UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

DIEGO RODRIGUES DE ALMEIDA

CONTEXT-AWARE PATH-BASED ANDROID APPLICATIONS TESTING

CAMPINA GRANDE - PB

2020

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

Context-Aware Path-Based Android Applications Testing

Diego Rodrigues de Almeida

Tese submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Dra. Patrícia D. L. Machado, Dr. Wilkerson L. Andrade
(Orientadores)

Campina Grande, Paraíba, Brasil

A447c Almeida, Diego Rodrigues de.
Context-aware path-based android applications testing / Diego Rodrigues de Almeida. - Campina Grande, 2020.
193f. : il . Color.

Tese (Doutorado em Ciência da Computação) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2020.
"Orientação: Profa. Dra. Patrícia Duarte de Lima Machado, Prof. Dr. Wilkerson de Lucena Andrade".
Referências.

1. Testing Automation. 2. Android. 3. Path-based Applications. 4. Context-aware Application. I. Machado, Patrícia Duarte de. II. Andrade, Wilkerson de Lucena. III. Título.

CDU 004.415.53(043)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECÁRIA ITAPUANA SOARES DIAS CRB-15/93

CONTEXT-AWARE PATH-BASED ANDROID APPLICATIONS TESTING

DIEGO RODRIGUES DE ALMEIDA

TESE APROVADA EM 31/07/2020

PATRICIA DUARTE DE LIMA MACHADO, PhD, UFCG
Orientador(a)

WILKERSON DE LUCENA ANDRADE, Dr., UFCG
Orientador(a)

TIAGO LIMA MASSONI, Dr., UFCG
Examinador(a)

JORGE CESAR ABRANTES DE FIGUEIREDO, Dr., UFCG
Examinador(a)

ROBERTA DE SOUZA COELHO, Dra., UFRN
Examinador(a)

ARILO CLAUDIO DIAS NETO, Dr., UFAM
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Smartphones geralmente possuem vários sensores, como bússola, GPS, acelerômetro, pedômetro, etc., permitindo que os aplicativos estejam cientes do ambiente em que estão executando. Aplicativos sensível ao contexto são aplicativos orientados a eventos que detectam e reagem às informações de contexto fornecidas pelos sensores para fornecer informações e/ou serviços ao usuário. Comparado aos aplicativos de desktop e da Web, os aplicativos sensíveis ao contexto apresentam desafios adicionais aos testes. Esses aplicativos devem processar a entrada dos usuários, bem como vários valores de entrada de contextos em constante mudança que podem levar à explosão de possíveis situações, algumas das quais são muito difíceis de executar. Aplicativos baseados em caminhos são uma classe muito utilizada de aplicações sensíveis ao contexto. Esses aplicativos monitoram continuamente e em tempo real o percurso que o usuário realiza enquanto utiliza a aplicação. O objetivo deste trabalho é apresentar uma abordagem que permita o teste black-box de aplicativos Android sensíveis ao contexto baseados em caminhos e, dessa forma, possibilite selecionar e executar um conjunto viável de cenários a serem testados. Nossa abordagem consiste em selecionar, por meio de pairwise testing, combinações de valores de sensores com eventos que ocorrem durante a execução do aplicativo em teste (AUT). Desenvolvemos uma ferramenta que permite ao testador executar variação de contexto, simulando valores de sensores em um ambiente emulado que possibilita testar cenários difíceis de executar manualmente. Atualmente, a ferramenta suporta dados de contexto GPS e status de conexão de Internet. Esse tipo de dados é usado por quase todos os aplicativos disponíveis atualmente. Para avaliar nossa abordagem, realizamos um estudo empírico com quatro aplicativos baseados em GPS reais amplamente baixados por usuários do Android. Nossos resultados mostram que nossa abordagem foi capaz de executar um conjunto otimizado de diferentes cenários e encontrar 13 defeitos nos quatro aplicativos, dos quais 6 são defeitos de contexto, com 2 deles detectados ao executar cenários de difícil execução manual. A combinação de pairwise testing com a execução de casos de teste em um ambiente emulado mostrou-se eficaz, pois nos permitiu encontrar defeitos de contexto em aplicativos amplamente usados com um conjunto otimizado de casos de teste.

Abstract

Smartphones usually have several sensors such as touch screen, compass, gyroscope, GPS, accelerometer, pedometer, and so on, allowing applications to be aware of the environment on which they are running. Context-aware applications are event-driven applications that sense and react to context information given by sensors to provide information and/or services to the user. Compared to desktop and web applications, context-aware applications present additional challenges to testing. These applications must process input from users as well as various input values from constantly changing contexts that can lead to the explosion of possible situations, some of which are very difficult to execute. A widely used class of context-sensitive applications are path-based applications. These applications continuously and in real-time monitor the path that the user takes while using the application. This work aims to present an approach that makes it possible the black-box testing of context-aware path-based Android applications and, in this way, makes it possible to select and execute an viable set of scenarios to be tested. Our approach consists of selecting, through pairwise testing, combinations of sensor values with events that occur during the execution of the application under test (AUT). We developed a tool that allows the tester to execute context variation by simulating sensor values in an emulated environment that makes it possible to test scenarios that would be difficult to execute manually. The tool currently supports GPS context data and internet status connection. These kind of data is used by almost all available applications nowadays. To evaluate our approach, we performed an empirical study with four real GPS based applications widely downloaded by Android users. Our results show that our approach was able to execute an viable set of different scenarios and to find 13 defects in the four applications, of which 6 are context defects, with 2 of them detected when executing challenging scenarios for manual execution. Combining pairwise testing with the execution of test cases in an emulated environment proved to be effective. It allowed us to find context defects in applications widely used with an viable set of test cases.

Agradecimentos

Agradeço primeiramente a Deus pela minha saúde e oportunidade de poder ter tido uma boa educação tanto moral quanto acadêmica. Agradeço a Ele ainda, por ter ouvido algumas das preces que fiz e ter atendido todas aquelas preces que não fiz. Por ter me guiado sempre pelo melhor caminho, mesmo que por muitas vezes contra minha vontade.

Agradeço também ao meu pai José Neide Neres de Almeida (*in memoriam*) e minha mãe Maria de Fátima Rodrigues de Almeida por terem sempre estado ao meu lado. Por terem tido a coragem de acreditar em mim, ainda quando eu mesmo não acreditava. Por terem sempre me apoiado nos meus momentos de alegria e mais ainda nos meus momentos de tristeza. Pelo sacrifício que fizeram para poder financiar a melhor educação que estivesse ao nosso alcance financeiro.

Agradeço também aos meus irmãos Thiago Danillo Rodrigues de Almeida e Diogo Rodrigues de Almeida por me apoiarem nas minhas decisões corretas e por criticarem as equivocadas. Por terem sido companheiros e sempre zelarem pelo melhor para mim.

Agradeço aos meus orientadores Patrícia Duarte de Lima Machado e Wilkerson de Lucena Andrade por terem sido muito pacientes e me mostrado o caminho a ser seguido para a construção deste trabalho. Agradeço pela sua ética, coerência, companheirismo e confiança durante todos os anos que trabalhamos juntos além desse programa de Doutorado.

Por fim, agradeço a todas as pessoas as quais posso chamar de amigos e que de forma direta ou indireta contribuíram para este trabalho.

Contents

List of Symbols

ADB - *Android Debug Bridge*

AUT - *Application Under Test*

AVD - *Android Virtual Device*

ADF - *Android Development Framework*

CAA - *Context-Aware Application*

EDS - *Event-Driven Software*

GPS - *Geographical Positioning System*

GUI - *Graphical User Interface*

MBT - *Model-Based Testing*

OCL - *Object Constraint Language*

RTES - *Real-Time Embedded Systems*

SLR - *Systematic Literature Review*

SMS - *Systematic Mapping Study*

UI - *User Interface*

UML - *Unified Modeling Language*

List of Figures

List of Tables

List of Source Codes

Chapter 1

Introduction

Mobile applications have become more than entertainment. They are increasingly pervasive in such a way that active users are wholly dependent on them. While mobile applications have been developed primarily for the entertainment industry, they are now touching more critical sectors, such as payment systems. The exponential growth of this market and the criticality of system development demand greater attention to the reliability aspects of mobile devices' applications. As demonstrated in some studies [61], [92], [110], mobile applications are not bug free and new software engineering approaches are required to test these applications [139].

Testing mobile applications is challenging. Accordingly to Muccini *et al.* [110], mobile applications have a few peculiarities that make their testing more complicated than conventional computer software. Some of the peculiarities are connectivity, limited resources, autonomy, user interface, context-awareness, and diversity of devices, operating systems, and touch screens.

Therefore, as the testing difficulty increases, so does the need for testing automation of mobile applications. Most of the researcher's and practitioner's efforts in this area target the Android platform for multiple reasons [46]. Some of them are:

i At the moment, Android has the largest mobile market share, which makes Android extremely appealing for industry practitioners;

ii A range of different devices and different releases have Android installed. As a result, Android apps often suffer from cross-platform and cross-version incompatibilities, which

makes manual testing of these apps particularly expensive and worth automating;

iii Android is an open-source platform. This quality makes it a more suitable target for academic researchers, making possible the complete access to both the apps and the underlying operating system;

iv Android is, nowadays, the most used operating system in smartphones.

According to StatCounter Web Site[2], in February 2020, the number of smartphones using the Android operating system reached a total of 73.3% of the total number of smartphones in the world followed by 25.89% with iOS and 0.81% for the rest. Nowadays, mobile devices have several sensors, such as touch screen, compass, gyroscope, GPS (Geographical Position Systems), accelerometer, pedometer, and so on. These sensors allow applications to know information about the environment in which they are running. Applications that use information from the environment are called Context-aware Applications. These applications make use of environmental information to guide their behavior.

Testing mobile context-aware applications is often a costly and time-consuming process, thus several techniques and tools have been proposed to reduce the cost of testing mobile devices [46], [88], [137], and [150]. Context-aware applications testing presents specific challenges compared to desktop and web applications, such as processing data from continually changing contexts. These applications are prone to bugs that are highly difficult to reproduce and repair [88]. Using conventional techniques variations to test context-aware software systems does not produce evidence on their feasibility to test the context-awareness features in such systems once mobile context-aware applications often have more complex structures to process a wide variety of dynamic context data in real-time [88], [96].

Considering the context-aware path-based applications, the execution scenarios involve the constant variation of GPS values many of which are difficult to be executed or reproduced in a real environment. Thus, the constant change of context, coupled with the wide variety of possible GPS values, can lead to an explosion of possible scenarios to test.

1.1 Problem

Context-aware path-based applications receive user interaction as inputs through taps on GUI elements and also through data provided by the GPS sensors and Internet connection. The outputs of these applications are their observable behavior. Thus, to test these applications, it is necessary to provide GUI and GPS inputs and observe the application's behavior. Context information changes continuously but according to Matalonga *et al.* [97], the techniques used nowadays follow the same strategy: identify predefined and fixed values for context variables.

Testing predefined and fixed values for context variables at unit testing is possible, but testing the continually changing context variation is more suitable at the system testing.

Consider a mobile application that monitors a driver's conduct. The application observes if the driver:

1. Abrupt accelerate;
2. Abrupt brakes;
3. Drives counter-hand;
4. Speeds above allowed;
5. Performs a high-speed curve;
6. Parks in a prohibited place;
7. Overtakes in a prohibited place.

These are context configurations that the application should judge as reckless and inform the user that he should avoid driving the vehicle in this way. These situations are difficult and costly to test. Also, testing variations of these settings is even more challenging. For example, imagine a motorist driving at a constant speed below the maximum allowable. Subtly he makes a curve without slowing down, drives at the same speed below the maximum allowed, and then goes to the counter-hand. Will the application detect the two imprudences in this scenario? We can observe that in this example, there are four contextual configurations in sequence:

i The driver drives at a constant speed below the maximum allowed;

ii The driver makes a curve without slowing down;

iii The driver drives at the same speed below of the maximum allowed;

iv The driver enters the counter-hand.

These scenarios are challenging to test at system level. They can even put the driver's life at risk. System testing of context-aware path-based Android Applications requires interacting with the AUT through inputs and observing outputs. This thesis focuses on the black-box test for context-aware path-based Android applications whose implementation and internal details are unknown.

The testing technique most used in Android applications is the GUI testing technique (Subsection 3.3.1). The GUI testing technique does not allow us to test scenarios like these because they vary according to the context variation and not through user interaction with the application interface. Existing tools that test context-aware applications do not support the test case generation and execution the asynchronous context variations considering with large sensors data (Subsection 3.3.2). So how to test the path-based ones? Besides, some events may not be part of the application specifications, but that are possible realities. Some examples are GPS inaccuracy, internet crash, phone call, orientation changing, and so on. Considering the 7 scenarios mentioned and only the 4 examples of events above, we have $7 \times 2 \times 2 \times 2 \times 2 = 112$ possible combinations of scenarios. In this example, each event has only two possible values. Therefore, as we increase the number of events or the number of possible values for each event, the number of possible combinations increases exponentially. Hence, the combination of execution scenarios with events that can occur during the execution of these scenarios can lead to a large number of test cases. Thus, how to generate a viable number of test cases that combine execution scenarios with events that can occur at any time?

1.2 Objective and Research Questions

Manually testing context-aware path-based applications can be very expensive or impractical. Therefore, testing these applications fully or partially automatic may be the best op-

tion in certain situations. At the unit level, methods and classes are tested in isolation as a software unit. In context-aware applications, many behaviors are unfeasible to test at the unit level. For example, how to test, at the unit level, if the level battery is low, will the application consume fewer resources? Or how to test if GPS is not well accurate, will the application use other sensors to locate it better? Therefore, this research has the following objective:

Objective *Propose a solution to test Android context-aware path-based applications semi-automatically.*

More specifically, this research is interested in automating the generation of test cases and making their execution semi-automatic at the system level. The execution must be semi-automatic because the decision of the verdict is the responsibility of the tester. Therefore, during the execution of each test case, the approach sends the inputs to the AUT automatically. At the same time, the tester observes its execution and decides whether the test case passes or fails. Also, the test should be feasible and must support (i) the testing of events that can occur during the test case execution, and (ii) the constant context variation. As the tester observes the execution of the AUT, this work focuses on system testing. In this context, this thesis addresses the following research questions:

Research Question 1: *What are the Android context-aware testing studies and tools published in the literature?*

To answer this research question, we performed a systematic mapping in the literature looking for tools or testing techniques for context-aware Android applications, Chapter 3. The discovered tools were then analyzed to identify their potential in testing Android context-aware path-based applications. We found 68 studies and 80 tools for testing Android applications. From these tools, five are tools for testing context-aware applications and five are not developed for context-aware applications, but support the test of the context-aware feature. We were able to identify that many works deal with the generation of GUI inputs for Android applications, but few studies on sensor inputs. Moreover, they do not support asynchronous context variations nor events that can occur during the test case execution.

Research Question 2: *How to create and execute test cases to context-aware path-based Android applications?*

To answer this research question, we developed an approach for generating and executing

test cases for context-aware path-based applications at the system level, Chapter 4. In our approach, test cases are defined by GPS sensor data sets and sequences of events that are more likely to find defects. These events were selected based on a search for problems reported in open-source context-aware applications hosted on GitHub [11] and on the work of Amalfitano *et al.* [25] which present a set of events that are most likely to cause failures. Our approach uses pairwise testing to select combinations of GPS sensor values with events that occur during the execution of the application under test (AUT). The pairwise testing avoids the explosion of the number of event combinations, minimizing the number of test cases significantly but maintaining their effectiveness. Regarding the test case execution, our approach proposes a simulated execution that allows the execution of test scenarios at low cost, and that can even test scenarios that otherwise might not be tested.

Research Question 3: *What kind of defects can the approach reveal?*

To answer this research question, we developed a tool (Chapter 5) that sends events to the AUT, simulating the variation of contexts in order to evaluate our approach. The tool supports the testing of GPS data and internet connection status. Thus, we evaluated our technique through two studies (Chapter 6) using the implemented tool and four widely used path-based applications. The results indicated that our technique was able to find 13 defects which we classified into five groups of causes: (i) GPS Calibration, (ii) Simulation of long background, (iii) internet crash in a long path, (iv) Activity went to the background or device orientation changed and (v) by internal reasons. Among the 13 defects, six are context defects. Besides, we found two context defects that are difficult to execute manually: (i) defect due to GPS calibration, and (ii) defect due to a drop in internet connection over a long path. To the best of our knowledge, no tools have been found in the literature that simulates the long paths under adverse situations (i.e., the variation of GPS calibration, GPS signal fall and recovery, internet connection fall and recovery). Thus, to date, we do not know any other tool capable of executing these scenarios and discovering the defects found in our evaluation.

Our results showed that our approach is promising in the generation of test cases with the potential to find context defects while drastically reducing the number of test cases that would be generated without pairwise testing. The execution in an emulated environment makes it possible to run scenarios that would be very difficult to run in a real environment and therefore allowed to find defects in real applications widely used by Android users.

1.3 Contributions

The main contributions of this work are:

- A systematic mapping exposing as main contributions: (i) the Android testing tools published in the literature, (ii) the testing technique most used in Android testing, and (iii) the research gaps addressed in android context-aware testing [22];
- A set of scenarios most likely to fail in path-based applications based on a search for problems reported in open source context-aware applications hosted on Github [11] and on the work of Amalfitano *et al.* [25];
- A test case generation approach whose main idea is to define test cases as sequences of events that are more likely to find defects. In the test case generation, we use pairwise testing to select test cases and avoid the explosion of the number of event combinations, minimizing the number of test cases significantly but maintaining their effectiveness [23];
- An execution approach that makes feasible the execution of challenging test scenarios;
- A tool that implements the black-box testing approach proposed by our work. The tool makes possible the generation and execution of path-based applications test cases [24].

1.4 Thesis Structure

The remaining parts of this document are structured as follows:

Chapter 2: Background This chapter presents the necessary theory to understand the work proposed in this thesis. Concepts of Android Applications, Android application testing, context-aware applications, and pairwise testing are presented.

Chapter 3: Systematic Mapping This chapter presents the planning, execution, and results obtained from a systematic mapping that we carry out, aiming to identify and discuss the state-of-the-art tools that allow the automation of testing Android context-aware applications.

Chapter 4: Context-Aware Testing Approach This chapter presents the test case generation approach proposed by our work.

Chapter 5: Tool This chapter presents the ENVIAR tool that implements the approach described in Chapter 4. It is presented an overview of the tool and details about its functionality, architecture, testing process, and the implementation of the test suite creation and execution.

Chapter 6: Evaluation This chapter describes how an experiment was planned, executed, and extracted the results.

Chapter 7: Related Works This chapter gives an overview of relevant work on testing context-aware Android applications, presenting limitations and differences with our work.

Chapter 8: Concluding Remarks This chapter presents the concluding remarks and prospects for future work.

Chapter 2

Background

This chapter describes the concepts used in this document to make it self-contained. It will be discussed concepts as Android Operation System, Android Applications, Android Testing, Context-Aware Application Testing and Context Failures, and Pairwise Testing.

2.1 Android Operating System

Android is Google's mobile operating system, and it is currently the world leader in this segment. Android is available for several platforms such as smartphones, tablets, TV (Google TV), watches (Android Wear), glasses (Google Glass), cars (Android Auto), and it is the most widely used mobile operating system in the world.

Developers use Java to build Android applications. However, there is no Java Virtual Machine (JVM) in the Android operating system. In fact, until before the Android 4.4 (KitKat), a virtual machine named Dalvik was used to run on mobile devices. After that, ART (Android Runtime) took the place of Dalvik. Thus, the bytecode (.class) is converted to the .dex (Dalvik Executable) format as soon as its compilation finishes. After that, the .dex files and other resources like images are compressed into a single .apk (Android Package File) file, representing the final application. Android applications run on top of the Android framework, as can be seen in Figure 2.1.

Android comes with a set of pre-installed applications such as calendars, email, contacts, SMS messaging, internet browsing, and so on. User-installed applications have no special priority or status over pre-installed applications. User-installed applications can become

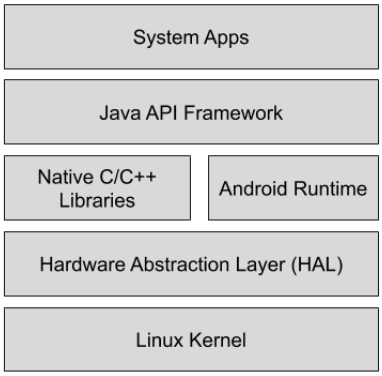


Figure 2.1: Android Framework (adapted from [3])

standard applications such as SMS messenger, web browser, or even the standard keyboard (but there are some exceptions, such as the system settings application)[1].

Android application developers enjoy an available set of APIs written in Java that brings together the Android operating system's full set of features. These APIs allow the reuse of system components and services, facilitating the development of Android applications. The Android platform also provides Java APIs that expose native code functionality written in C and C ++. For example, it is possible to develop 2D and 3D graphics using the Android OpenGL Java API by accessing OpenGL ES.

Each application runs in its process and with its Android Runtime (ART) instance since Android 4.4 (KitKat). ART was designed to support running multiple virtual machines, even on low memory devices. For this, ART executes DEX files that are a bytecode file type designed for Android and optimized to use minimal memory space.

Android is based on the Linux kernel structure. For example, to manage low-level chaining and memory, ART uses the Linux kernel. Besides, using the Linux kernel also has the advantage of reusing the security features that the Linux kernel provides. This allows device manufacturers to develop hardware drivers for a known kernel.

