_5_E = enqueue(N5, DOT, N5, ENTER)
    ;
private static final String MINUS_N10_E = enqueue(MINUS, N1, N0,
    ENTER);

private static final String ADD_RESULT;
private static final String ADD_DECIMAL_RESULT;
private static final String ADD_MINUS_RESULT;

static {
  ADD_RESULT = String.valueOf(Integer.valueOf(N2 + N4)
      + Integer.valueOf(N7 + N4));

  ADD_DECIMAL_RESULT = String.valueOf(Double.valueOf(N5 + "." + N5)
      + Integer.valueOf(N7 + N4));

  ADD_MINUS_RESULT = String.valueOf(Integer.valueOf("-" + N1 + N0)
      + Integer.valueOf(N7 + N4));
}

private static String enqueue(String... args) {
  StringBuilder toRet = new StringBuilder();

  for (String s : args) {
    toRet.append(s);
    toRet.append(" ");
  }

  return toRet.toString();
}

/**
 * Default constructor
 */
public MainActivitySumTest() {
  super(MainActivity.class);
}

/**
 * Initialize result textview
 */
protected void setUp() throws Exception {
  super.setUp();
  MainActivity mainActivity = getActivity();
  result = (TextView) mainActivity.findViewById(R.id.result);
```

```
  }

  /**
   * Test add integer
   */
  public void testAddValues() {
    // Send 24 to value1
    sendKeys(N_24_E);
    // Send 74 to value2
    sendKeys(N_74_E);
    // Send Enter button
    sendKeys(ENTER);
    // Get result
    String mathResult = result.getText().toString();
    assertTrue("Add result should be " + ADD_RESULT + " but was "
        + mathResult, mathResult.equals(ADD_RESULT));
  }

  /**
   * Test add decimal number
   */
  public void testAddDecimalValues() {
    // Send 5.5 to value1
    sendKeys(N_5_DOT_5_E);
    // Send 74 to value2
    sendKeys(N_74_E);
    // Send Enter button
    sendKeys(ENTER);
    // Get result
    String mathResult = result.getText().toString();
    assertFalse("Add result should be " + ADD_DECIMAL_RESULT + " but
        was "
        + mathResult, mathResult.equals(ADD_DECIMAL_RESULT));
  }

  /**
   * Test add negative number
   */
  public void testAddNegativeValues() {
    // Send -10 to value1
    sendKeys(MINUS_N10_E);
    // Send 74 to value2
    sendKeys(N_74_E);
    // Send Right pad sig
    sendKeys(ENTER);
    // Send Enter button
    String mathResult = result.getText().toString();
    assertFalse("Add result should be " + ADD_MINUS_RESULT + " but was
        "
        + mathResult, mathResult.equals(ADD_MINUS_RESULT));
  }
}
```