[1]https://developer.android.com/guide/platform/index.html

2.2 Android Applications

Android applications were initially developed only in Java and run on top of the Java API Framework (Section 2.1). After Google I/O[2] 2017, Android applications can be written using either Java or Kotlin[3]. In general, an android application can be composed of four component categories: (i) Activity, (ii) Broadcast Receiver, (iii) Content Provider, and (iv) Service.

i **Activities** are focused windows, and they are the only type of components that contain Graphical User Interfaces (GUI) in which the user interaction occurs. The Activities behaviors are implemented in a .java file while the Activities structure graphical interface is described in a .xml file. Just one Activity can be active at a time. Therefore, the Android operating system manages the Activities that are in the background and the Activity that is displayed according to the Activities lifecycle, Figure 2.2. The Activity lifecycle organizes each Activity's state during its RAM allocation. Each state represents situations the Activities assume from the moment of their creation until their destruction. During the creation of an Activity ("Created" state), the onCreate() method is called. The developer must implement the onCreate() method to manage the Activity object's variables as well as everything needed to create that object. Activity assumes the "Started" state when it goes to the foreground after it has been created or after the "Stopped" state. An Activity goes to the "Resumed" state after the "Started" state or after the "Paused" state. When in the "Resumed" state, the onResume() method is called. The developer must implement in the onResume() method all the care that must be taken when the Activity returns to the foreground. Activity goes to the "Paused" state when Activity goes to the background. In this state, the onPaused() method is called. The developer must implement all necessary precautions in the onPaused() method when an Activity goes to the background (i.e., turn off the GPS to save battery). When the Android operating system needs to free up RAM space, it puts Activities in the "Paused" state to the "Stopped" state. In this state, all Activity variables are deallocated from the device's RAM. When finishing an application, its background's Activities goes to the "Stopped" state. Then, all Activities' applications go to the "Destroyed" state where all the memory space of all Activities is

[2]https://events.google.com/io

[3]https://kotlinlang.org

freed up.

ii **Services** are used to perform long time operations and are always executed in the background. For example, an instant messaging application may be looking for new messages while another application executes in the foreground. Services do not have a graphical interface. Therefore, most testing tools are not suitable for testing Services.

iii **Content Providers** act as a structured interface to shared data stores, such as contacts, photos, and calendar databases. Applications may have their content providers and may make them available to other apps.

iv **Broadcast Receivers** are always listening and reacting to broadcast announcements. For example, an application may receive a low battery notification and change its appearance to darker colors.

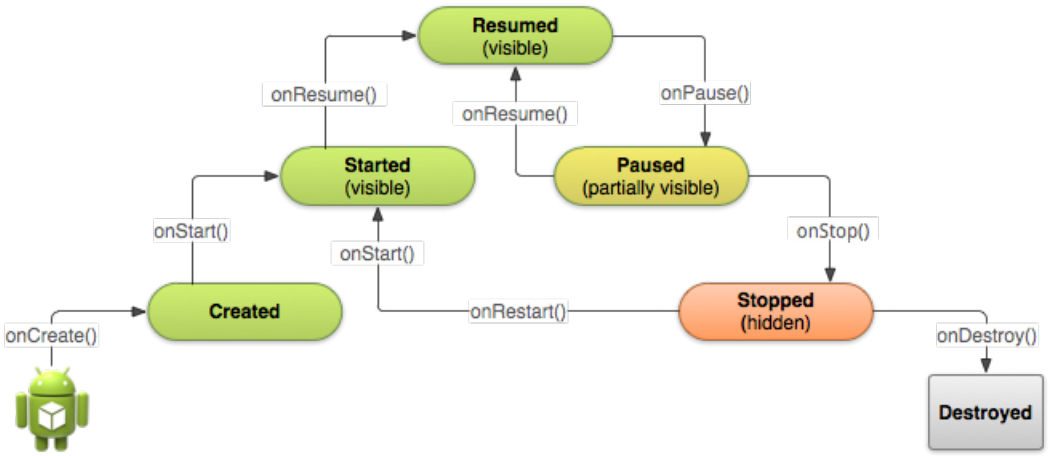


Figure 2.2: Activity Lifecycle (from [3])

One of the most important files in the development of Android applications is the XML manifest file. This file provides essential information such as elements of the Activities, Services, Broadcast Receivers, and Content Providers graphical interface.

Although Android applications are GUI-based and mainly written in Java, they differ from Java standalone GUI applications and manifest somehow different kinds of bugs [61], [74]. Existing test input generation tools for Java [55], [94], [100] cannot be directly used to test Android apps, and custom tools must be adopted instead. For this reason, the academic community has made much effort to research Android application testing tools and

techniques. Several test input generation techniques and tools for Android applications have been proposed [148], [137], [106], [81].

2.3 Android Testing

Android apps are event-driven; that is, the program is running waiting for the user to interact with the app via the GUI, sensors interaction, or some system events from the Android OS. In Android, GUI events include user-actions such as clicks, pinches, swaps, scrolls, or system events, such as incoming calls. Users may also use the GUI to enter specific values for the widgets. Inputs can be entering some text-box value, selecting a particular item in a list, clicking in a button, and so on. User-actions and the inputs can be generated either randomly or by following a systematic approach. Generally, in the latter case, a model of the app is used to guide the process and limit the search space. These models can be constructed statically, dynamically, or entirely manually.

Most current techniques for dealing with traditional event-driven systems either use capture-replay (also known as record and replay) or model-driven approaches. In capture-replay approaches [101], the user records his/her interaction sequences with the GUI, which are replayed at the time of testing. Model-driven techniques such as [141], [99] require the user to provide a model usage of the software system. Both capture-replay and model-driven approaches depend on manual effort, requiring considerable effort. There are also efforts to extract directed graph models automatically by crawling the GUI [29], [100], and use those graphs to generate test sequences.

2.4 Context-Aware Applications

The first mobile applications were applications with features of desktop applications but adapted for mobile devices. Muccini *et al.* [110] differentiates mobile applications into two sets:

i App4Mobile: These are traditional applications that have been recompiled for mobile devices;

ii MobileApps: They are mobile applications that use context information to generate context-based results.

Thus, over time, mobile applications have become increasingly pervasive. Therefore, the behavior of applications depends not only on user input but also needs context information. Therefore, today's applications increasingly have characteristics of context-aware applications.

Before understanding what a context-aware application is, it is first necessary to define what is context. Some authors consider context to be the user's environment, while others consider it as the application's environment. Some examples are:

- Brown [41] defines context as the elements of the user's environment that the computer knows about;
- For Franklin and Flaschbart [51], context is the situation of the user;
- Ward, Jones, and Hopper [138] view context as the state of the application's surroundings;
- Rodden *et al.* [120] define it to be the application's setting;
- Hull, Neaves, and Bedford-Roberts [68] consider context as aspects of the environment's current situation.

In this work, we follow the definition provided by Abowd *et al.*:

Definition 2.1 (Context)**.** *Any* ***information*** *that can be used to characterize the situation of* ***entities*** *(e.g. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects.*

Apart from being more general, this definition facilitates the understanding of what information can be considered context information. Location is the context information most commonly used by mobile applications. However, there are several other types of context

information that may be relevant to a mobile application such as temperature, brightness, time, date, mobile device tilt, geographic orientation (north, south, east, west), and so on.

Abowd *et al.* [20] define context-aware applications as "a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task". In this work, we consider **context-aware applications** as defined as follows.

Definition 2.2 (Context-Aware Application)**.** *Context-aware applications are event-driven applications that sense and react to context information given by sensors to provide information and/or services to the user.*

2.5 Pairwise Testing

In many situations, testing all possible combinations of parameter values is a non-feasible task.

For example, imagine that we want to test software that creates disk partitions [15]. To create a partition, the user must choose one of the following values for each of the following parameters:

- Type: Single, Span, Stripe, Mirror, RAID-5;
- Size: 10, 100, 500, 1000, 5000, 10000, 40000;
- Format method: Quick, Slow;
- File system: FAT, FAT32, NTFS;
- Cluster size: 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536;
- Compression: On, Off.

In this example, the software has six parameters: the first one has 5 possible values, the second has 7, the third 2, the fourth 3, the fifth 8, and the sixth 2. Thus, 3,360 ($5 \times 7 \times 2 \times 3 \times 8 \times 2 = 3,360$) test cases would be needed to test all possible combinations. The most common software bugs are usually due to a single parameter value or the interaction of pairs of parameter values [39]. For example, Single, FAT is one pair, 10, Slow is another.

Bugs involving interactions between three or more parameter values are progressively less common [77]. Pairwise Testing (also known as all-pairs testing) is a combinatorial testing technique that, among all possible combinations of parameter values, guarantees to test the smallest number of test cases that cover, at least once, all possible pairs of values.

Table 2.1 illustrates the result of performing pairwise testing for the discussed example. Pairwise testing reduced from 3,360 test cases to just 56. All combinations of value pairs for the type, size, format method, file system, cluster size, and compression parameters are present in the 56 test cases selected by the pairwise testing technique.

Table 2.1: Pairwise testing example.

#	Type	Size	Format method	File system	Cluster size	Compression
1	RAID-5	1000	Quick	FAT32	16384	On
2	Span	5000	Slow	FAT	8192	Off
3	Mirror	10000	Slow	NTFS	32768	On
4	Single	40000	Quick	NTFS	65536	Off
5	Stripe	500	Slow	FAT32	4096	Off
6	Stripe	100	Quick	FAT	65536	On
7	Span	1000	Quick	NTFS	4096	On
8	RAID-5	10000	Slow	FAT	16384	Off
9	Mirror	10	Quick	FAT32	2048	Off
10	Single	5000	Quick	FAT32	32768	On
11	Single	40000	Slow	FAT	1024	On
12	RAID-5	100	Slow	NTFS	32768	Off
13	RAID-5	40000	Slow	FAT32	4096	Off
14	Span	500	Quick	FAT	32768	On
15	Stripe	10	Quick	NTFS	8192	On
16	Span	100	Slow	FAT32	16384	On
17	RAID-5	5000	Slow	NTFS	2048	On
18	Mirror	100	Slow	FAT	4096	Off
19	Span	10000	Slow	FAT32	65536	On
20	Stripe	40000	Quick	FAT32	32768	On
21	Single	1000	Slow	FAT32	8192	Off
22	Stripe	1000	Quick	NTFS	1024	Off
23	Mirror	500	Slow	NTFS	8192	On
24	Span	100	Slow	FAT32	1024	On
25	Mirror	1000	Slow	FAT	2048	On
26	Mirror	5000	Slow	FAT32	1024	Off
27	RAID-5	500	Slow	FAT32	65536	Off
28	Stripe	5000	Quick	NTFS	16384	On
29	Single	500	Quick	FAT	1024	On
30	Mirror	5000	Slow	FAT	65536	Off
31	RAID-5	10	Slow	FAT	1024	Off
32	Mirror	40000	Quick	FAT	512	On
33	Single	10000	Quick	FAT	8192	Off
34	Span	1000	Slow	FAT32	512	Off
35	Single	10	Slow	FAT	16384	Off
36	Span	40000	Quick	FAT32	2048	Off
37	Span	1000	Quick	NTFS	32768	On
38	Span	10	Quick	FAT	65536	Off
39	Stripe	10000	Slow	FAT32	1024	On
40	Single	10	Quick	FAT32	4096	On
41	Stripe	1000	Slow	NTFS	65536	Off
42	Single	10000	Slow	NTFS	2048	Off
43	Single	10000	Slow	FAT	4096	Off
44	Single	10	Quick	FAT32	32768	Off
45	Stripe	500	Slow	NTFS	2048	Off
46	RAID-5	500	Quick	NTFS	512	Off
47	Stripe	5000	Quick	NTFS	512	Off
48	Single	10	Quick	FAT32	512	On
49	RAID-5	10000	Slow	FAT32	512	On
50	Span	5000	Slow	FAT32	4096	Off
51	Single	100	Slow	FAT	512	Off
52	RAID-5	100	Quick	FAT	8192	Off
53	RAID-5	40000	Quick	NTFS	8192	On
54	Stripe	100	Slow	FAT	2048	Off
55	Mirror	500	Quick	NTFS	16384	Off
56	RAID-5	40000	Quick	NTFS	16384	On

2.6 Concluding Remarks

This chapter presented the necessary concepts to understand the work of this thesis. Fundamental concepts about the Android operating system, components and concepts about Android applications, types of Android application tests, the definition of context-awareness and context-aware application, and the pairwise test case selection technique were presented.

Chapter 3

Systematic Mapping

This chapter presents the planning and the execution of a systematic mapping conducted in our work to identify and discuss the state-of-the-art tools that allow the automation of testing Android context-aware applications. This chapter is divided as follows: Section 3.1 details how the systematic mapping was planned and performed. The results are presented in Section 3.2. The analysis and discussion of the results are made in Section 3.3.

3.1 Research Method

The purpose of a Systematic Mapping Study (SMS for short) is to comprehensively identify, evaluate, and interpret all work relevant to the defined research questions. Thus, this section is based on the work of Petersen *et al.* [116] and details the central research questions of this systematic mapping, as well as the procedure followed to identify the relevant studies required to do so.

3.1.1 Research Questions

This systematic mapping aims at summarizing the current state of the art concerning test automation tools for Android context-aware applications. In order to do so, we conducted an SMS following the recommendations defined by Petersen *et al.* [116] and, therefore, proposed the following research questions (RQs):

- RQ1: What are the Android testing tools published in the literature?

  - RQ1.1: What technique do they implement?
  - RQ1.2: What are the most used ones?

- RQ2: What are the Android context-aware testing studies and tools published in the literature?
  - RQ2.1: Which research groups are involved in Android context-aware testing research?
  - RQ2.2: What are the research gaps addressed in Android context-aware testing?

For each main research question, we formulated sub-questions as listed above. Those sub-questions are answered to support our main questions. In RQ1, we aim to determine the existing Android testing tools currently discussed in the literature. In RQ2, we aim to identify and better understand the existing research about Android context-aware testing. To answer the research questions, we searched for studies from four digital libraries, as can be seen in sub-section 3.1.2.

3.1.2 Sources of information

To gain a broad perspective, as recommended in Kitchenham's guidelines [75], we widely searched for references in electronic sources. The following databases were covered:

- ACM Digital Library[1];
- IEEE eXplore[2];
- Science Direct[3];
- Springer Link[4].

These databases cover the most relevant journals, conferences, and workshop proceedings within Software Engineering.

[1] https://portal.acm.org
[2] https://ieeexplore.ieee.org
[3] https://www.sciencedirect.com
[4] https://link.springer.com

3.1.3 Search criteria

In order to select just articles related to potentially Android context-aware applications testing tools, some keywords were defined.

- **Sensibility to context:** Context-aware, context aware, context driven, context sensitive, context sensitivity, pervasive, ubiquitous, self adaptive, self adapt.
- **Others:** Android, test, testing, tool, framework.

As a result of the combination between the keywords and the connectors **AND** and **OR**, the following search string was defined:

```
``Android'' AND
(``context-aware'' OR ``context aware'' OR
 ``context driven'' OR ``context sensitive'' OR
 ``context sensitivity'' OR ``pervasive'' OR
 ``ubiquitous'' OR ``self adaptive'' OR
 ``self adapt'') AND
(``test'' OR ``testing'') AND
(``tool'' OR ``framework'')
```

However, when executing the search string, the number of results was too small. Thus, we decided to split the search string into two search strings: in the first one, we would address studies related to context-aware Android applications; the second one, studies related to Android application testing tools.

Thus, the resulting search strings were the following.

Search String 1:

```
``Android'' AND
(``context-aware'' OR ``context aware'' OR
 ``context driven'' OR ``context sensitive'' OR
 ``context sensitivity'' OR ``pervasive'' OR
 ``ubiquitous'' OR ``self adaptive'' OR
 ``self adapt'')
```

Search String 2:

```
``Android'' AND
(``test'' OR ``testing'') AND
(``tool'' OR ``framework'')
```

3.1.4 Inclusion and Exclusion Criteria

Studies were selected for the SMS if they met the following inclusion criteria:

1. The study describes at least one Android testing tool;
2. The study clearly describes the method and purpose of the testing tool;
3. The study is written in English, Portuguese, or Spanish.

In terms of exclusion criteria, studies were excluded if they:

1. Did not match the inclusion criteria;
2. Did not have relevant information about the tools;
3. Were published before 2008;
4. Described only obsolete Android testing tools.

3.1.5 Study Selection and Extraction

To obtain more confidence in the research results, we divided the study selection into 3 (three) steps: electronic search, selection, and extraction. The electronic search was conducted executing the search strings in the Sources of Information. The search process performed on all databases was based on the advanced search feature provided by the online databases. The search strings were applied using an advanced command search feature and set to include meta-data of studies with initial data set up since 2008.

After executing the search strings in each of the sources of information, 24,005 studies were found. The search reported too many results diverging from the research objective in two cases: Search String 2 of Science Direct and two Search Strings of the Springer Link.

Thus, some filters were considered to obtain results closer to the objective of the work. At Science Direct, the search was filtered to find studies that presented the search string words in the title, abstract, or keywords. In Springer Link, the studies that were of the discipline of Computer Science and had Software Engineering as a subdiscipline were filtered. Thus, after applying the filters, we found 6,648, as can be seen in Table 3.1.

Table 3.1: Studies found in the Electronic Search.

		First Search	Filtered Search
ACM	Search String 1	462	462
	Search String 2	274	274
	TOTAL	736	736
IEEE	Search String 1	725	725
	Search String 2	267	267
	TOTAL	992	992
Science Direct	Search String 1	2284	2284
	Search String 2	4995	294
	TOTAL	7276	2578
Springer Link	Search String 1	4062	763
	Search String 2	10939	1549
	TOTAL	15001	2342
	TOTAL	24005	6648

The systematic mapping was followed using the Start [10] tool. We used this tool to organize the systematic mapping. The Start tool also helped us by automatically identifying duplicate articles. Only the results of Springer Link could not be manipulated in the Start tool because it cannot be exported in any of the formats accepted by the tool. Thus, Microsoft Excel[5] was used. The file containing the search results in the search engines and the Start tool file containing the references can be downloaded here[6].

After reading the abstract of the 6,648 studies resulted from the electronic search, a total

[5]https://www.microsoft.com

[6]http://bit.ly/ArticleArchives

of 143 studies were selected. The selection of these studies was carried out by reading the abstract and verifying if the study belonged to the SMS area of interest. Table 3.2 illustrates how many studies were accepted, rejected, and duplicated in each of the sources of information.

Table 3.2: Selected Studies.

		Quantity
ACM	Accepted	64
	Rejected	624
	Duplicated	48
IEEE	Accepted	45
	Rejected	910
	Duplicated	37
Science Direct	Accepted	13
	Rejected	2536
	Duplicated	29
Springer Link	Accepted	21
	Rejected	1855
	Duplicated	466
TOTAL	Accepted	143
	Rejected	5925
	Duplicated	580

After the complete reading of each of the 143 selected studies, 68 studies were accepted and included in this systematic mapping study. These studies were extracted by reading each of the selected studies and noting whether they meet the inclusion and exclusion criteria of Section 3.1.4. Figure 3.1 summarizes how the studies resulting from the systematic mapping of this work were selected and extracted.

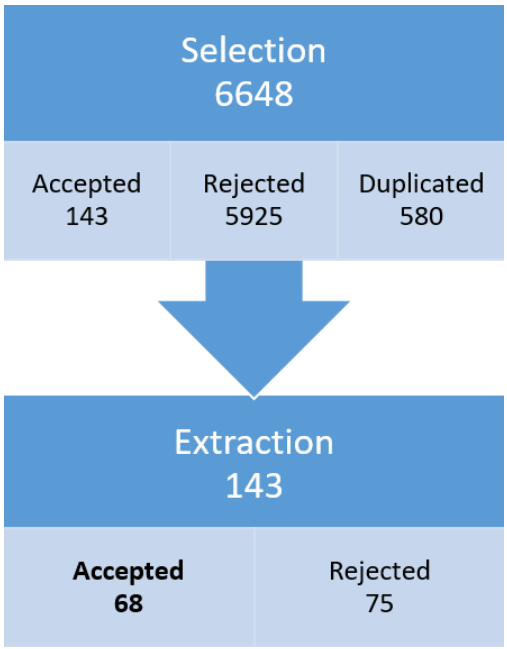


Figure 3.1: Study selection and extraction summary

3.1.6 Study Analysis

All information extracted from the 68 found studies is presented in Section 3.2. We analyzed the number of publications per year, the number of publications per country, the main conferences in which articles were published, and the main authors. The main contribution of this information was the inference of which are the groups that publish most in the area of SMS.

Section 3.3 presents the analysis performed in the 80 tools found in the systematic mapping conducted in this work. In this analysis, we discussed the used testing techniques, which tools generate and/or execute test cases, test case generation strategy, which sensor data each tool considers, test approach, and whether the tool is available for download. This information allowed us to answer the SMS research questions.

3.1.7 Validity Evaluation

There are validity threats associated with all phases during the execution of this SMS. Thus, this section discusses the threats and possible mitigation strategies according to each SMS phase.

Study search and selection

We may have excluded studies during the search due to various reasons such as personal bias; this may negatively impact on the SMS result. The following strategies were used to reduce risks.

1. Four popular databases (e.g., IEEE Explore) on software engineering were included for the database search, and we also used Snowballing in the main studies;
2. We designed and reached an agreement on inclusion and exclusion criteria (see 3.1.4) for selecting studies, which helped avoid wrong exclusions;
3. The electronic search and the study selection was executed twice.

Another threat is that we may have missed primary papers published before the year 2008. The Android operating system was publicly released in September 2008. Since the research is directed to the Android operating system, we do not believe there are any publications in the research area before 2008.

Data extraction and analysis

Personal bias may decrease the quality of the extracted data from the studies (e.g., the incompleteness of the extracted data). The strategy used to mitigate this threat is the conduction of weakly meetings where we discussed:

1. Potential problems in data extraction (e.g., whether certain data should be extracted);
2. Extracted partial results;
3. Potential problems in data analysis.

3.2 Results

The systematic mapping performed resulted in the 68 studies presented in Table 3.3. We can see that they have been published since 2012 (Figure 3.2). The largest number of studies (60.3% of the total) was published in 2014, 2015, and 2016. In 2017 and 2018, the number

of publications has decreased, which supposes the beginning of disinterest in the area (Figure 3.2). However, the systematic mapping focused only on Android testing tools, so we cannot say that the number of studies on Android testing has diminished.

Table 3.3: Accepted Studies

Studies
Imparato [69], Vieira *et al.* [131], Li *et al.* [79], Yang *et al.* [145], Amalfitano *et al.* [28] [27] [25] [30], Griebe and Gruhn [53], Bernardo *et al.* [124], Prathibhan *et al.* [117], Anand *et al.* [33], Zaeem *et al.* [149], Villanes *et al.* [132], Coppola *et al.* [49], Jensen *et al.* [71], Moran *et al.* [107], [106], Haoyin *et al.* [58], Wang *et al.* [135], Anbunathan *et al.* [35], Liu *et al.* [85], McAfee *et al.* [98], Nguyen *et al.* [112], Anbunathan *et al.* [34], Li *et al.* [81], Ye *et al.* [146], Jamrozik *et al.* [70], Machiry *et al.* [90], Hu *et al.* [62], Song *et al.* [125], Linares-Vasquez *et al.* [83], Mahmood *et al.* [91], Merwe *et al.* [130] [129], Meng *et al.* [102], Su *et al.* [126], Hu *et al.* [65], Choi *et al.* [45], Paulovsky *et al.* [115], Hu *et al.* [63], Qin *et al.* [118], Lin *et al.* [82], Wen *et al.* [140], Hao *et al.* [57], Lam *et al.* [78], Mirzaei *et al.* [105], Gomez *et al.* [52], Jun *et al.* [153], Farto *et al.* [50], Mao *et al.* [93], Neto *et al.* [111], Mirzaei *et al.* [104], Adamsen *et al.* [21], Salihu *et al.* [121], Azim *et al.* [36], Kaasila *et al.* [73], Morgado *et al.* [108], Zhauniarovich *et al.* [152], Li *et al.* [80], Hu *et al.* [66], Liu *et al.* [86], Hu *et al.* [64], Cao *et al.* [43], Ami *et al.* [32], Yan *et al.* [144], Chen *et al.* [44] and Koroglu *et al.* [76]

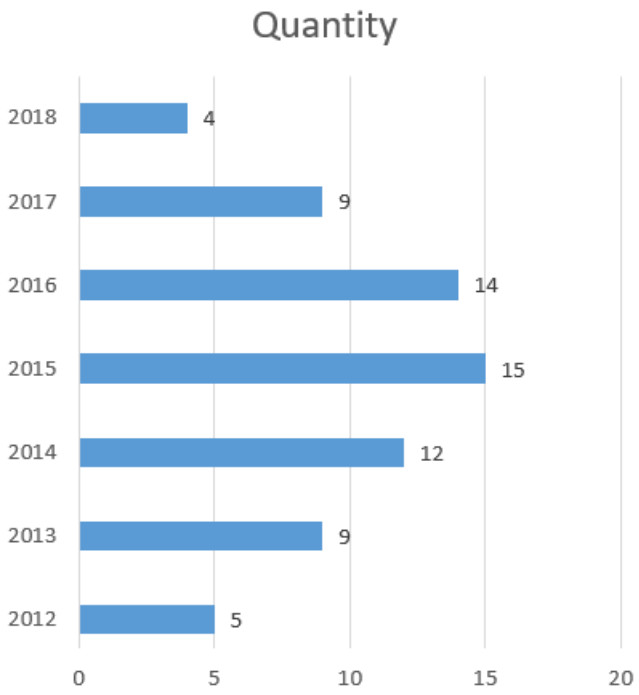


Figure 3.2: Year VS Quantity

From Figure 3.3, the country that most published Android testing tool articles was the USA (with 35.3% of the total), followed by China, Italy, and Brazil.

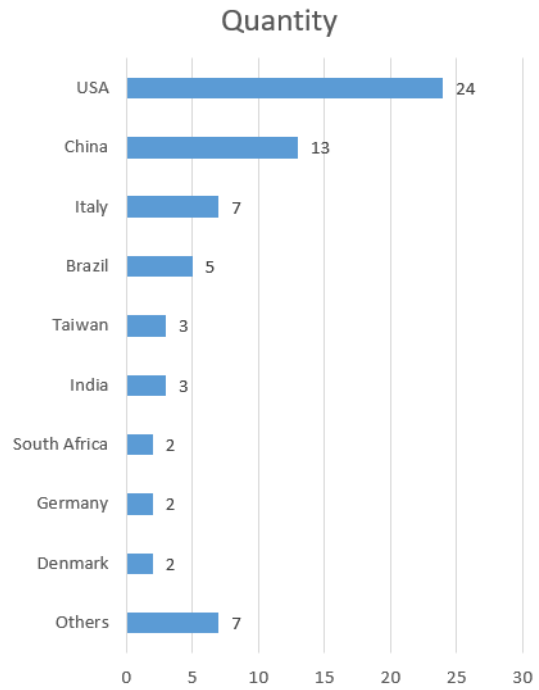


Figure 3.3: Country VS Quantity

Figure 3.4 illustrates the major conferences in which all accepted studies were published. The International Conference on Software Engineering (ICSE) is the event that most accepted studies related to our SMS, 11.8 % of all of them.

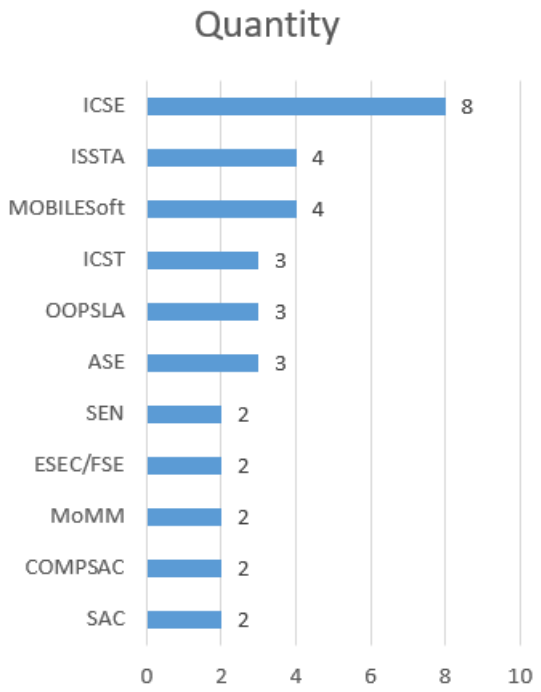


Figure 3.4: Main Conferences

Figure 3.5 shows the authors who most published articles in the research area of this systematic mapping. Analyzing the Google Scholar [8] profile of each of the authors, we can see that Iulian Neamtiu[7] posted 21 articles on Android application testing, Anna Rita Fasolino[8] posted 9, Domenico Amalfitano[9] posted 12, Porfirio Tramontana[10] posted 9, Tanzirul Azim[11] posted 11, and Yongjian Hu[12] posted 10. In addition, considering the studies accepted in this systematic mapping, we noticed that Iulian Neamtiu, Yongjian Hu, and Tanzirul Azim published 6 articles in which at least two of them wrote together. Also, Domenico Amalfitano, Anna Rita Fasolino, and Porfirio Tramontana published 4 articles together. Thus, from the number of articles about Android application testing and the number of articles written to-

[7]https://scholar.google.com/citations?user=8qU-5YMAAAAJ&hl=en
[8]https://scholar.google.com/citations?user=lC5j76YAAAAJ&hl=en
[9]https://scholar.google.com/citations?user=ReafO6YAAAAJ
[10]https://scholar.google.com/citations?user=Q7z44GcAAAAJ&hl=en
[11]https://scholar.google.com/citations?user=YQ72v64AAAAJ&hl=en
[12]https://scholar.google.com/citations?user=gFODw24AAAAJ&hl=en

gether, we can identify two research groups with a significant amount of published work in the scope of this study.

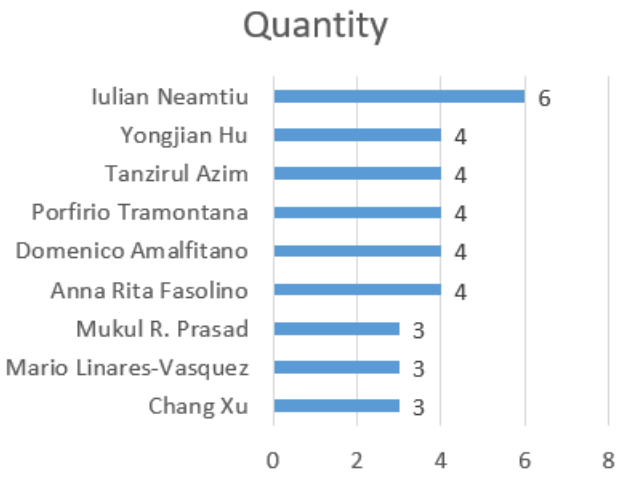


Figure 3.5: Main Authors

3.3 Analysis and Discussion

From the systematic mapping, we found 68 studies about Android testing tools. Section 3.2 presented an overview of the found studies. In this section, we will answer and discuss the research questions elaborated during the systematic mapping planning.

3.3.1 RQ 1: What are the Android testing tools published in the literature?

From the 68 found studies presented in Section 3.2, we identified 80 tools (Table 3.4). The tools were analyzed and classified with respect to:

- Testing technique: The general technique the testing tool implements to test applications;
- Test case generation: Does the tool generate test cases to test the application?
- Generation strategy: If the tool generates test cases, what strategy does the tool apply?
- Use of sensor data: Does the tool consider the data from sensors to test applications?

- Test case execution: Does the tool execute test cases?
- API: Is the tool an API?
- Download availability: Is the tool currently available to download?
- Testing approach: Is the tool designed for black-box, white-box, or gray-box testing?

For the sake of space, the table with the complete classification of the attributes can be found here[13].

Table 3.4: Tools found in the SMS.

Tools
SlumDroid[69] and GUIAnalyzer[69], Espresso[49] [78] [124], Espresso Recorder [78], UIAutomator[49], Selendroid[49], Silk Mobile[49], Sikuli GUI Automation Tool[49], Segen[111], DroidMate[70], FSMdroid[126], SwiftHand[45] [45], A^3E[36] [37], TrimDroid[105], AMOGA[121], AGRippin[28], AndroidRipper[30], Extended AndroidRipper[25], T+[83], QUANTUM[149], Collider[71], EvoDroid[91], VeriDroid[86], JPF-ANDROID[129], Improved JPF-ANDROID[130], Thor[21], Monkey[151], Improved Monkey[58], PUMA[57], Dynodroid[90], ATT[102], Sapienz[93], DroidFuzzer[146], VALERA[65] [63] [64] [66] [78], MobiPlay[118] [78], RERAN[52] [78], Testdroid[73], CrashScope[106] [107], DroidBot[81], M[agi]C[112], ACTEve[33], PATS[140], BBOXTESTER[152], AppDoctor[62], ORBIT[145], Fest[124], EasyMock[124], Hamcrest[124], JUnit[124], Robolectric[124], Robotium[124] [78], Android.Test[124], DroidCrawler[135], Custom-built version of the calabash-android[53], CATE[98], MobiGUITAR[31], [27], Context Simulator[131], ADAutomation[79], MAT[117], AM-TaaS[132], VTE[35], [34], ACRT[85], EHBDroid[125], ATG[115], SPAG-C[82], Appetizer[78], Bot-bot[78], Culebra[78], monkeyrunner[78], Mosaic[78], Ranorex[78], HiroMacro[78], RepetiTouch[78], MAFT[153], MBTS4MA[50], SIG-Droid[104], iMPAcT[108], UGA[80], Xdroid[43], Automate toolkit[60], FragDroid[44], MobiCoMonkey[32], LAND[144], AndroFrame[76] and TCM[76]

RQ 1.1: What technique do they implement?

Each tool tests Android applications through its implemented technique. Table 3.5 shows the main types of testing techniques of the identified tools. GUI Testing tools are the most

[13] http://bit.ly/Found_Tools

common ones. We found 32 tools that base their testing by identifying and exploring the interface elements to test the applications. These represent 40% of the total found tools. GUI Testing tools use algorithms to identify interface elements such as Text Views, Buttons, Check Boxes, and Image Views to generate and/or execute tests. Many tools use these graphical elements to construct state machines and thus determine the application behavioral model.

Some Android applications should be prepared to react to some events coming from the Android operating system such as low battery level, battery charging, incoming call, change of application in the foreground, airplane mode on/off, and so on. Moreover, the interaction between the user and many mobile applications does not occur exclusively through the interface elements. Many of the interactions can also be through sensors like GPS, gyroscope, compass, and accelerometer. Thus, some tools test Android applications not looking at the interface components, but rather through events generated for the application simulating the user touching the screen, events from the system, and data from the device's sensors. For these tools, we call them system events testing tools because they interact with the application under test by stimulating events at the system level. We identified 17 system events testing tools that represent 21% of the total found tools.

In functional testing, as crucial as finding a usage scenario that fails is to be able to replicate it. Replicating a failing usage scenario allows us to identify whether the application defect has been fixed. With that in mind, Record and Replay testing tools were developed. These tools can record a user's usage scenario and run the same scenario as often as the tester wishes. We identified 14 Record and Replay tools, representing 18% of the total found tools.

Many failures occur when executing a bug code. Consequently, the higher code coverage in a usage scenario, the greater the chance of finding bugs. For this reason, some tools test applications to maximize the amount of code covered. Among the tools found in this systematic mapping, 6 of them are Code Coverage tools.

Table 3.5: Types of Testing Technique.

Technique	Quantity of Tools
GUI Testing	32
System events testing	17
Record and Replay	14
Code Coverage	6
Others	11

Among the identified Android testing tools, 45 of them are capable of generating test cases. Each of these implements its generation algorithms. Thus, 24 different test strategies were identified in the 45 tools that generate test cases. The most commonly used strategies are presented in Table 3.6.

Model-based testing (MBT) is an approach to generate test cases using a model of the application under test. In this strategy, (formal) models are used to describe the application's behavior and, thus, generate test cases. Among the identified tools that generate test cases, we observed that 20 of them use the MBT strategy to generate their test cases. The application models mostly describe the application's behavior under test by identifying GUI elements and changing Activity based on these elements.

The second most commonly used generation strategy is GUI Ripping; it has been found 7 tools that implement this strategy. GUI Ripping is a strategy that dynamically traverses an app's GUI and, based on its GUI elements, creates its state-machine model or a Tree Graph.

The third most used strategy by the found tools was the random strategy. A total of 6 tools implements a random strategy. Although it is a less ingenious strategy than the others, some studies point out that it is a very efficient strategy to find crashes [46].

Table 3.6: Types of Generation Strategy

Generation Strategy Approach	Quantity of Tools
MBT	20
GUI Ripping	7
Random	6
Others	14

Regarding the remaining data acquired from the questions presented at the beginning of this section about the characteristics of the tools (use of sensor data, test case generation, test case execution, download availability, and testing approach), table 3.7 summarizes how many tools have each of them.

Table 3.7: Number of tools that presents a given characteristic.

Characteristic	Quantity of Tools
Use of sensor data	9
Test case generation	45
Test case execution	75
Download availability	43
White-Box	14
Gray-Box	7
Black-Box	59

RQ 1.2: What are the most used ones?

Among the studies found by the systematic mapping, Bernardo *et al.* [124] present an investigation on 19 open-source mobile applications for Android to identify how automated tests are employed in practice. They concluded that 47% of these applications have some automated tests, and they observed that the most used testing tools were JUnit, Android. Test, Hamcrest, Robolectric, EasyMock, Robotium, and Fest. Finally, Bernardo *et al.* observed that the most critical challenges in testing Android applications such as rich GUIs, limited resources, and sensors had not been properly handled in the automated tests of the analyzed 19 open- source applications.

Linares-Vásquez *et al.* [84] conducted a survey on 102 Android mobile developers about their practices when performing testing. One of the questions to be answered by the survey was: *"What tools do you use for automated testing?"*. As a result, Linares-Vásquez *et al.* concluded that: *"The most used tool is JUnit (45 participants), followed by Roboelectic with 16 answers, and Robotium with 11 answers. 28 participants explicitly mentioned they had not used any automated tool for testing mobile apps. 39 out of 55 tools were mentioned only by one participant each, which suggests that mobile developers do not use a well-established set of tools for automated testing."*

Villanes *et al.* [133] performed a study using the Stack Overflow[14] to analyze and cluster the main topics of interest on Android testing. One of their results pointed out that recently developers have shown increased interest in the Appium[15], Espresso, Monkey, and Robotium tools.

Bernardo *et al.* did not present a significant amount of applications when compared to Vasquez *et al.* and Villanes *et al.* work. Thus, based on these studies, we can say with greater certainty that the Robotium, JUnit, Roboelectic, Appium, Espresso, and Monkey tools are, according to Vasquez *et al.* and Villanes *et al.*, the most used tools for testing Android applications.

Also, we observed which of the studies identified in the SMS are the most cited in ACM, IEEE, and Google Scholar. Table 3.8 presents the studies and their respective tools that are most cited among the identified studies and tools.

Table 3.8: Most cited study and tools.

		Citations		
Study	Tool	ACM	IEEE	Schoolar
Machiry *et al.* [90]	Dynodroid	119	0	395
Amalfitano *et al.* [30]	AndroidRipper	88	94	356
Anand *et al.* [33]	ACTEve	80	0	292
Azim *et al.* [36]	A3E	88	0	283
Choi *et al.* [45]	SwiftHand	63	0	233
Gomez *et al.* [52]	RERAN	75	60	227
Yang *et al.* [145]	ORBIT	60	0	218
Hao *et al.* [57]	PUMA	69	0	183
Jensen *et al.* [71]	Collider	52	0	155
Amalfitano *et al.* [27]	MobiGUITAR	0	51	148

3.3.2 RQ 2: What are the Android context-aware testing studies and tools published in the literature?

From the 68 selected studies, Griebe and Gruhn [53], Vieira *et al.* [131], and Amalfitano *et al.* [25] explicitly focus on testing of context-aware applications. Sections 3.3.2, 3.3.2

[14]https://stackoverflow.com

[15]http://appium.io

and 3.3.2 discuss the found Android context-aware applications testing tools investigated in these three studies. Besides that, these three studies cite other two tools that also explicitly test context-aware applications: ContextDrive and TestAWARE. These tools are discussed in Sections 3.3.2 and 3.3.2, respectively.

Custom-built version of the Calabash-Android

Griebe and Gruhn [53] propose a model-based approach to improve the testing of context-aware mobile applications. Their approach is based on a four-tier process system as follows:

- Tier 1: UML Activity Diagrams models are enriched with context information using a UML profile developed for integrating context information into UML models;
- Tier 2: Models are then transformed into Petri Nets for analyzing and processing structural model properties (e.g., parallel or cyclic control flows);
- Tier 3: From the Petri Net representation, a platform and technology-independent system testing model is generated that includes context information relevant for the test case execution;
- Tier 4: Platform and technology-specific test cases are generated that can be executed using platform-specific automation technology (e.g., JUnit, calabash-android/ios, Robotium).

To assess the proposed approach, Griebe and Gruhn have extended the Calabash[16] tool to implement it. Calabash is a test automation framework that supports the creation and execution of automated acceptance tests for Android and iOS apps without the necessity of coding skills [59]. It works by enabling automatic UI interactions within an application such as pressing buttons, inputting text, validating responses, and so on.

Calabash is an entirely free and open-source tool. It uses the Gherkin pattern. Gherkin is a writing pattern for an executable specification that, through keywords, maintains a standard for the writing of execution criteria called Given, When and Then. In order to do so, Calabash expresses the test cases as cucumber features [1].

[16]https://github.com/calabash/calabash-android

A limitation of the Griebe and Gruhn approach is the need for creating a model that describes possible AUT activities. Modeling is not a widely understood activity between testers and developers, and a poorly designed model can lead to false positives or false negatives in test verdicts.

Context Simulator

Vieira *et al.* [131] argue that testing context-aware applications in the lab is challenging because of the number of different scenarios and situations that a user might be involved. Hence, the Android platform provides simulation tools to support the physical sensor test. However, it is not enough to test context-aware applications only at the physical sensors level. Thus, Vieira *et al.* have developed a simulator that simulates a real laboratory environment. The simulator provides support for modeling and simulating context in different levels: physical and logical context, situations, and scenarios.

The simulation is separated into two main components: the desktop application and the mobile component:

- The desktop application: it is responsible for context modeling, simulation execution, and context transmission to the mobile device;
- The mobile component: it receives signals from the desktop application and processes the data. The mobile component is responsible for simulating context data and examining the app's reaction under the simulated context.

The modeling in the context simulator is made in four different levels:

1. Low-Level Context: the data can be acquired from hardware sensor measuring (e.g., location, light, movement or touch), named as physical context, or the data can be acquired from software applications or services (e.g., current activity of an employee determined by his calendar), named as virtual context;
2. High-Level Context (or Logical Context): the combination of low-level context and virtual context processing results in a High-level context. For example, a context "Room 001 at Fraunhofer" is identified through an aggregation of two low-level sources: GPS coordinates from "Fraunhofer" and Wi-Fi identification of "Room 001";

3. Situation: it is the composition of high-level contexts. The situation represents the circumstances in which someone currently is. For example, the situation "Meeting 12-13 at Room 001 at Fraunhofer" is a situation composed by three high-level contexts: "Meeting" (it can be a specific date and time plus an appointment in the user's calendar); "Room 001"; and "Fraunhofer".

4. Scenario: a Scenario is a chain of situations for causal relations. In other words, a scenario is a time-ordered sequence of situations.

The context simulator supports a large variety of context sources, 22 contexts divided into 6 categories supporting 41 context sources [131].

A limitation of the context simulator is that the tester must model each test case. Thus, if the tester wishes to test an AUT under possible adverse situations such as weak GPS signal, receiving a phone call, and changing internet connection conditions, then the tester should model all scenarios that he/she wishes to test.

Extended AndroidRipper

Amalfitano *et al.* [25] analyzed bug reports from open-source applications available at GitHub[17] and Google Code[18]. From the results, they defined some use scenarios, called by them as Event-patterns, that represent a use case which presents more potential to failure in context-aware applications. Some examples of event-patterns are:

- Loss and successive recovery of GPS signal while walking;
- Network instability;
- The user enables the GPS provider through the Settings menu and starts walking;
- Incoming of a phone call after any other event.

Amalfitano *et al.* carried out an experiment to examine if the event-patterns represent scenarios of a greater chance of context-aware application failures. Thus, they extended the tool AndroidRipper [30]. The Extended AndroidRipper can fire context events such as

[17]https://github.com

[18]http://code.google.com

location changes, enabling/disabling of GPS, changes in orientation, acceleration changes, reception of text messages and phone calls, shooting of photos with the camera. Both versions of AndroidRipper explore the application under test looking for crashes measuring the obtained code coverage and automatically generating Android JUnit test cases that reproduce the explored executions.

The Extended AndroidRipper tool generates test cases watching for events that cause a reaction from the application. Once the events that cause a reaction are detected, the technique of Amalfitano *et al.* generates test cases based on event-patterns identified by the authors. Therefore, the tool does not focus on testing high-level context variations.

Other Tools from Cited Papers

By studying the papers of Griebe and Gruhn [53], Vieira *et al.* [131], and Amalfitano *et al.* [25], we found two papers related to testing context-aware Android applications: Mirza and Khan [103] and Luo *et al.* [89]. The corresponding tools are presented in the sequel.

ContextDrive Mirza and Khan [103] argues that testing context-aware applications is a difficult task due to challenges such as:

i Developing test adequacy and coverage criteria;

ii Context adaptation;

iii Context data generation;

iv Designing context-aware test cases;

v Developing test oracle;

vi Devising new testing techniques to test context-aware applications.

In response to these challenges, they argue that context adaptation cannot be modeled using a standard notation, such as the UML activity diagram. Therefore, Mirza and Khan extended the UML activity diagram, by adding a context-aware activity node, for behavior modeling of context-aware applications.

Mirza and Khan proposed a test automation framework named as ContextDrive. Its proposed model consists of six phases.

1. First phase: An UML activity diagram is used to model the application under test. In this phase, a new element for the UML activity diagram is proposed for modeling context-aware applications;

2. Second phase: The UML activity diagram is transformed into a testing model;

3. Third phase: The test model is annotated in order to enhance readability and maintainability;

4. Fourth: Abstract test cases are generated;

5. Fifth: Abstract test cases are converted into platform-specific executable test scripts;

6. Sixth: The test scripts are executed.

Mirza and Khan's technique is similar to the one implemented in the tool of Section 3.3.2. Therefore, there is also the restriction that the tester has experience in UML activity diagram modeling. Also, the tool uses static data to execute test cases. Therefore, testing situations that use many sensor data becomes infeasible (i.e., testing a GPS navigator application).

TestAWARE One of the difficulties in testing context-aware applications is the heterogeneity of context information and the difficulty and/or high cost of reproducing contextual settings. As an example, Luo *et al.* [89] present a real-time fall detection application, the application detects when the user drops the mobile phone under different circumstances such as falling out of the pocket or falling out of the hand. The application is programmed to send an email to a caregiver every time a fall event is detected by the phone. For this application, testing new versions of the application is very costly. Thus, Luo *et al.* [89] introduce the TestAWARE tool.

TestAWARE is able to download, replay and emulate contextual data on either physical or emulators devices. In other words, the tool is able to obtain and replay "context", and thus provide a reliable and repeatable setting for testing context-aware applications.

Luo *et al.* compare their tools with other available tools. In summary, they say TestAWARE aims at a wide variety of mobile context-aware applications and testing scenarios. It is possible because TestAWARE incorporates:

i Heterogeneous data (i.e., sensory data, events and audio);

ii Multiple data sources (i.e., online, local and manipulated data);

iii Black-box and white-box testing;

iv Functional/non-functional property examination;

v The environments of device/emulator.

A limitation of the tool is that it is not possible to create test cases without executing each test case at least once in a real device in the real scenario. That is because it is a record and replay tool, it is first necessary to record the test cases and, therefore, it is necessary to submit the AUT on a real device under each of the conditions to be tested.

Potential tools for testing context-aware applications

Among the found studies, Moran *et al.* [106] [107], Z. Qin *et al.* [118], Yongjian and Iulian [65], Gomez *et al.* [52], and Farto *et al.* [50] present tools that were not intended for context-aware application testing. However, they support the testing of context-aware features.

CrashScope Moran *et al.* [106] [107] argue that one of the most challenging and essential maintenance tasks is the creation and resolution of bug reports. For this reason, they introduced the CrashScope tool. The tool is capable of generating augmented crash reports with screenshots, crash reproduction steps, and captured exception stack trace, along with a script to reproduce the crash on a target device. In order to do so, CrashScope explores the application under test by performing input generation by static and dynamic analyses, which include automatic text generation capabilities based on context information such as device orientation, wireless interfaces, and sensors data.

The CrashScope GUI Ripping Engine systematically executes the application under test using various strategies. Then, the tool first checks for contextual features that should be tested according to the exploration strategy. So, the GUI Ripping Engine checks if the current Activity is suitable for exercising a particular contextual feature in adverse conditions. The testing of contextual features in adverse conditions consists in setting unexpected values to the sensors (GPS, Accelerometer, and so on.) that would not typically be possible under

normal conditions. For instance, to test the GPS in an adverse contextual condition, CrashScope sets the value to coordinates that do not represent physical GPS coordinates. Thus, for each running Activity, CrashScope checks the possible contextual features, checks if contextual features should be enabled/disabled, and sets feature values. CrashScope attempts to produce crashes by disabling and enabling sensors as well as sending unexpected (e.g., highly unfeasible) values. Because of that, some scenarios cannot be tested in CrashScope (i.e., testing if the application crashes if the user leaves a pre-established route).

MobiPlay Accordingly to Z. Qin et al. [118], MobiPlay is the first Record and Replay tool that can capture all possible inputs at the application layer. Thus, MobiPlay is the first tool capable of recording and replaying, at the application layer, all the interactions between the Android app and both the user and the environment the mobile phone is inserted.

While the user executes the app, MobiPlay records every input the application receives and the interval time between every two consecutive inputs. After that, the tool can re-execute the application under test with the same provided inputs when executing it. The expected result is that the application behaves the same way as the original execution. MobiPlay is composed of two components: a mobile phone and a remote server. Initially, the mobile phone sends and saves all sensor data and user interactions to the remote server that also stores it. The remote server can then reproduce the executed scenario by sending back the saved data to the mobile phone.

The application under test is called Target App, and it is installed at the Remote Server, not on the mobile phone. The communication between the Mobile Phone and the Target App will be done through the Client App. The Client App is installed at the Mobile Phone, and it is a typical Android app that does not require root privilege and is dedicated to intercepting all the input data for the target app. The basic idea of MobiPlay is that the target app runs on the server, while the user interacts with the client app on the mobile phone in a way that the user is not explicitly aware that he is, in effect, using a thin client. The client app shows the GUI of the target app in real-time on the mobile phone, just like the way as if the target app was running on the mobile phone. While the user interacts with the Target App through the Client App, the server records all the touch screen gestures (pinch, swipe, click, long click, multi touches, and so on) and the other inputs provided by the sensors like gyroscope,

compass, GPS, and so on, in a transparent way to the user. Once the inputs are recorded, MobiPlay can re-execute the Target App with the same inputs and, at the same interval time, simulating the interaction between the user and the target app.

Just like the TestAWARE tool (Section 3.3.2), MobiPlay first needs to record the test cases that the tester wants to check.

VALERA **V**ers**A**tile yet **L**ightweight r**E**cord and **R**eplay for **A**ndroid (VALERA) is a tool capable of record and replay Android Apps by focusing on sensors and event streams, rather than system calls or the stream instruction. Its approach promises to be effective yet lightweight. VALERA can record and replay inputs from the network, GPS, camera, microphone, touchscreen, accelerometer, compass, and other apps via IPC. The authors' main concern is to record and replay Android applications with minimal overhead. Therefore, they claim to be able to maintain performance overhead low, on average 1.01% for record and 1.02% for replay. The timing overhead is critical when replaying an application. The variation of the original time of the application data entries can cause different behavior than when recording the iteration data with the application. Thus, VALERA is designed to minimize timing overhead. In order to evaluate VALERA, the tool was exercised against 50 applications with different sensors. The evaluation consisted in exercising the relevant sensors of each application, e.g., scanning a barcode for the Barcode Scanner, Amazon Mobile and Walmart apps; playing a song externally so that apps Shazam, Tune Wiki, or SoundCloud would attempt to recognize it; driving a car to record a navigation route for Waze, GPSNavig.&Maps, NavFreeUSA; and so on.

VALERA has the same limitations as TestAWARE and MobiPlay.

RERAN It is a black-box record and replay tool capable of capturing the low-level event stream on the phone, including GUI events and sensor events, and replaying it with microsecond accuracy. RERAN is a previous record and replay system of the authors of VALERA. It is similar to VALERA but with some limitations. RERAN is unable to replay sensors whose events are made available to applications through system services rather than through the low-level event interface (e.g., camera and GPS). When validating the tool, the authors declare RERAN was able to record and replay 86 out of the Top-100 Android apps on Google

Play and to reproduce bugs in popular apps, e.g., Firefox, Facebook, Quickoffice.

RERAN has the same limitations as TestAWARE, MobiPlay, and VALERA. Another limitation of RERAN is that it does not support testing the GPS sensor.

MBTS4MA Farto *et al.* [50] proposed an MBT approach for modeling mobile apps. Their test models are reused to reduce the effort on concretization and to verify other characteristics such as device-specific events, unpredictable users' interaction, telephony events for GSM/text messages, and sensors and hardware events.

The approach is based on an MBT process with Event Sequence Graphs (ESGs) models representing the features of a mobile app under test. Specifically, the models are focused on system testing, mainly user's and GUI's events. Farto *et al.* implemented the proposed testing approach in a tool called MBTS4MA (Model-Based Test Suite For Mobile Apps).

MBTS4MA provides a GUI for modeling. Thus, it supports the design of ESG models integrated with the mobile app data like labels, activity names, and general configurations. Although the models are focused on system testing, mainly user's and GUI's events, it is also possible to test sensors and hardware events. The supported sensor events are: change acceleration data, change GPS data, disable Bluetooth, enable Bluetooth, and update coordinates. However, the authors argue that extending the stereotypes of the tool to support more sensor events is possible.

Similar to the Custom-built version of the Calabash-Android tool (Section 3.3.2), MBTS4MA needs the creation of a model representing the features of a mobile app under test.

RQ 2.1: Which research groups are involved in Android context-aware testing research?

In order to answer this research question, we have observed the publications of the authors of the Android context-aware testing studies, such as Griebe and Gruhn [53], Vieira *et al.* [131], and Amalfitano *et al.* [25].

The authors of Griebe and Gruhn [53] are Tobias Griebe[19] and Volker Gruhn[20]. Both

[19]https://dblp.org/pers/hd/g/Griebe:Tobias

[20]https://dblp.uni-trier.de/pers/hd/g/Gruhn:Volker

authors have written only two more publications that refer to context-aware applications:

- "Towards Automated UI-Tests for Sensor-Based Mobile Applications" [54]: presents an approach that integrates sensor information into UI acceptance testing. The approach uses a sensor simulation engine to execute test cases automatically.
- "A Framework for Building and Operating Context-Aware Mobile Applications" [123]: presents a work-in-progress paper with the description of a framework architecture design to address the following context-aware mobile applications problems: interoperability, dynamic adaptability and context handling in a frequently changing environment.

The authors of Vieira *et al.* [131] are Vaninha Vieira[21], Konstantin Holl[22], and Michael Hassel[23]. Vaninha Vieira is a professor of Computer Science at Federal University of Bahia, Brazil. Her research interests include Context-Aware Computing, Mobile and Ubiquitous Computing, Collaborative Systems and Crowdsourcing, Gamification and User Engagement, and Smart Cities (Crisis and Emergency Management, Intelligent Transportation Systems). Among her publications, we can note the interest in mobile applications concerning to context modeling, Quality Assurance, Context-Sensitive Systems Development, Context Management, and so on. Konstantin Holl has published papers related to Quality Assurance, but nothing can be seen about the research interest of Michael Hassel due to the lack of publications.

The authors of Amalfitano *et al.* [25] are Domenico Amalfitano[24], Anna Rita Fasolino[25], Porfirio Tramontana[26], and Nicola Amatucci[27]. Domenico Amalfitano, Anna Rita Fasolino, and Porfirio Tramontana are not only professors of the same institution (University of Naples Federico II) but most of their articles were written together. Their publication concerns software engineering, testing, and reverse engineering. Many of the testing publications are about Android app testing. In particular, they have much experience in the GUI Ripping

[21] https://scholar.google.com/citations?user=tkNSlXIAAAAJ&hl
[22] https://dblp.org/pers/hd/h/Holl:Konstantin
[23] https://dblp.org/pers/hd/h/Hassel:Michael
[24] https://scholar.google.com/citations?user=ReafO6YAAAAJ&hl=en
[25] https://scholar.google.com/citations?user=lC5j76YAAAAJ&hl=es
[26] https://scholar.google.com/citations?user=Q7z44GcAAAAJ&hl=en
[27] https://scholar.google.it/citations?user=AaLCcaAAAAAJ&hl=it

technique. Most of Nicola Amatucci's publications are about testing on Android applications. Most of them written together with Domenico Amalfitano, Anna Rita Fasolino, or Porfirio Tramontana.

All of these authors have significant publications regarding mobile application testing. Besides them, as mentioned in Section 3.2, we can refer to Iulian Neamtiu, Tanzirul Azim, and Yongjian Hu, who have significant contributions in the research area. However, among the studied authors, Vaninha Vieira is the author who most directly contributed to the research on context-aware applications.

RQ 2.2: What are the research gaps addressed in Android context-aware testing?

In this systematic mapping, we identify 5 tools for testing of context-aware Android applications and 5 tools that support testing of context-aware applications, totaling 10 tools.

The context-aware application testing has challenges such as a wide variety of context data types and context variation. There is a vast variety of context data types. The most commonly used context data type is location, acquired by the GPS sensor. However, there are many other types of data, such as temperature, orientation, brightness, time, and date.

Context-aware applications use context data provided by sensors to provide service or information. Waze[28], for example, uses the GPS, the time, and information provided by the cloud to inform the driver about obstacles along the way to the final destination. However, many context-aware applications use combinations of sensor information to infer contexts and provide service or information from these inferred contexts. Vieira *et al.* [131] call Low-Level Context the context information directly collected from sensors or other sources of information such as database or cloud, and High-Level Context for the contexts that are the product of the combination of Low-Level Contexts.

Many context-aware applications use High-Level Context to provide their services or information. Samsung has developed an application called Samsung Health [9] that tracks user's physical activities. Combined with its SmartWatch, the application monitors heartbeat, movement, steps, geographical location, time of day, and other information. From this information, the application infers contexts in which the user is and then concludes whether the user is practicing physical activity or resting. Taking the example of the Samsung Health

[28]https://www.waze.com

application, Table 3.9 exposes some examples of High-Level Contexts from the composition of Low-Level Contexts.

Table 3.9: High-Level Contexts examples

Brief Description	Low-Level Contexts	High-Level Context
If the user has stopped for more than one hour since it is not at night, the application infers that the user is at rest for a long time and suggests that the user take a short walk or lengthen.	Time GPS Pedometer Accelerometer Heartbeat	Long rest
If the user is in full rest with low heart rate, the application infers that the user is sleeping and counts the duration of sleep as well as infers the quality of sleep based on the luminance, noise and amount of movements that the user makes while sleeping.	Time GPS Pedometer Accelerometer Brightness Noise Heartbeat	Sleeping
If the user is walking, the application monitors the distance and speed. From this information and the user's profile (weight and age) the application infers the amount of lost calories.	GPS Pedometer Age Weight	Walking
If the user is pedaling, the application infers that the user is riding a bike and then calculates the time, distance and lost calories.	GPS Pedometer Accelerometer Noise Heartbeat	Riding a bike

As we have said, another challenge in testing context-aware applications is the constant variation of context. The context changes asynchronously, and the application must respond

correctly and effectively to context variations. Taking Samsung Health as an example, the application must realize when the user is changing their activities throughout the day and thus provides all the information and services in the correct way. Thus, when the user is sleeping and get up, the application should stop counting the time and the quality of sleep. If the user starts walking, the application must count for time, distance, and lost calories. When the user stops walking and gets in the car and drives home, the application should stop counting the walking information and understand that the user is resting, even though he is moving.

Considering the difficulties of testing context-aware applications, the 10 tools identified in this work were analyzed and compared according to 11 questions raised:

- Q1: What low-level context data does the tool support?
- Q2: Does the tool support high-level context data?
- Q3: Are context data treated differently?
- Q4: Is it possible to test context variations?
- Q5: Is it possible to test abnormal context situations?
- Q6: What criteria is used to select the context data?
- Q7: What is the test stop criterion?
- Q8: Does the tool generate test cases?
- Q9: Is the tool White Box, Black Box, or Grey Box?
- Q10: Does it need instrumentation in the code?
- Q11: Is the tool automatic or semi-automatic?

Table 3.10 presents the result of the analysis of the 10 tools by looking at the 11 questions.

Table 3.10: Tools Comparison

	Tool									
	Custom calabash [53]	Context Simulator [131]	Extended AndroidRipper [25]	CrashScope [106]	MobiPlay [118] [78]	VALERA [65] [63] [64] [66] [78]	RERAN [52] [78]	MBTS4MA [50]	ContextDrive [103]	TestAWARE [89]
Q1										
GPS	X	X	X	X	X	X		X	X	X
Wi-Fi	X	X		X		X		X	X	X
Accelerometer	X	X		X	X	X	X	X	X	X
Thermometer	X	X	X	X				X	X	X
Barometer		X	X		X			X	X	X
Light-Sensor	X	X			X		X		X	X
Magnetometer	X	X	X	X	X	X	X		X	X
Gyroscope		X			X			X	X	X
Clock		X							X	
Calendar		X							X	
Other		Camera, Microphone, Battery Level, Call, text message, Alarm, etc.	Call, text message, Battery Level, USB, etc.			Microphone		Bluetooth, Call, text message		
Q2	No	Yes	No	No	No	No	No	No	Yes	No
Q3	Yes	Yes	No	No	No	No	No	No	Yes	Yes
Q4	Yes	Yes	No	No	No	No	No	No	Yes	No
Q5	Yes	Yes	No	Yes	No	No	No	No	Yes	Yes
Q6	Manually	Manually	Design patterns	On/Off or Abnormal values	None	None	None	None	None	Manually or Recorded from Sensor
Q7	all-transition-coverage criterion	All scenarios executed	Code coverage	Top-down or botton-up GUI Hierarchy transverse	No more recorded events left	No more recorded events left	No more recorded events left	all-edges	breadth first search	No more recorded events left
Q8	Yes	No	Yes	Yes	No	No	No	Yes	Yes	No
Q9	Black-box	Black-box	White-box	Black-box	Black-box	Black-box	Black-box	Black-box	Black-box	Black-box and White-box
Q10	No	No	No	No	No	No	No	No	None	No
Q11	Automatic	Semi-automatic	Automatic	Automatic	Semi-automatic	Semi-automatic	Semi-automatic	Semi-automatic	Automatic	Semi-automatic

The first observation we had of the tools was on the type of context data they support. Except for RERAN, they all support GPS. It was natural to expect this result since location is the most commonly used data type by context-aware applications. We can also see that Context Simulator and ContextDrive are the only tools that support all low-level context data types. Also, these are the only tools that support high-level context data.

Mirza and Khan [103] propose an extension of the UML activity diagram for modeling high-level context variation. Thus, their ContextDrive tool can test variations from one context to another. The authors use static data to execute the test cases. Therefore, the tool is unable to generate new test cases automatically.

The Context Simulator tool provides a graphical interface for the tester that enables the creation of application usage scenarios. Thus, the tester can simulate high-level contexts. However, the tester needs to explicitly describe each test case he/she wants to execute and

which sensor values will be used in the test.

Context variations occur asynchronously, and some of them in an unexpected way. When using a context-aware application, a phone call can be received, and, during the calling, the user context may change. As another example, the GPS signal can drop and then return after a few moments.

Although 3 tools support context variation testing, none of them can automatically generate test cases that use high-level contexts and test variations of high-level contexts, taking into account unexpected scenarios such as the event-patterns described by Amalfitano *et al.* [25], presented in Section 3.3.2.

3.4 Concluding Remarks

In this chapter, a systematic mapping was carried out in order to identify and investigate tools that allow the automation of testing Android context-aware applications. A total of 6,648 studies were obtained, 68 of which were considered as relevant publications when taking into account our research questions. These works were first analyzed according to the conference publication, year, country, and authors. The main result of this first analysis was the identification of research groups in the area of interest.

Another significant contribution of this systematic mapping was the identification of 80 Android application testing tools. From these tools, we identified which techniques they implement, which generate test cases, which execute test cases, which are the test methods, which ones are available for download, and which ones are most commonly used. We noticed that 40% of Android testing tools implement GUI Testing. We also note that, among the tools that generate test cases, 42% use MBT as the generation strategy approach.

The main contribution and objective of this systematic mapping were identifying context-aware Android application testing tools and the analysis of their limitations. We have identified 10 tools that support the test of context-aware applications. Five of these tools have been developed explicitly for context-aware applications, and five have been developed for general applications, but they also support context-aware features testing.

Chapter 4

Context-Aware Path-Based Testing Approach

Mobile applications are applications that respond to User Interface (UI), system events, or sensor events. Events in mobile applications are touches on the screen, updating the geographic location, changing the orientation of the device, and so on. Therefore, many authors (if not all) recognize that mobile applications are event-driven [119], [38], [25], [121], [128]. Each event produced in Android applications brings information (message) regarding this event. Thus, how to test if a context-aware path-based application has faults? How to ensure that it behaves correctly with different context configurations, that is, with different sequences of events?

This chapter describes the approach proposed by our work. Section 4.1 outlines the proposed approach. Section 4.2 describes how test cases are generated while section 4.3 describes how the test cases are executed.

4.1 Overview

Context-aware applications react to inputs from the environment in which it is inserted. Therefore, in this work, an approach is proposed whose objective is to simulate environmental stimuli. The approach considers the environment as the source of all events that serve as input to the AUT. Regarding context-aware path-based applications, its environment information is mainly composed by the GPS values captured by the GPS sensor and the events

that can occur during its execution.

The approach consists of providing a sequence of inputs (path positions and events) to the AUT and observing if the application executes correctly in the same way as if it were running in a real-world environment even if events occur during execution. The approach proposed in this work does not require code and is intended to test executable versions of context-aware path-based applications. Therefore, the proposed approach is black-box and tests applications at the system level.

Considering:

- lat a real value representing a latitude coordinate;
- lng a real value representing a longitude coordinate;
- P a GPS position composed by a latitude and a longitude in the form (lat, lng);
- n an integer such that $n \geq 0$.

We define a path as following.

Definition 4.1 (Path)**.** *A Path is a sequence of GPS positions such that:* $Path = (P_o, P_1, P_2, ..., P_n, P_d)$ *where:*

- P_o *represents the path origin;*
- $P_1, P_2, ..., P_n$ *represents a sequence of zero or more points* P*;*
- P_d *represents the path destiny;*

Amalfitano *et al.* [25] present a set of events that they called "Event-patterns". These event patterns are sequences of events that, according to the authors, are most likely to cause failure. Some of the event patterns defined by Amalfitano *et al.* are:

- Drop and successive recovery of GPS signal while walking;
- Internet instability;
- Phone call arrival after any event except the call event itself.

Observation 1: After looking for bugs reported by users of open source applications in Github [11], we identified other scenarios with a chance of failure.

In order to identify bugs that can occur in real context-aware path-based applications, we search for problems reported in open-source applications hosted on GitHub [11]. Github is a collaborative platform where programmers and users of published applications can report tips and bugs that occurred during the software's execution.

Therefore, we perform searches initially for path-based applications. Due to the small number of applications and few reported bugs in the path-based applications found, we search for applications that use GPS, either to read only the current location or to perform GPS tracking. Therefore, we selected the following applications:

- OsmAnd[1];
- NextGIS[2];
- Sky Map[3];
- CityZen[4];
- GPSLogger[5];
- Traccer[6].

For each selected issue, we observed its recurrence among the listed applications and categorized in scenarios. We observed that some scenarios resort to problems well known in the literature such as boundary-value analysis and others fall into common problems in Android development such as the management of an Activity's life cycle. Together, these applications add up to more than 3,200 issues of the most varied types of bugs.

The identified scenarios are:

[1]https://github.com/osmandapp
[2]https://github.com/nextgis
[3]https://github.com/sky-map-team
[4]https://github.com/CityZenApp
[5]https://github.com/mendhak/gpslogger
[6]https://github.com/traccar/traccar

- Paths with coordinate limit values: Faults have been reported on some routes with values close to the coordinate limit values, for example: (37.9872 -0.65483), (-37.9872 0.65483), (0.00001 0), (0 -0.000001);
- Short paths;
- Long paths;
- Leave the application on standby for a long time;
- Take a photo while running the application;
- Change phone orientation “portrait” <–> “landscape”.

Based on the work of Amalfitano *et al.* [25] and in our search for bugs events in Github [11], our approach takes into account the following events:

i Turn on GPS;

ii Turn off GPS;

iii GPS calibrated;

iv GPS not calibrated;

v Simulate background application for a long time;

vi Receive phone call;

vii Accept phone call;

viii Cancel phone call;

ix Internet with full speed;

x No internet signal;

xi Portrait orientation;

xii Landscape orientation;

xiii Take a picture.

In our context, a test case is executed when a sequence of inputs is sent to the AUT. We differentiate these inputs into two types:

- GPS values: GPS coordinates of a path;
- Events: system events from the Android system in response to situations that occur in the environment, such as phone calls and internet crash.

Therefore, a test case input is a sequence of inputs and system events sent to the AUT.

4.2 Test case generation

Test cases consist of a setup and a sequence of events. Setup determines the initial environment configuration in the test case. Considering the exposed events in Section 4.1, path type, long background, take a photo, and phone calls do not make sense to be part of the test setup configuration. Thus, a setup is defined by combining four situations: (i) GPS signal, (ii) GPS calibration, (iii) orientation, and (iv) Internet Connection. Table 4.1 presents the possible values for each one of the four situations.

Table 4.1: Possible situation values.

Event	GPS Signal	GPS Calibration	Orientation	Internet
Values	On	Calibrated	Portrait	On
	Off	Not Calibrated	Landscape	Off

Before starting the test cases, it is crucial to define which setup to be used. Since there are 4 variables that can assume 2 values, we have 16 setup combinations (2^4 combinations), but considering that GPS cannot be calibrated when it is turned off, then 12 setup combinations are possible, with scenarios are presented in Table 4.2.

Table 4.2: Setups considered in this work.

Name	GPS Signal	Orientation	GPS Calibration	Internet Signal
S1	ON	PORTRAIT	CALIBRATED	ON
S2	ON	PORTRAIT	CALIBRATED	OFF
S3	ON	PORTRAIT	NOT CALIBRATED	ON
S4	ON	PORTRAIT	NOT CALIBRATED	OFF
S5	OFF	PORTRAIT	NOT CALIBRATED	ON
S6	OFF	PORTRAIT	NOT CALIBRATED	OFF
S7	ON	LANDSCAPE	CALIBRATED	ON
S8	ON	LANDSCAPE	CALIBRATED	OFF
S9	ON	LANDSCAPE	NOT CALIBRATED	OFF
S10	ON	LANDSCAPE	NOT CALIBRATED	ON
S11	OFF	LANDSCAPE	NOT CALIBRATED	OFF
S12	OFF	LANDSCAPE	NOT CALIBRATED	ON

In our approach, events may occur immediately or after some time interval. The sequence of events may have one or more events; all events may or may not have a time interval between them.

We call as "waiting situation" the condition of an event to occur immediately or after some time interval. In other words, assume that a test case has the events E_1 and E_2. AUT can receive these events in the following ways:

1. Upon starting the test case, E_1 is immediately sent to the AUT, and E_2 is sent to the AUT immediately after E_1;

2. Upon starting the test case, E_1 is sent to the AUT after slapsed a time t, and E_2 is sent to the AUT immediately after E_1;

3. Upon starting the test case, E_1 is immediately sent to the AUT, and E_2 is sent to the AUT after t has elapsed after E_1;

4. Upon starting the test case, E_1 is sent to the AUT after slapsed a time t, and E_2 is sent to the AUT after t has elapsed after E_1.

Therefore, we call as "waiting situation" the possibility of existing a waiting time $t \neq 0$ or $t = 0$ before the E_1 and E_2 events.

Table 4.3 present the events considered in this work.

Table 4.3: Possible situation values.

GPS	CALLS	INTERNET	DISPLAY ORIENTATION	OTHER
GPS_ON	RECEIVE_CALL	INTERNET_ON	ORIENTATION_PORTRAIT	TAKE_A_PICTURE
GPS_OFF	ACCEPT_CALL	INTERNET_OFF	ORIENTATION_LANDSCAPE	LONG_BACKGROUND
GPS_CALIBRATED	CANCEL_CALL			
GPS_NOT_CALIBRATED				

Considering:

- S as one of the possible setups in Table 4.2;
- E_i as one of the possible events in Table 4.3;
- W_i as one of two possible waiting situations such that $W_i \in$ {WAIT, NOT_WAIT}.

We can define test case and subtest case according to Definitions 4.2 and 4.3.

Definition 4.2 (Test Case)**.** *A Test Case (TC) defines the AUT setup and a sequence of pairs of waiting situations W_i and events E_i to be sent to the AUT such that:*
$TC = (S, W_1, E_1, W_2, E_2, \ldots, W_n, E_n)$ *where n is the order of the test case TC.*

Definition 4.3 (Subtest Case)**.** *Considering TC_n a test case of order n that has a Setup S and the sequence of waiting situations and events $(W_1, E_1, ..., W_n, E_n)$, we say that TC_n is subtest case of TC_N if (i) TC_N has order N such that $N > n$, (ii) TC_N has Setup S and (iii) its sequence of waiting situations and events starts with $(W_1, E_1, ..., W_n, E_n)$.*

In this work, we call first-order test cases the test cases with only one wait and one event. For two-wait and two-event test cases, we call them second-order test cases, and so on. Table 4.4 illustrates the possibilities of wait and event values for test cases of n-order for each set of sensor values.

Table 4.4: Possibilities of waits and events for n-order test cases.

	Setup	Wait 1	Event 1	Wait 2	Event 2	...	Wait n	Event n
Sensors Values	S1	WAIT	GPS_ON	WAIT	GPS_ON	...	WAIT	GPS_ON
	S2	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_OFF	...	NOT_WAIT	GPS_OFF
	S3		LONG_BACKGROUND		LONG_BACKGROUND			LONG_BACKGROUND
	S4		TAKE_A_PICTURE		TAKE_A_PICTURE			TAKE_A_PICTURE
	S5		ORIENTATION_PORTRAIT		ORIENTATION_PORTRAIT			ORIENTATION_PORTRAIT
	S6		ORIENTATION_LANDSCAPE		ORIENTATION_LANDSCAPE			ORIENTATION_LANDSCAPE
	S7		RECEIVE_CALL		RECEIVE_CALL			RECEIVE_CALL
	S8		GPS_CALIBRATED		ACCEPT_CALL			ACCEPT_CALL
	S9		GPS_NOT_CALIBRATED		CANCEL_CALL			CANCEL_CALL
	S10		INTERNET_ON		GPS_CALIBRATED			GPS_CALIBRATED
	S11		INTERNET_OFF		GPS_NOT_CALIBRATED			GPS_NOT_CALIBRATED
	S12				INTERNET_ON			INTERNET_ON
					INTERNET_OFF			INTERNET_OFF

In first-order test cases, we suppress ACCEPT_CALL and RECEIVE_CALL events because it is not possible to answer or reject a call without first receiving the call. Therefore, the number of possible combinations for each set of sensor data for test cases of order 1 is $12 \times 2 \times 11 = 264$. The number of order 2 test case combinations for each sensor data is $12 \times 2 \times 11 \times 2 \times 13 = 6,864$. So, the number of test cases of order n is $12 \times 2 \times 11 \times (2 \times 13)^{(n-1)}$.

Observation 2: Pairwise testing significantly decreases the number of test cases.

The number of test case combinations is impractical to execute. Pairwise testing has been shown to be very efficient by some authors [47], [42], [87]. In addition to reducing the number of exhaustive combinations for pairing, we can also exclude meaningless test cases, by explicitly specifying combinations that are not valid. Some test scenarios make no sense when executing; for example:

- Turn on GPS when it is already on;
- Turn off GPS when it is already off;
- GPS gets calibrated when it is off;
- Turn on the Internet when it is already on;
- Turn off the Internet when it is already off;
- Answer a non received call;

- Reject a non received call
- Change the device orientation to the same orientation as it is.

Table 4.5 shows the difference in the number of test cases generated with and without pairwise considering the list of events in Table 4.4. We can see that the number of test cases without pairwise increases exponentially as the order increases according to the formula $qte = 12 \times 2 \times 11 \times (2 \times 13)^{(n-1)}$. However, the same is not valid for the number of test cases generated with pairwise. The number of test cases using pairwise decreases considerably by: (i) the pairwise technique, and (ii) by eliminating combinations of sequences of events that do not make sense (i.e. turning on GPS already on or disconnecting a phone call that did not occur). Thus, we can create viable combinations of executions for order greater than 5.

Table 4.5: Number of Test Cases

	Order 1	Order 2	Order 3	Order 4	Order 5
Without pairwise	264	6,864	178,464	4,640,064	120,641,664
With pairwise	81	178	223	270	283

Although pairwise is very effective at reducing the number of test cases, generating the minimum set of test cases using pairwise is an NP-Complete problem [147]. Also, while the number of test cases can be reduced from hundreds of millions to hundreds of units, manual execution is still very costly. Therefore, it is necessary to automate the generation process and, at least, semi-automate the test case execution process. Chapter 5 describes a tool developed to automate the generation and semi-automate the execution of test cases using this proposed approach.

4.3 Test case execution

In our approach, AUT execution occurs by sending two types of input: a sequence of sensor input values and events most likely to cause failures (Section 4.1). Therefore, the execution must take place in an emulated environment to allow the sending of the input values of the sensors as well as allowing a controlled execution at a low cost.

Before executing each test case, the AUT must be configured according to the test case setup. After configuring the setup, the AUT must receive each input value from each sensor. Meanwhile, events are sent to the AUT. According to Definition 4.2, there is a waiting situation before any event. Therefore, if the test case establishes that there must be a wait before the event E, then event E must occur after a predetermined time interval. Otherwise, event E occurs immediately after its predecessor or the setup, whether E is the first event. The test case ends when there are no more sensor values to send to the AUT, and when there are no more events to send.

The tester observes the AUT during the entire execution of the test case. He assigns the fail verdict if any behavior occurs that does not conform to the specifications during the test case execution. That is, if during the execution of the test case the application exhibits some behavior that it shouldn't or shows no behavior, the tester judges that that test case failed. For example, if the AUT stops responding to inputs or adds an intermediate destination on a route without the user having requested it, the tester will judge this behavior as faulty. If the test case ends without any failure occurring, the tester assigns the pass verdict.

The verdict is made in parallel with the execution. The tester must observe the AUT's behavior during the execution of the test cases and judge the verdict of each execution. As the approach proposed in this work generates and executes test cases automatically, the cost of the test falls on the tester's observation and evaluation. That is, the longer it takes to run the entire test suite, the greater the tester's effort to observe and judge all verdicts.

4.4 Concluding Remarks

This chapter described the approach proposed by our work. The approach allows the generation of test cases for context-aware path-based Android applications assuming that these applications are event-driven. Thus, the approach proposes the creation of sequences of inputs that simulate the sending of data from the GPS sensor as well as system events with a greater chance of causing failures.

The test case generation assumes that a test case is composed of a setup configuration and a sequence of system events that occur during the arrival of inputs from the sensors. The number of possible test cases grows exponentially as the number of system events (test case

order) increases. Therefore, we employ the pairwise technique to reduce the number of test cases without significantly losing the quality of the final test suite.

Chapter 5

The ENVIAR Tool

In this work, we developed the ENVIAR (**ENV**Ironment d**A**ta simulato**R**) tool that implements the approach described in Chapter 4. The tool can simulate the environment in which the AUT is running by sending mock data to the AUT. Section 5.1 gives an overview of the tool. The tool's architecture is explained in Section 5.2. Section 5.3 outlines the Testing Process performed in ENVIAR. Section 5.4 explains how ENVIAR uses the PICT tool in order to perform the test case generation. The ENVIAR's graphical interface is exposed in Section 5.5.

5.1 Overview

The Android development environment makes it possible to execute applications on real devices or emulated devices. In this work, to reduce the cost of testing and make viable the execution of challenging scenarios of manual execution, the execution of the applications is performed in an emulated way. Besides that, simulating sensor values on real devices requires modifying the Android operating system. Thus, we decided to use the Android Emulator that facilitates GPS simulation without the need to modify the Android operating system.

ENVIAR is a free tool capable of simulating the sending of application environment data, system events, and simulating adverse situations (i.e., GPS not calibrated) available for download at https://github.com/diegotabira/enviar. Using the ENVIAR tool, it is possible to create test scenarios that would be difficult to perform manually. Besides, it is possible

to create situations during a route such as an internet connection changing. For example, consider that a team is developing a GPS navigation application that uses data provided over the internet. It is possible to test the application manually by turning off the internet phone connection. However, if the route is from one city to another, this test may not be feasible. ENVIAR can simulate any route in the Android emulator and simulate events like the example above. Also, ENVIAR does not require any changes to the Android operating system, does not require code instrumentation, nor does the AUT source code. ENVIAR only needs the tester to create the test cases (Section 5.5) and opens the AUT in the functionality he wants to test. For example, if the tester wants to check an application's GPS navigator, he must open the application and make it ready to use the GPS navigation functionality.

ENVIAR is capable of automatically detecting crashes in the AUTs. Still, it requires the tester to observe the AUT's behavior to assign the pass or fail verdicts. An AUT crashes when a failure occurs and makes the application unable to recover, so the Android operating system needs to terminate the application. Therefore, there are three possible verdicts in ENVIAR: pass, fail, and crash.

In this work, we consider that a test case passes when the AUT correctly navigates through the path. In other words, independently of what events happen, the AUT is robust enough to recover (when possible) and continue the navigation correctly after all events happen in the test case. In this case, we assign the *pass* verdict. For example, the AUT must be able to continue navigation if the GPS is switched off and then switched on again. For obvious reasons, we consider that the application passed the test case if the GPS is turned off, but it is not turned on again. We consider a test case to fail when the AUT does not recover from the events in the test case or if the AUT's behavior does not match the specifications. In this case, we assign the *fail* verdict. Some failures make the application unresponsive. That is, there are failures that when they occur, the running application no longer responds to inputs coming from the user or the system forcing the Android operating system to close it. Many authors call this type of failure as crash [106], [107], [46], [72] and [143]. In our work, we attribute the *crash* verdict to this kind of special case of *fail* verdict.

Given the scenarios and events described in Section 4.1 that are more likely to fail, the tool supports sending the following events:

- Turn on GPS;

- Turn off GPS;
- GPS calibrated;
- GPS not calibrated;
- Simulate background application for a long time;
- Receive phone call;
- Accept phone call;
- Cancel phone call;
- Internet with full speed;
- No internet signal;
- Portrait orientation;
- Landscape orientation;
- Take a picture.

5.2 Architecture

Figure 5.1 illustrates the basic architecture of ENVIAR. The black components are third-party solutions that we fully incorporate into our tool with no modifications. Gray components are those components in which we use the third-party solution internally and increment algorithms. White components are those that we implement entirely without adding third-party solutions.

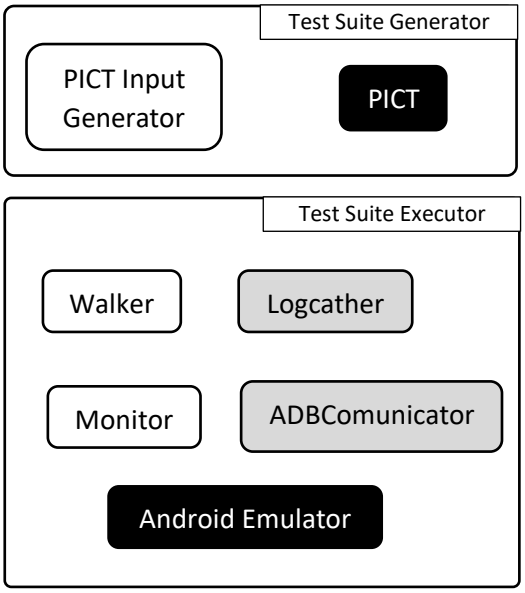


Figure 5.1: ENVIAR Architecture

The following components compose the tool:

- Test Suite Generator: Responsible for generating the test cases. Basically composed of:
  - PICT: PICT [15] is a command-line tool developed by Microsoft [16] that receives as input a plain-text model file and produces a set of test cases selected by pairwise testing. More details about PICT is described in Section 5.4;
  - PICT Input Generator: As mentioned, PICT needs a plain-text model file as input. This plain-text model file contains rules and descriptions of what parameters will be used to execute the pairwise testing. The PICT Input Generator is responsible for creating the plain-text model file from the paths, events, speeds, and order chosen by the tester. The plain-text model file will be further detailed in Section 5.4;
- Test Suite Executor: Responsible for executing the test cases. Basically composed of:
  - Walker: This module is responsible for sending the geographic coordinates of the path and the system events to the AUT;

- Monitor: Module responsible for observing if an AUT crashes. This module also notifies the Test Suite Executor about the verdicts associated by the tester, so the current test case execution ends, and the next test case can be executed;
- Logcatcher: Logcat [12] is a command-line tool developed by Google that logs system messages, crash traces, and Log class messages produced by application developers. The Logcatcher module encapsulates the use of Logcat to facilitate the identification of failures or crashes produced by AUT;
- Android Emulator: The Android Emulator [17] is a tool developed by Google that simulates Android Devices in the computer making possible the application testing without the need for the real device. ENVIAR uses the Android Emulator tool to execute the AUT test cases in a simulated way;
- ADBComunicator: The Android Debug Bridge (ADB) [13] is a command-line tool developed by Google that communicates with the Android Emulator. The ADBComunicator module encapsulates all the necessary commands to facilitate communication between ENVIAR and the Android emulator through ADB. Besides, ADBComunicator is also responsible for storing all commands sent to the emulator during the execution of a test case to facilitate the later reproduction of scenarios that have caused the failure.

5.3 Testing Process

The ENVIAR testing process initiates with the tester informing which path(s) to test and the order of the test (first-order, second-order, and so on). By default, ENVIAR generates tests for all the system events described in Section 4. From these inputs, the PICT Input Generator creates the plain-text model file (PICT Input). The Test Suite Generator calls the PICT passing the PICT Input, which, consequently, generates a file containing the combinations of paths and events following pairwise testing. The test suites in ENVIAR are stored internally in the output format of the PICT tool.

The Test Suite Executor module receives the test cases that are the result of the PICT execution. Thus, for each test case, the Test Suite Executor configures the Android emulator according to the test case setup configuration and executes the test case. On execution, the

Walker component simulates (in a new thread) the variation of the geographical position of the path(s) while the ADBComunicator sends the events of the test case. At the end of the execution of each test case, the Test Suite Executor generates three outputs:

i Verdict of the test case;

ii File containing all sent commands to AUT during the execution of the test case;

iii File with the execution logcat of the test case.

Figure 5.2 summarises the ENVIAR testing process.

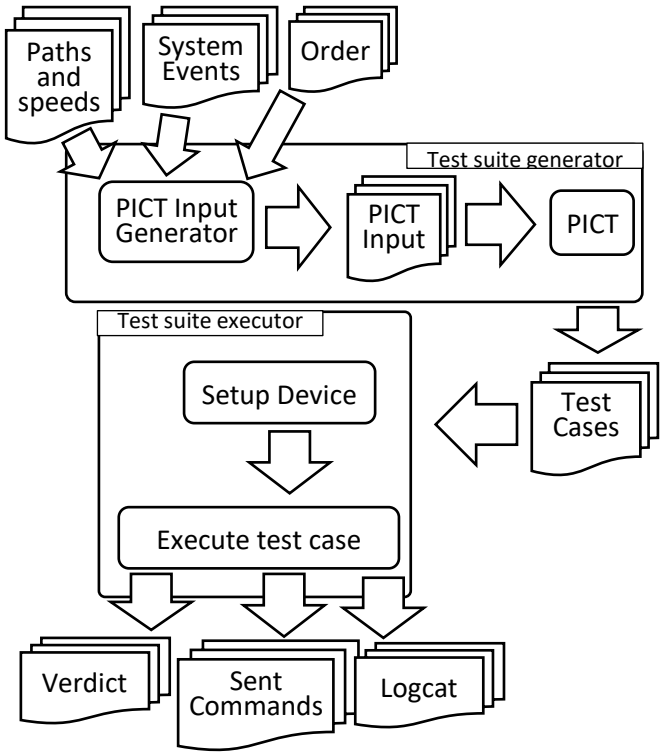


Figure 5.2: Enviar Testing Process

The pseudo-code 5.1 describes in more details the ENVIAR testing process.

Source Code 5.1: ENVIAR Testing Process Pseudo-code

```
1 ExecuteENVIAR(){
2    Set<Tuple[Path, speed]> pathsAndSpeeds = Tester.getPathsAndSpeeds();
3    int testOrder = Tester.getTestOrder();
4    Set<SystemEvents> systemEvents = ENVIAR.getSystemEvents();
```

```
5    Set<TestCase> testCases = generateTestCases(pathsAndSpeeds,
         systemEvents, testOrder);
6    executeTestSuite(testCases);
7  }
8
9  generateTestCases(Set<Tuple[Path, speed]> pathsAndSpeeds, Set<
       SystemEvents> systemEvents, int testOrder){
10   TextFile PICTInput = PICTInputGenerator.generatePictInput(
         pathsAndSpeeds, systemEvents, testOrder);
11   TextFile PICTOutput = PICT.executePairwise(PICTInput);
12   Set<TestCase> testCases = ENVIAR.extractTestCases(PICTOutput);
13   return testCases;
14 }
15
16 executeTestSuite(Set<TestCase> testCases){
17   AndroidEmulator.start();
18   Tester.prepareAUT();
19   Monitor.startMonitoring();
20   foreach(TestCase testCase : testCases){
21     setupTestCase(testCase);
22     executeTestCase(testCase);
23   }
24 }
25
26 executeTestCase(TestCase testCase){
27   Walker.simulateWalk(testCase.getPath());
28   while(NOT Walker.finished() AND NOT Tester.verdictSetted()) {
29     if(testCase.hasWait(){
30       ENVIAR.wait();
31     }
32     if(testCase.hasNextSystemEvent()){
33       ADBComunicator.sendEvent(testCase.getNextSystemEvent());
34     }
35     Tester.verifyAUTBehavior();
36   }
37   save(verdict);
38   save(ADBComunicator.getSentCommands());
```

```
39     save(Logcather.getAUTLogcat());
40 }
```

Initially, ENVIAR receives from the tester a set of tuples composed by the path and respective maximum speeds and the order of the test case (lines 2 and 3). In line 4, ENVIAR loads all system events supported by the tool. The generateTestCases subroutine generates the test cases for the entered parameters in line 5. In line 6, ENVIAR passes the test suite to the executeTestSuite subroutine.

To generate the test cases, the generateTestCases subroutine initially generates the plain-text model file (PICT Input) through the PICTInputGenerator component (line 10). This file is passed as a parameter to the PICT component, which executes pairwise to select the set of test cases (line 11). In line 12, the test cases are extracted from the PICT output.

The test suite is executed by the executeTestSuite subroutine (line 16). Whenever the test suite starts to run, the Android Emulator starts (line 17), the tester "prepares" the AUT (line 18), that is, the tester opens the AUT and leaves the application ready to test the functionality of interest. In our case, the navigation functionality along the simulated path. In line 19, The Monitor module starts monitoring. It will check if the AUT crashes or if the tester associates any verdict during the execution. For each test case (line 20), ENVIAR performs the test case setup (line 21) and executes the test case (line 22).

To execute a test case, the Walker component initially starts its simulation of geographic location variation for the path points (line 27). This walking simulation is an asynchronous function, that is, as soon as the Walker component starts executing the walking, ENVIAR continues its execution and goes to line 28. As long as the Walker has not finished simulating the walking and the tester has not assigned a verdict (line 28), ENVIAR waits the pre-defined time of 15 seconds (lines 29 and 30) if the test case has a "wait" condition (Section 4.2) and then sends the next system event if there is any system event yet to be sent (lines 32 and 33). During the walking and sending events, the user observes the AUT to see if any incorrect behavior occurs (line 35). At the end of the test case execution, ENVIAR stores the verdict, the sent commands, and the execution logcat (lines 37, 38, and 39).

5.4 Test Case Generation Using PICT

ENVIAR implements the test case generation approach described in Chapter 4. As described in Section 4.2, the selection of test cases is done by pairwise technique. The ENVIAR tool implements the technique using the PICT tool [15]. PICT is a command-line tool developed by Microsoft [16]. PICT receives as input a plain-text model file and produces a set of test cases. The model file consists of the following sections:

- parameter definitions
- sub-model definitions (optional and not used by our work)
- constraint definitions (optional and used by our work)

Each parameter definition is listed in a separate line with the respective values delimited by commas:

<ParamName>: <Value1>, <Value2>, <Value3>, ...

Example:

Path: Perfect, Small, Long, Limit
Speed: 80, 5, 200
Setup: S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12
First_Delay: WAIT, NOT_WAIT
First_Event: GPS_ON, GPS_OFF, SIMULATE_LONG_BACKGROUND, TAKE_A_PICTURE, ORIENTATION_PORTRAIT, ORIENTATION_LANDSCAPE, RECEIVE_CALL, GPS_CALIBRATED, GPS_NOT_CALIBRATED, INTERNET_ON, INTERNET_OFF

The PICT tool makes possible the creation of constraints definitions to avoid unwanted combinations from the final combination result or to specify limitations on the domain. Considering the given example, if we want to:

- limit the speed of the Perfect and Limit paths to 80 km/h;
- limit the speed of the small path to 5 km/h;
- limit the speed of the long path to 200 km/h;
- avoid the first event from being GPS_ON in setups that start with GPS on;
- avoid the first event from being GPS_OFF in setups that start with GPS off.

The model file should be:

```
Path: Perfect, Small, Long, Limit
Speed: 80, 5, 200
Setup: S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12
First_Delay: WAIT, NOT_WAIT
First_Event: GPS_ON, GPS_OFF, SIMULATE_LONG_BACKGROUND, TAKE_A_PICTURE, ORIENTATION_PORTRAIT, ORIENTATION_LANDSCAPE, RECEIVE_CALL, GPS_CALIBRATED, GPS_NOT_CALIBRATED, INTERNET_ON, INTERNET_OFF
IF [Path] = "Perfect" THEN [Speed] = 80;
IF [Path] = "Limit" THEN [Speed] = 80;
IF [Path] = "Small" THEN [Speed] = 5;
IF [Path] = "Long" THEN [Speed] = 200;
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} THEN [First_Event] <>"GPS_ON";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <>"GPS_OFF";
```

When creating the PICT input model file, the definitions of the parameters are determined by choosing which paths the tester wants to take, the maximum speed on each path, and the order of the test suite. ENVIAR generates the constraint definitions considering the test order and the following restrictions:

- GPS cannot be turned on while it is already turned on;

- GPS cannot be turned off while it is already turned off;
- Cannot calibrate an already calibrated GPS;
- It is not possible to make not calibrated an already not calibrated GPS;
- It is not possible to change the orientation of the device to the orientation it is already in;
- Internet connection cannot be turned on while it is already turned on;
- Internet connection cannot be turned off while it is already turned off;
- A call that did not occur cannot be accepted;
- A call that did not occur cannot be canceled.

To generate test cases, ENVIAR creates the PICT input model file internally and then executes the PICT tool by passing the created model file as input to the tool. The rules used for test cases of order 1, 2, 3, 4 and 5 are respectively A.1, A.2, A.3, A.4 and A.5.

5.5 ENVIAR Graphical Interface

Figure 5.3 illustrates the main ENVIAR window. It is divided into two parts: "Test Suite Generation" and "Test Suite Execution". To generate test cases, it is necessary to choose which path(s) to test. It is possible to choose more than one path by holding down the keyboard CTRL or SHIFT button and clicking which paths to generate test cases. Four paths are pre-installed in ENVIAR (Figure 5.4). As explained in Section 4.1, paths with geographic coordinate values close to the limit values (Figure 5.4(a)), long paths (Figure 5.4(b)) and short paths (Figure 5.4(d)) can be faulty paths. The perfect path (Figure 5.4(c)) is a straight path that does not fit into any of the types of paths that can cause failure. It was used to generate test cases whose possible failure is not linked to the path, but to the system events.

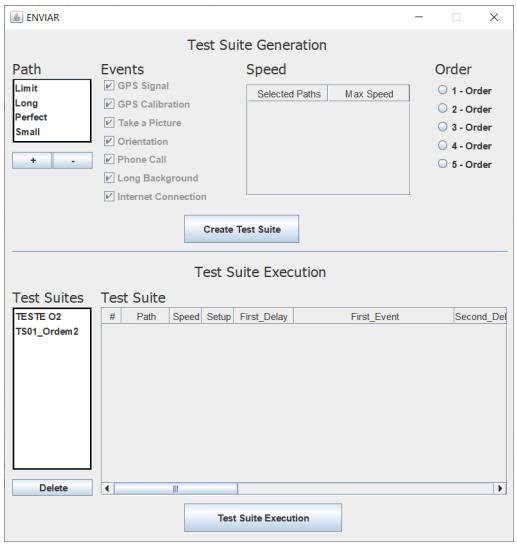
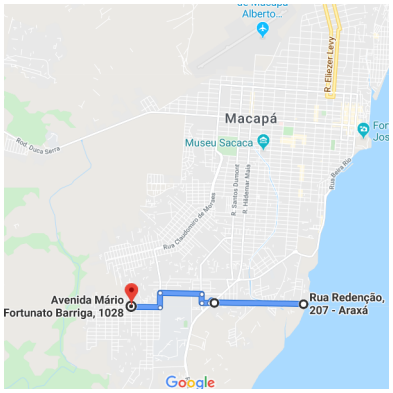
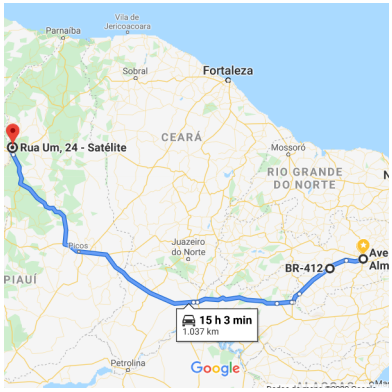


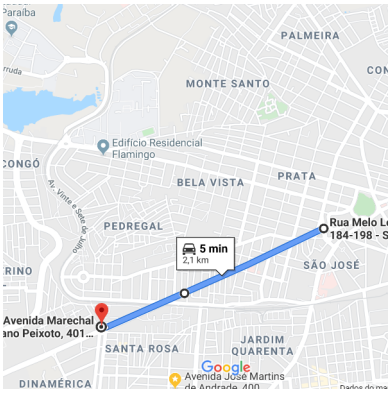
Figure 5.3: ENVIAR Main Window



(a) Limit



(b) Long



(c) Perfect

(d) Small

Figure 5.4: Paths

The “+” and “-” buttons are respectively for creating or deleting paths. As can be seen in

Figure 5.5, to create a path, it is necessary to enter the points to go through.

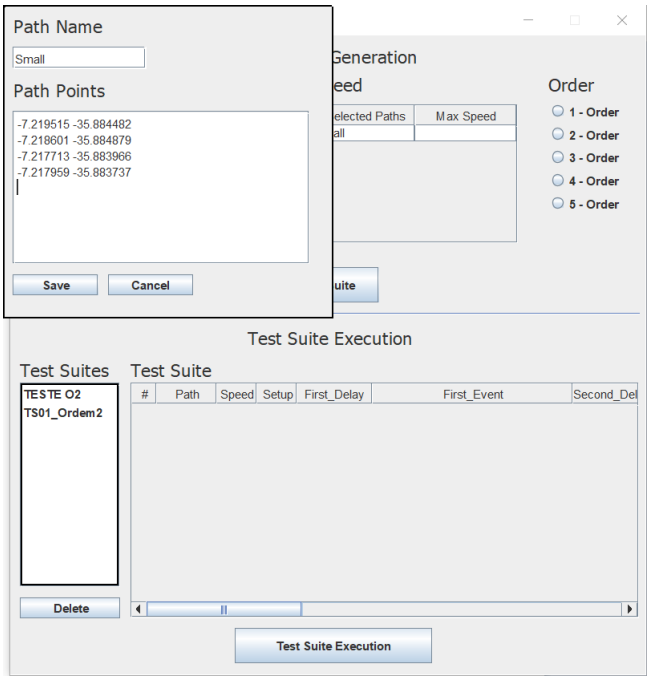


Figure 5.5: ENVIAR Path Creation

The tester must set which maximum speed (in km/h) will be emulated in each of the chosen paths (Figure 5.6) and which order of the test suite to generate.

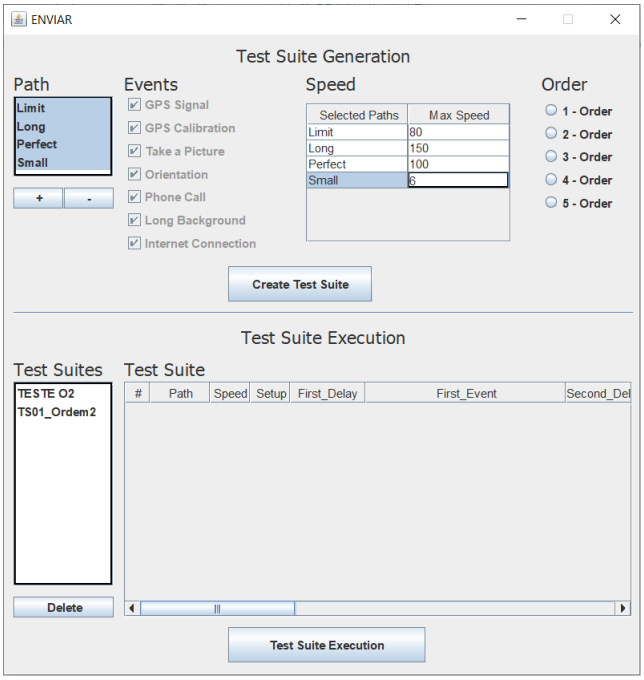


Figure 5.6: ENVIAR Speed Setting

The generated test suites are available to execute in the test suite list, Figure 5.7. To execute a test suite, click on one of the generated suites and click the "Test Suite Execution" button. By pressing the button, it will open the execution window, Figure 5.8.

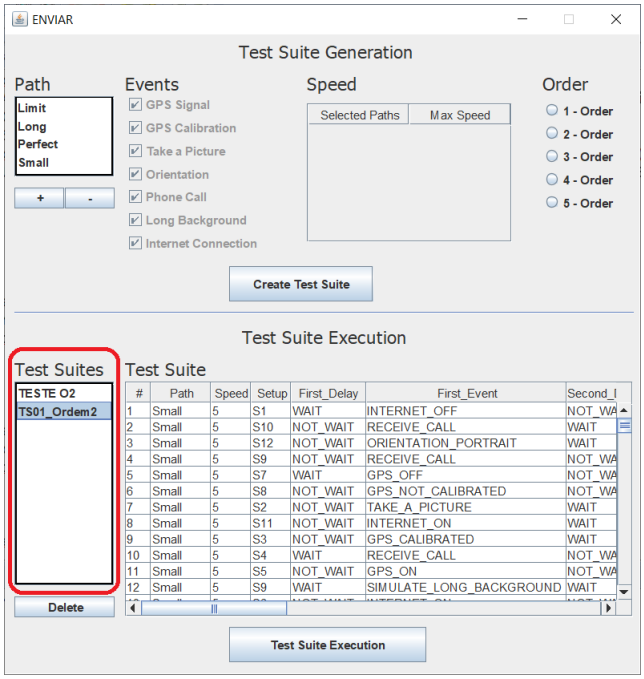


Figure 5.7: ENVIAR Execution

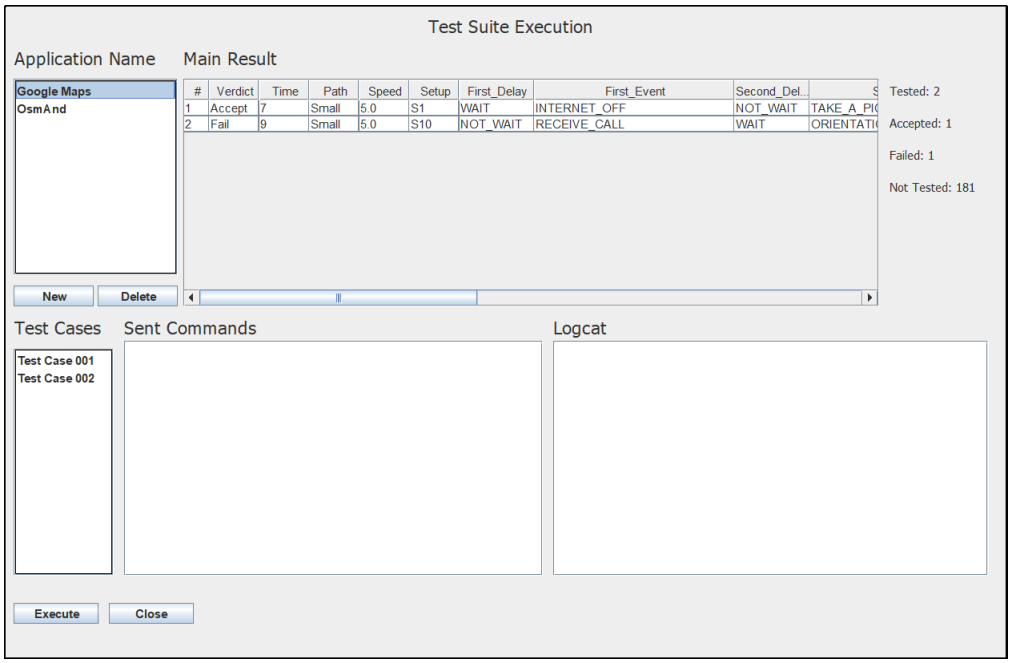


Figure 5.8: ENVIAR Execution Window

Information about executions is arranged in the execution window. The information is separated by AUT organized in the application list, Figure 5.9. The tester should add the name and package of the application to be tested via the "New" button. For each added application, information about the number of test cases tested, the number of test cases accepted, rejected, and not yet tested is displayed.

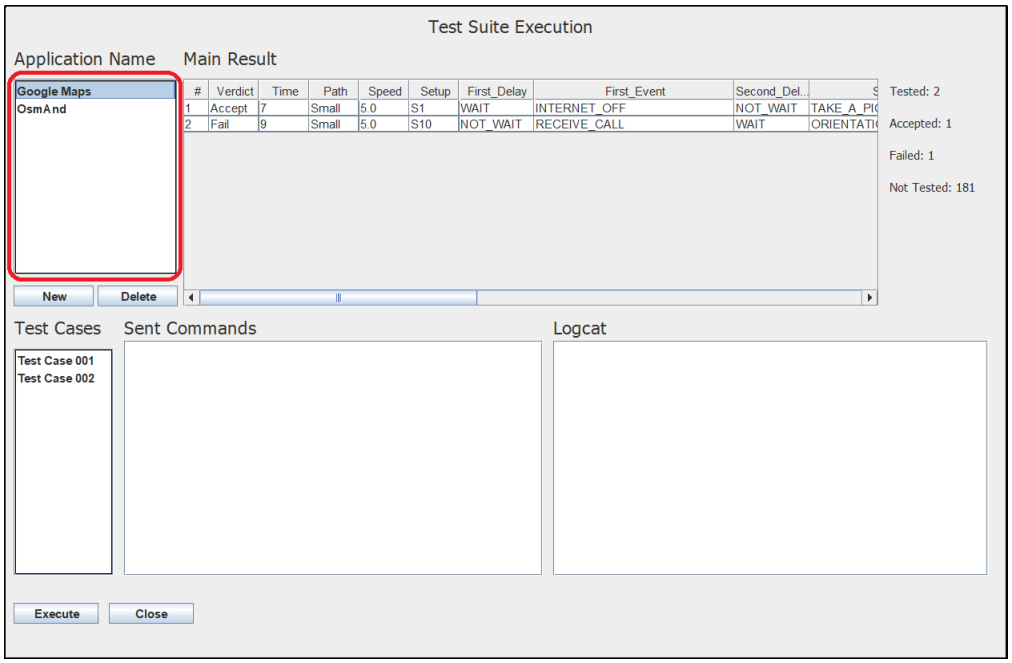


Figure 5.9: ENVIAR Applications Under Test

After running the test case, the tool generates two outputs: logcat and sentCommands.

- Logcat: ENVIAR retrieves the emulator-generated logcat from the time the test case execution starts until it ends. In Table 5.1 is a part of a logcat generated by running a crashed application. Note that logcat informs precisely which part of the code caused the failure, making it easier for the developer to identify the reason.

- sentCommands: All commands sent by the tool to the AUT are saved in this output. Thus, it is possible to rerun the same test case under the same conditions, enabling the re-execution of failed scenarios. In Table 5.2 is a part of a generated sentCommands. In it, it is possible to see that initially, the internet was at full speed. After traversing two geographical points, the AUT was opened. After traversing 4 geographical positions,

the internet connection was interrupted. As can be seen, all data simulated by ENVIAR is sent to AUT via the Android Debug Bridge (ADB) [13].

Table 5.1: Logcat example

——— beginning of crash
08-21 14:50:23.700 9280 13480 E AndroidRuntime: FATAL EXCEPTION: AsyncTask #11
08-21 14:50:23.700 9280 13480 E AndroidRuntime: Process: com.voicenavigation.gps.driving.directions, PID: 9280
08-21 14:50:23.700 9280 13480 E AndroidRuntime: java.lang.RuntimeException:
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at android.os.AsyncTask$3.done(AsyncTask.java:353)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at java.util.concurrent.FutureTask.finishCompletion(FutureTask.java:383)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at java.util.concurrent.FutureTask.setException(FutureTask.java:252)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at java.util.concurrent.FutureTask.run(FutureTask.java:271)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at android.os.AsyncTask$SerialExecutor1.run(AsyncTask.java:245)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1162)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:636)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at java.lang.Thread.run(Thread.java:764)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: Caused by: java.lang.IndexOutOfBoundsException: Index: 0, Size: 0
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at java.util.ArrayList.remove(ArrayList.java:503)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at c.d.e.a.a.a.g.l.doInBackground(:1)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at android.os.AsyncTask$2.call(AsyncTask.java:333)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: at java.util.concurrent.FutureTask.run(FutureTask.java:266)
08-21 14:50:23.700 9280 13480 E AndroidRuntime: ... 4 more

Table 5.2: Sent commands example

adb emu network speed full
adb emu geo fix -37.541069 -7.589196
adb emu geo fix -35.884482 -7.219515
adb shell monkey -p com.voicenavigation.gps.driving.directions -c android.intent.category.LAUNCHER 1
adb emu geo fix -37.541069 -7.589196
adb emu geo fix -37.541069 -7.589196
adb emu geo fix -37.541069 -7.589196
adb emu geo fix -37.541069 -7.589196
adb emu network speed 1 1
adb emu geo fix -37.541069 -7.589196
adb emu geo fix -37.541069 -7.589196
adb emu geo fix -37.541069 -7.589196

Clicking on one of the executed test cases (Figure 5.10) displays information about sent commands and logcat. Sent commands and logcat are also displayed in real-time while the test case is running.

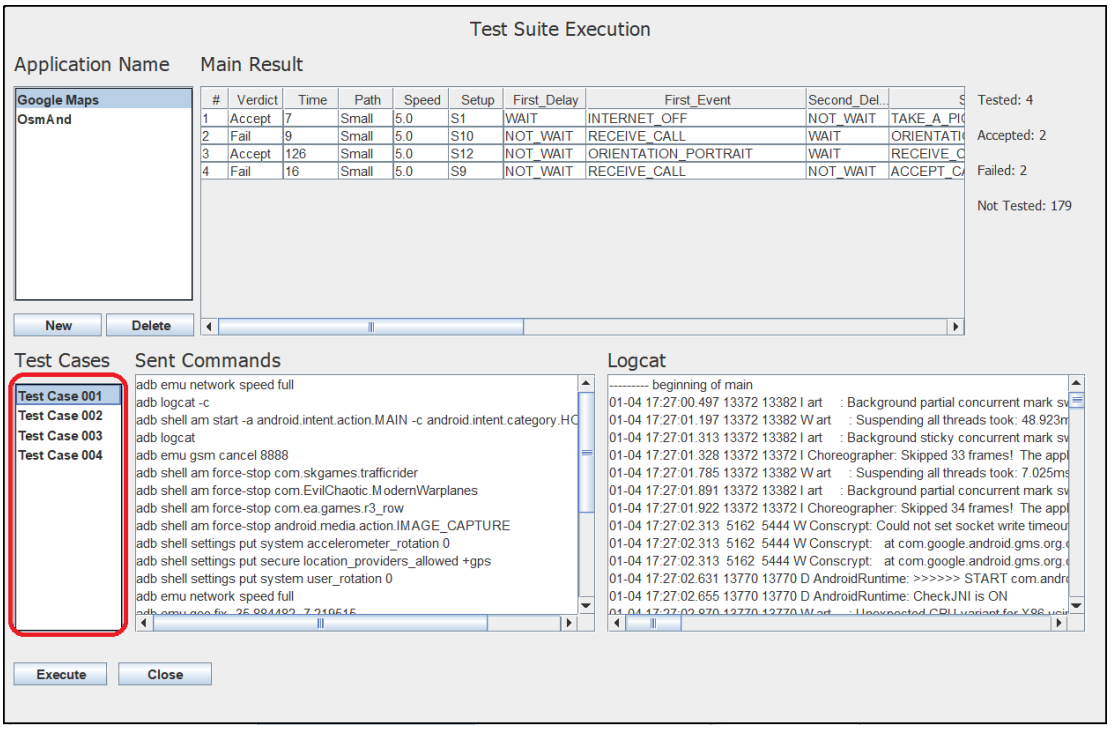


Figure 5.10: Test Cases

By clicking the “Execute” button, ENVIAR will check if there are any Android Emulator (emulator for short) running. If not, ENVIAR will ask to choose one of the installed emulators to run. As explained in Section 4.2, each test case has one of 12 Table Setups 4.2. Therefore, the first step of running each test case is the AUT setup, Figure 5.11.

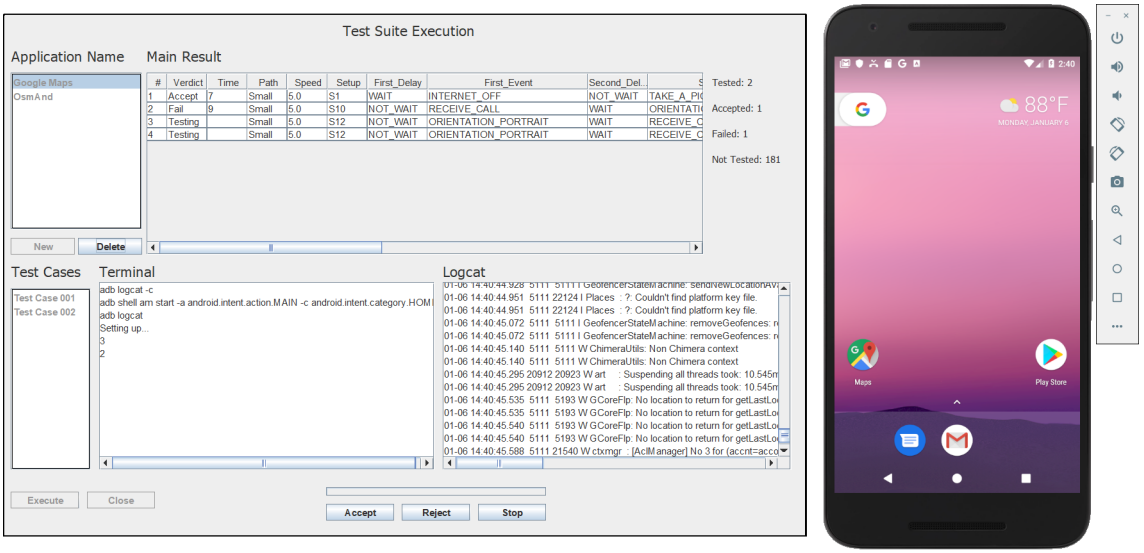


Figure 5.11: Test Cases Setup

After setup, ENVIAR will start sending the geographical coordinates of the path. During path creation (Figure 5.5), the tester describes the geographic coordinates through where the simulator should travel. Based on these coordinates and the maximum velocity that the tester sets during the creation of the test suite (Figure 5.6), ENVIAR calculates intermediate points between the coordinates setted by the tester on path creation so that the speed never exceed the maximum.

ENVIAR automatically identifies AUT crashes and allows the tester to assign verdicts during test case execution, Figure 5.12. If during the execution of a test case, the AUT crashes, then ENVIAR determines the Crash verdict and stops the execution so the tester can prepare the AUT again for testing. If the tester observes any abnormal behavior during the execution of the test case, he/she may press the "Reject" button. Thus, ENVIAR will determine the Reject verdict for that test case and then start the next test case. The Accept verdict can be given to the test case if it ends execution without the tester having determined the Reject verdict or if the tester presses the "Accept" button. The tester may press the "Stop" button while the test case is running if it wishes to stop the test without giving any verdict on the test case execution.

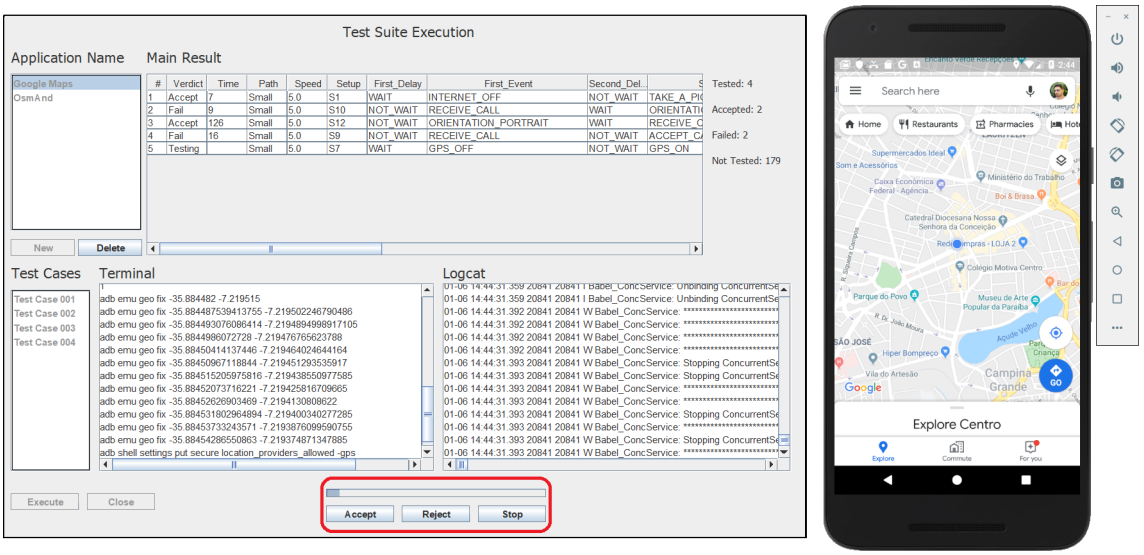


Figure 5.12: Verdicts Buttons

5.6 Concluding Remarks

This chapter described the implementation of the approach proposed in Chapter 4. ENVIAR is a tool capable of generating and executing test cases for context-aware path-based Android applications simulating variation of GPS location and system events that can cause failures. The ENVIAR architecture, its components, and the implementation of the generation and execution of test cases were described in this chapter. The ENVIAR graphical interface was also described, allowing a more detailed comprehension of the tool.

Chapter 6

Evaluation

In this chapter, we show an evaluation of the approach proposed in Chapter 4. The evaluation was carried out through an exploratory study realized to identify which kind of defects the approach can find and features on test case orders, Section 6.1. We compared ENVIAR with other existing tools in Section 6.2. Section 6.3 highlights how ENVIAR can be used as a support tool to assist other existing tools for testing context-aware path-based applications.

6.1 Exploratory Study

The goal of this study, according to the description in the GQM template, was to analyze the **defects found** with the intention of **evaluating them** with respect to their **nature and quantity** from the **tester's** point of view in the context of **GPS based Android applications**.

Hence, the research questions in this study are as follows:

- **RQ1:** What kind of defects can the approach reveal?
- **RQ2:** Are any of the defects found contextual?
- **RQ3:** Is there a difference between test case orders?

We use as a metric to answer the questions the number of context defects found by ENVIAR.

Software testing is part of the software development process, and its main objective is to find defects (bugs) in the source code to be corrected. This activity can be done statically

(through code analysis) or dynamically (through the execution of test cases). However, in both cases, it is necessary to perform the debugging process to identify defects.

In our study, we do not have access to the source code of the AUTs. Therefore, instead of investigating defects in the code, we observed possible defects from failures during the execution of the test cases. That is, we identified possible defects from scenarios that caused failures.

Therefore, the dynamics performed to define the possible defects were:

1. We observed the execution of each test case;
2. We documented the observed behavior of each test case that failed;
3. At the end of the test suite execution, we grouped the test cases that failed to match by AUT and observed behavior;
4. Observing the logcat saved by the ENVIAR tool and the events sent to the AUT during the test case execution, we reorganized the test cases observing cause/effect relationships.

Thus, for the same test cases that failed under the same scenario, we assumed that the same defect in the code possibly caused it. Hence, in the evaluation, we did not look at the number of test cases that failed, but at the number of possible different defects found. From now on, we will call simply as ***defect*** each possible defect in the source code.

The next subsection describes how the exploratory study was carried out.

6.1.1 Methodology

Figure 6.1 illustrates how the exploratory study was planned, executed, and how its results were analyzed. The flowchart is composed of activities (represented by rectangles with rounded corners), flow (represented by arrows), and artifacts (represented by several documents picture).

Initially, we selected the applications used to evaluate our approach, resulting in 4 real GPS navigation applications (Subsection 6.1.1). In parallel, we carried out a study of a set of test cases we would use to execute in each one of the selected applications in our approach

and which GPS data to use (Subsection 6.1.1). We observe generation and execution time in each of the possible orders of test cases and select the paths to be emulated observing the types of paths most likely to fail (Section 4.1).

The execution time did not take into account the time for the setup. Setup can take 10 to 60 seconds. When an AUT is opened, the tester must inform the path it will take and then start the test case in ENVIAR. In this case, the tester takes about 60 seconds to prepare the AUT to execute the test cases. If the test case does not crash, the application does not need to be started again, so ENVIAR only needs about 10 seconds to perform the setup and start the execution of the next test case.

With the selected applications, the choice of test orders, and GPS data, we carried out the planning of the execution of the exploratory study. The approach proposed by our work was implemented in the ENVIAR tool, and through it, we executed the exploratory study.

At the end of the execution, the ENVIAR tool saved files containing the verdict, execution time, and sent events from each test case. We call this set of saved files as raw results. The raw results contain no analysis of defects or whether the failures occurred due to context variation.

The ENVIAR tool does not automatically evaluate verdict in test cases. ENVIAR automatically detects crashes in the AUT, but it cannot automatically determine if a test case has passed or failed. Therefore, the execution is monitored by a tester who observes the AUT's behavior and associates the pass and fail verdicts for the test case.

After the execution of the test cases, we performed the analysis of the raw results and distinguished the defects found and cataloged them as context defects and non-context defects. A defect is a context defect when the failure of the test case was due to a context variation (i.e., failure due to a change in GPS calibration or a change in the internet connection). We identify context defects by looking at whether the context variation of the test case caused the failure. This analysis was performed after running all the test cases in the AUTs. The following subsections explain each of the activities in Figure 6.1 flowchart.

Applications Selection

We chose four path-based applications widely used for the execution of the exploratory study, as shown in Table 6.1.

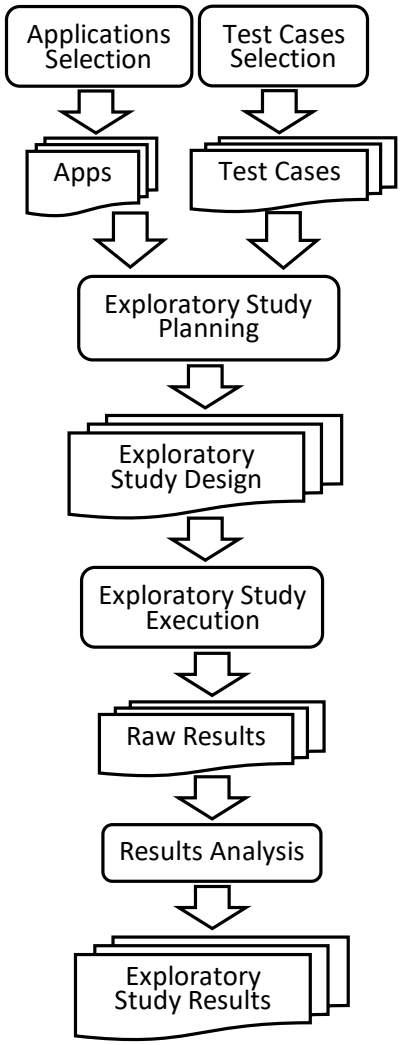


Figure 6.1: Exploratory Study Flowchart

Table 6.1: Selected Applications

Application	**Installations**	**Stars**	**Version**
OsmAnd [6]	5,000,000+	4.7	3.3.8
Genius Maps: Offline GPS Navigator [4]	500,000+	4.2	2.5.0
GPS Offline Navigation Route Maps & Directions [5]	100,000+	4.0	1.0.3
Voice GPS Navigation: Live Driving Direction [7]	10,000+	4.2	1.13

All applications are similar in the service they offer, GPS navigation applications. All applications are available for download from the Google Play web store [14].

It was chosen real applications in order to avoid any bias by the authors. Also, it was chosen applications with few downloads (more than 10,000) and many downloads (more

than 5,000,000). The number of downloads indirectly influences the quality of the software. Google Play provides a platform for users to report their impressions about the apps. Therefore, they also report bugs they identify throughout use. For this reason, applications with many downloads tend to have fewer bugs than applications with few downloads, as it is natural that there will be more posts about presented bugs.

Test Cases Selection

Given the approach defined in Chapter 4, we can automatically generate test cases for applications. ENVIAR generates and executes test cases receiving as input the route the tester wishes to travel and the events the tester wants to occur during the path travel. The approach proposed in our work aims to identify context defects by sending events and/or as a result of one of the paths with the highest chance of failure (Section 4.1). Thus, we define a path that we call the "Perfect Path" where we hope that failures will not occur due to the path execution itself. As explained in Section 4.1, we identified scenarios with a chance of failure, from where, there are bug reports about GPS based applications that indicate failures in executing short paths, long paths, and paths with coordinate limit values. Therefore, we made the "Perfect Path" as simple as possible to prevent the application from failing due to the path being executed. Thus, the "Perfect Path" is a straight line on which the application will travel without leaving the route. We have defined this path so that if it fails, it will most likely be due to events.

We generated test cases using four different paths:

- Limit Path (Figure 5.4(a)): a path that goes over the equator line;
- Long Path (Figure 5.4(b)): a path with more than 1.000 kilometers;
- Perfect Path (Figure 5.4(c)): a straight line on which the application will travel without leaving the route;
- Small Path (Figure 5.4(d)): a path with less than 350 meters.

As explained in Chapter 5, ENVIAR is capable of generating test cases of order 1, 2, 3, 4, and 5. We generate test cases of order 1, 2, 3, 4, and 5 and calculate the number of possible test cases without pairwise. Table 6.2 shows the difference in the number of

test cases with and without pairwise in each order. We can see that the number of test cases without pairwise increases exponentially as the order of the test case increases since, without pairwise, all possible combinations of events and paths in the test cases are generated.

Table 6.2: Test Case Orders Comparison

	Order 1	Order 2	Order 3	Order 4	Order 5
Without pairwise	1,056	27,456	713,856	18,560,256	482,566,656
With pairwise	88	183	223	255	284

Each test case in Table 6.2 is composed of (i) a path that will be sent to the AUT, (ii) the maximum speed that will be simulated, (iii) a setup that will be configured when starting the test case, and (iv) a sequence of N waits and events that will be dispatched for N-order test cases (see Section 4.2). The execution of the test case only ends after all events and after walking all the route.

It is possible to calculate an estimate of how long a test case can take to execute. In our work, we used the speed of 5 km/h to the small path, 80 km/h to the perfect path, 80 km/h to the limit path, and 1000 km/h to the long path. We chose the speed of 5 km/h for a small path to simulate a person walking. The speed of 80 km/h was chosen because it is an average maximum speed widely used on highways in Brazil. The speed of 1000 km/h was chosen for the long path to reduce the execution time of the test case and, at the same time, test the AUTs at speed limit values. After several executions of test cases with each of the types of paths with the mentioned speeds, we were able to conclude that:

- Test cases that have a small path can take up to 312 seconds to execute completely;
- Test cases that have a perfect path can take up to 156 seconds to execute completely;
- Test cases that have a limit path can take up to 404 seconds to execute completely;
- Test cases that have a long path can take up to 6134 seconds to execute completely;

This time that we measure is the maximum time that each test case can take. In these executions, the test cases were run until the end of each path. In other words, the time may be shorter because the test case may fail at some point in the path and, consequently, it will take less time to execute it from the beginning of the path to the point that failed.

Considering:

- Q_{Sm} as the number of test cases with small path;
- Q_{Pf} the number of test cases with perfect path;
- Q_{Lm} the number of test cases with limit path;
- Q_{Lg} the number of test cases with long path.

We can estimate the time in hours of the execution of a test suite using the formula $Time = (Q_{Sm} \times 312 + Q_{Pf} \times 156 + Q_{Lm} \times 404 + Q_{Lg} \times 6134) \div 3600$.

After generating the test cases of order 1, 2, 3, 4, and 5, we count the number of test cases with each kind of path for each order. Also, we measure how long it takes the tool to generate the test cases for each of the orders. Table 6.3 illustrates how long the test cases for each order take to generate and how long it can take to execute on each of the AUTs when executing on the machine described in the context selection (subsection 6.1.1).

Table 6.3: Generation and Execution Time

	Order 1	Order 2	Order 3	Order 4	Order 5
Generation Time	<1 second	<1 second	00:00:01	00:01:33	04:20:06
Execution Time Estimate	54:09:54	120:44:58	133:26:54	146:15:26	183:47:18

This increase is because pairwise testing is an NP-Complete problem. For example, to generate the order 5 test cases, PICT needs to generate the 482,566,656 test cases internally to select 284. It was necessary 04 hours, 20 minutes, and 06 seconds to generate the test cases of order 5. It is important to remember that the generation and execution times of the Table 6.3 can vary depending on the used computer hardware. However, regardless of the power of the hardware, the difference in generation times between the orders will still be exponential.

On the other hand, the execution time does not increase exponentially because the number of test cases with pairwise does not increase exponentially yet. The execution time was calculated by observing the time necessary to execute the entire test case route considering that the test case only ends at the end of the path simulation. Thus, as test cases can fail, the total execution time may be less than that calculated in Table 6.3.

The generation time for test cases of orders 1, 2, 3, and 4 is low. The test cases of order 5 took 04:20:06 to be generated. Although it is a long time compared to the generation time for the 1, 2, 3, and 4 test cases, it is not a long time compared to the execution time.

In our approach, test cases are generated automatically and its execution is semi-automated once the analysis of the execution is done manually. That is, a tester must observe the execution of the test case in order to assign a verdict. Thus, the execution time is a limitation in this study since we only have a single tester to observe the execution of all test cases in each of the four applications. The tester monitors the execution of the test case. If he perceives any behavior that is not consistent with the specifications, then the tester assigns the failure verdict and notes the visualized behavior. At the end of all test case executions, the tester observes in each test case the verdict, the path traveled, and the events received and, based on this information and notes, catalogs defects found. Thus, in this study, we only consider test cases of order 1 and order 2.

The generation of the test cases for both orders takes only 2 seconds. However, the execution time takes approximately 54 and 121 hours in each of the four selected applications, totaling up to 700 hours of execution in all four applications (without considering the data analysis time). A study of all five orders would take approximately 2,548 hours of execution in the four selected applications, which is not feasible in this study's reality. To make this study's execution feasible, we decided to generate test cases of orders 1 and 2 only. Therefore, 88 test cases of order 1 (Table B.1) and 183 test cases of order 2 (Table B.2) were generated to be used to execute each one of the selected applications.

Exploratory Study Planning

With the selected applications, the choice of test orders, and GPS data, we carried out the planning of the execution of the exploratory study. In order to give more confidence, the exploratory study was planned based on the Wohlin *et al.* [142] approach. Therefore, the planning is organized by the sequence of the following subsections (i) context selection, (ii) selection of subjects, (iii) instrumentation, and (iv) validity evaluation.

i. Context Selection The planning starts with the decision about the context in which the study would be carried out. The Android development environment makes it possible

to execute applications on real devices or emulated devices. Initially, we were interested in running the test cases on real devices. However, it would be necessary to modify the Android operating system to allow simulation of values on real devices. Thus, we decided to use the Android Emulator [17] that facilitates GPS simulation without the need to modify the Android operating system. Therefore, we defined the context of our study in an emulated environment.

ENVIAR performs tests in the emulated environment. We ran the test cases on an emulator that simulates a smartphone with 1.5 GB of RAM, 64 GB of internal storage, and running Android Oreo 8.1. The emulators are very faithful to the real devices; they use the same operating system, have emulated hardware, and provide an interface for the tester to enter data from some sensors, such as GPS and accelerometer. They can also simulate phone calls and SMS messages arrivals. However, because they do not have their own hardware, the emulator suffers from physical limitations imposed by its own computer that executes the emulator, such as the limit of RAM and processor capacity. The process scheduler also limits the execution speed of applications since the process that runs the emulator is not the only one allocated by the operating system of the running computer. To minimize this problem, we ran the tests on a dedicated computer to execute only this activity. The emulator ran on a computer equipped with an Intel Core i7 2.4 GHz processor (3.0 GHz with Turbo Boost), 16GB of RAM, 1TB of SSD hard disk and dedicated AT RADEON 2GB graphics card. As the computer's hardware is much superior to the hardware allocated to the emulator, the emulator's performance is similar to the performance of a real device with the same emulated configurations. Also, we use real apps that are available on the Google Play web store to eliminate any bias by the part of the authors.

ii. Selection of Subjects In this work, we identified events and paths that have a good chance of causing context-aware applications to fail (Section 4.1). Therefore, we selected GPS navigators Applications as path-based applications.

iii. Instrumentation Accordingly to Wohlin *et al.* [142], the instruments for an experiment are of three types, namely objects, guidelines, and measurement instruments. The objects are 4 Android GPS based applications. The measurement instrument is the defects captured by

the ENVIAR tool. In this study, the guidelines are not applicable.

iv. Validity Evaluation There are different classification schemes for different types of threats to the validity of a study. In this work, we will follow the Cook and Campbell [48] classification. Cook and Campbell classify validity evaluation in four categories: conclusion, internal, construction, and external.

1. **Conclusion validity:** We may not have correctly classified the defects found because we do not have access to the AUT code. To minimize this risk, we organized the defects at the end of the execution of all test cases. In this way, we were able to make comparisons between executions of similar test cases. Thus, it was possible to observe groups of executions with similar scenarios that exhibited the same failure and, therefore, to state more confidently that a defect was found based on similar execution scenarios. Once the scenarios that caused the failures were identified, we were able to more reliably determine if the defect is a context defect based on context variation.

2. **Internal validity:** 88 test cases of order 1 and 183 test cases of order 2 were performed in each of the 4 AUTs, totaling 1,084 executions, which could take up to 700 hours. As the tester gives the verdict of each test case, the tester could become fatigued and assign an incorrect verdict. Therefore, we took care to take rest breaks while observing the test case executions. Moreover, the sequence of executions can cause a threat of internal validity because there is a possibility that a test case execution will affect the result of the execution of the next test case. To avoid this risk, ENVIAR configures the test case setup before executing it. Besides, ENVIAR waits for the AUT to be configured according to the setup that was sent to the AUT. For example, if the test case setup determines that the GPS is on, ENVIAR waits for the GPS to be turned on and ready for use. Since each test case has its own setup and ENVIAR waits for it to be properly configured, we believe that each test case's verdict is not affected by previous test cases. Another internal threat risk factor is that a test case fails not due to a defect found, but for another reason (i.e., failure due to the Android Emulator). To eliminate this risk, we re-run all failed test cases to see if they failed again and only catalog the test cases that failed again. Finally, a possible cause of threat to internal validity

would be defects in the implementation of ENVIAR that would cause false failures in the execution of AUT. To minimize this threat, we chose to use the Android Emulator and the ADB to carry out the communication between ENVIAR and the AUT. Both the Android Emulator and the ADB were developed and tested by Google. Also, our solution uses the Android operating system without modifications. In this way, we minimize the possibility of ENVIAR to cause false failures in the AUT.

3. **Construction validity:** Smartphones currently have several sensors, such as a barometer, gyroscope, pedometer, compass, and GPS. Faced with so many resources provided by smartphones, the possibilities and heterogeneity between the types of context-aware applications are great. However, it is impracticable in our work's scope to test our approach for context-aware applications that can receive information from several sensors. Most context-aware applications use GPS. For this reason, we decided to use GPS based applications to represent context-aware applications.

4. **External validity:** We select four real applications from the Google Play Market place. These selected applications have up to 5,000,000 downloads. Therefore, they are widely used. Thus, we believe the applications well represent the applications used in practice by Android users. All applications were executed by the ENVIAR tool on a dedicated machine. In this way, the selection of the applications well represents the GPS based applications, and the test case execution environment is free of external interferences. Besides, manually introducing faults into applications could lead the exploratory study to bias. To minimize this threat, we selected events based on reported bugs and Amalfitano *et al.* Section 4.1. Consequently, the external validity threat has been minimized.

Exploratory Study Execution

With the exploratory study planning, we defined which applications to test under which test cases, in which context, and how to minimize threats to validity (exploratory study design). Thus, the next step is to execute the exploratory study under the planning guidelines.

The execution of the exploratory study was done by using the ENVIAR tool observing the design of the exploratory study defined in the planning. Therefore, test cases of order 1 and

order 2 were executed in each of the four selected applications. Before starting the execution, the environment was prepared to run the test cases. A computer was prepared (described in the context selection subsection 6.1.1), leaving it dedicated to the task of executing the test cases. Thus, we configured the computer so that only the essential processes for the operation of the ENVIAR tool and the Android Emulator were to be loaded by the Windows operating system to minimize delays and slowness or interference in the execution of the test cases. The collection of the raw result is described in subsection 6.1.2.

Results and Analysis

We analyzed the raw result in order to answer the research questions. To do so, we catalog defects from failed test cases from each order and each AUT. The analysis of the raw result is more detailed in subsection 6.1.3.

6.1.2 Data Collection

At the end of the execution, the ENVIAR tool saved files (available here[1]) containing the verdict, execution time, and sent events from each test case of orders 1 and 2, the raw results. Tables C.1, C.2, C.3, C.4, C.5, C.6, C.7 and C.8 show the verdicts for each test case execution of order 1 and 2 in each AUT.

6.1.3 Results and Analysis

We can summarize the raw results from Tables C.1, C.2, C.3, C.4, C.5, C.6, C.7 and C.8 in Table 6.4.

[1] https://bit.ly/30z8t4h

Table 6.4: Summary of Raw Result

	Pass		Fail		Crash		Slept Time
App	#	%	#	%	#	%	
OsmAnd	248	91.51%	23	8.49%	0	0%	105.2h
GPS Offline Navigation	254	93.73%	17	6.27%	0	0%	118.2h
Genius Maps	263	97.05%	8	2.95%	0	0%	124.4h
Voice GPS Navigation	219	80.81%	52	19.19%	27	9.96%	87.5h
Total	984	90.77%	100	9.23%	27	2.49%	435.3h

Table 6.4 summarizes the result of the execution of the 271 test cases in each of the four applications. The table shows the number of test cases that passed, failed, crashed, and the time it took to run all 271 test cases in each of the applications. We discussed in Section 6.1.1 that running all test cases in all applications could take up to 700 hours. However, we can see that the total time of 435.3 hours has been spent. The total time of 700 hours is the maximum time that the test could take if all test cases had passed. Once a test case fails, its execution is terminated, and the next test case is executed. Therefore, test cases that fail take less time than those that pass. Considering that approximately 35% of the test suite runs the long path, and each long path takes a little over 1 hour, we can see that time can drop significantly if test cases fail.

Defects Found

Test case execution found different defects in the four applications: 2 defects in the OsmAnd application, 4 defects in the GPS Offline Navigation application, 2 in the Genius Maps application, and 5 in the Voice GPS Navigation application, totaling 13 defects. Table 6.5 presents them.

In the table 6.5, we have six columns: (i) name of the AUTs, (ii) the defect identifier, (iii) brief description about the defect, (iv) test case number identifiers from the tables C.1, C.2, C.3, C.4, C.5, C.6, C.7 and C.8 where the failures occurred, (v) order of the test cases and (vi) what triggered the failures.

Table 6.5: Applications Under Test Defects

Application	Defect ID	Failure Description	Test Case Number	Test Order	Cause
OsmAnd	D01	The AUT included a distant point in the path's route (which should not be part of the route) when the GPS initially out of calibration is calibrated	16	1	GPS Calibration
			9, 12, 21, 31, 39, 41, 47, 49, 67, 71, 81	2	
	D02	Did not recalculate the route	40, 57, 61	1	GPS Calibration
			83, 123, 133, 134, 144, 148, 169, 170	2	
GPS Offline Navigation	D03	Finished the Navigation Activity	3, 15, 26, 28, 30, 64, 66, 77, 81	1	The event SIMULATE_LONG_BACKGROUND
			85, 123, 168, 179	2	
	D04	The AUT included a distant point in the path's route (which should not be part of the route) when the GPS initially out of calibration is calibrated	9	2	GPS Calibration
	D05	Navigation stopped working	22, 144	2	GPS Calibration
	D06	The AUT restarted	160	2	Internal reasons
Genius Maps	D07	Navigation stopped working	3, 30	1	The event SIMULATE_LONG_BACKGROUND
			32, 46	2	
	D08	Did not rendered the arrow which indicates the user's position	13	1	Activity went to the background or device orientation changed.
			60, 61, 129	2	
Voice GPS Navigation	D09	The AUT stopped working abruptly (Crashed)	4, 7, 21, 51, 61, 76, 82, 84	1	Internal reasons
			30, 35, 37, 40, 44, 58, 66, 71, 88, 89, 102, 111, 124, 135, 148, 153, 156, 176, 180	2	
	D10	Finished the Navigation Activity.	11, 13	1	Activity went to the background or device orientation changed.
			39	2	
	D11	The AUT included a distant point in the path's route (which should not be part of the route) when the GPS initially out of calibration is calibrated	16	1	GPS Calibration
	D12	The application did not respond, but it did not crash	63, 85, 87	1	Internal reasons
			103	2	
	D13	The execution of AUT become very slow	69, 72, 75	1	The event Internet_OFF in a long path
			125, 128, 133, 134, 137, 140, 141, 144, 145, 149, 159, 162, 165, 170	2	

D01 was found in scenarios where the GPS was initially out of calibration and then became calibrated again. When the GPS is out of calibration, the GPS accuracy is low and the AUT can understand that the device is in a distant location from where it is [19], [18]. During the execution of test cases that failed because of defect *D02*, OsmAnd tried calculating the new route, but never managed to finish calculating.

Regarding defect *D03*, when ENVIAR simulated the execution of the GPS Offline Navigation application in the background for a considerable time, the Navigation Activity ended, and navigation finished abruptly. Failure from defect *D04* had a behavior similar to the failures from defect *D01*. In the test cases that fail for *D05*, the AUT stopped providing the navigation service: the AUT started with GPS out of calibration and then it was recalibrated. From that moment, the AUT should react to the user's location variation and indicate the path to the destination. However, the navigation Activity was not rendered. In the test cases that fail for defect *D06*, the AUT should normally return to navigation after having executed the photography application. However, AUT restarted.

During the execution of test cases from defect *D07*, the AUT returned to the Navigation Activity but finished the navigation functionality unexpectedly. During the execution of test cases from defect *D08*, the arrow indicating the direction the user is in navigation was not rendered in the graphical interface.

Failures from defect *D09* caused a crash in the AUT. Failures from defect *D10* are similar to failures from defect *D03*, and defect *D11* failures are similar to failures from defect *D01*. Failures from defect *D12* occur when the AUT does not close abruptly but remains unresponsive to any stimuli from the sensors or the user. That is, the AUT becomes unresponsive to any stimulus forcing the user to close it. Failures from defect *D13* occur when the AUT continues to function but is very slow in responding to stimuli from the sensors or the user.

More details on the defects found and a report describing the observation of the 13 defects' behavior can be found in Table D.1.

If we compare our approach with manual execution, we can classify the 13 defects in the following categories:

- Impossible to be reproduced: *D01*, *D02*, *D04*, *D05*, *D11*;
- Difficult to reproduce: *D13*;

- Easily reproducible: *D03*, *D07*, *D08*, *D10*.

Defects *D01*, *D02*, *D04*, *D05*, and *D11* are defects that cannot be reproduced manually because there is no way for the user to test these test cases without computational assistance. All of these defects are defects due to GPS Calibration. This scenario cannot be reproduced without the tester using simulated data. For this reason, we believe that these defects were present in three of the four tested applications.

The *D13* defect is considered a difficult defect to reproduce manually because it occurs due to the combination of two situations: internet failure and long path. It is easily possible for a user to simulate a failed internet connection over a long path; it is enough to disconnect the antennas' internet connection during the trip. However, we find this test difficult as it requires much effort to execute. That is, to reproduce this scenario manually, it is necessary for the tester to physically move during a long path without an internet connection and observe if the application behaves correctly. Our study simulated the displacement from Campina Grande - PB to Teresina - PI; approximately 1,025 km. To reduce execution time, we set up the test case to simulate a maximum speed of 1,000 km/h. In a manual simulation, this speed cannot be reached. It is impossible to reach more than 120 km/h on this route without violating traffic laws. According to Google Maps[2], the path takes more than 15 hours to complete. Therefore, this scenario is difficult to be tested manually.

Defects *D03*, *D07*, *D08*, and *D10* are defects that we believe can be reproduced manually without much effort. These defects occurred due to long background simulation, Activity went to the background, or device orientation changed. It is entirely possible to test these situations manually.

As defects *D06*, *D09*, and *D12* occurred for internal reasons, we have not been able to conclude which of the three categories they belong to.

RQ1: The tool was able to find 13 defects that were classified into 5 groups of causes: (i) GPS Calibration, (ii) Simulation of long background, (iii) The event Internet_OFF in a long path, (iv) Activity went to the background or device orientation changed, and (v) by internal reasons.

[2]https://maps.google.com

Failures from defect *D01*, *D02*, *D04*, *D05*, and *D11* were caused after the GPS_CALIBRATED event happened. However, it was not the GPS_CALIBRATED event that alone triggered the failures. The failures occurred because the GPS was out of calibration (in the test case setup or because it received the GPS_NOT_CALIBRATED event during the execution of the test case) and later received the GPS_CALIBRATED event. Failures from defect *D03* and *D07* occurred after the event SIMULATE_LONG_BACKGROUND. As the name suggests, this event simulates the user leaving the AUT in the background for a long period of time. Failures from defect *D08* and *D10* occurred after the AUT went into the background. Unlike the *D03* and *D07* defects, it was not the SIMULATE_LONG_BACKGROUND event that caused the failures. Failures from defect *D08* and *D10* were due to the same lack of data management as Activity, but it was not necessary to force Activity to leave the paused state for the stopped state. The failures from defect *D13* happened when the INTERNET_OFF event occurs during the route of a very long path. Unlike the other applications, the Voice GPS Navigation application requires the internet to calculate its route. During the route of not so long paths, the Voice GPS Navigation application did not fail when the internet connection failed, but on the long route, the application was extremely slow to respond to stimuli when the internet connection failed, even if the connection becomes on again.

Failures from defects *D02*, *D03*, *D04*, *D05*, *D07*, *D08*, *D10* and *D11* occurred under the same context scenarios and by the same events respectively, but failures from defects *D06*, *D09* and *D12* occurred under different events and different context conditions. As we do not have access to the application code, it was not possible to identify what caused the failures. In this case, accessing the code would allow us to look, using the logcat of the execution, which part of the code caused the failure and identify the reason. We attributed as internal reasons for these defects. For this reason, as *D09* was observed in more than one test case failure and the same can be said of *D12*, it is possible they are not just two single defects in the code.

Figure 6.2 presents a graph that illustrates the number of defects found by each of the described causes. We can see that GPS calibration was the cause that revealed the most defects.

Table 6.6 organizes the defects found as context defects and non-context defects. Thus,

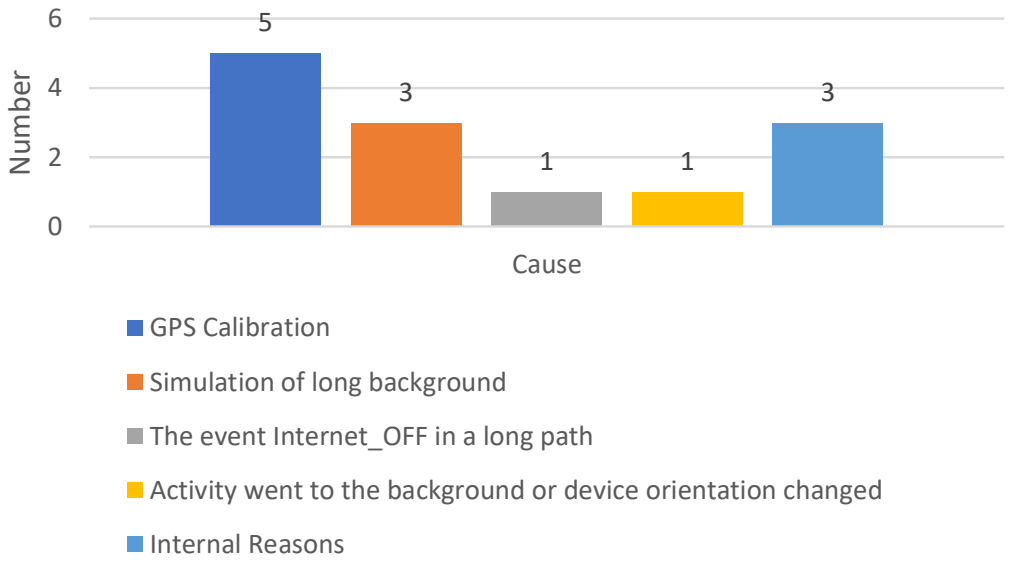


Figure 6.2: Defects Cause Frequency Graphic

it answers the RQ2 of our exploratory study. Defects *D01*, *D02*, *D04*, *D05*, *D11* and *D13* are classified as context defects as it was exclusively the context circumstances that triggered the failures. The other defects found caused failures that did not occur due to exclusively contextual circumstances. Therefore, they are not context defects. Even so, although other tools can simulate some events such as placing an application in the background or changing the orientation of the device (landscape or portrait), we do not know tools that generate test cases combining types of paths with events with greater possibility of fail using the technique of selection of pairwise test cases.

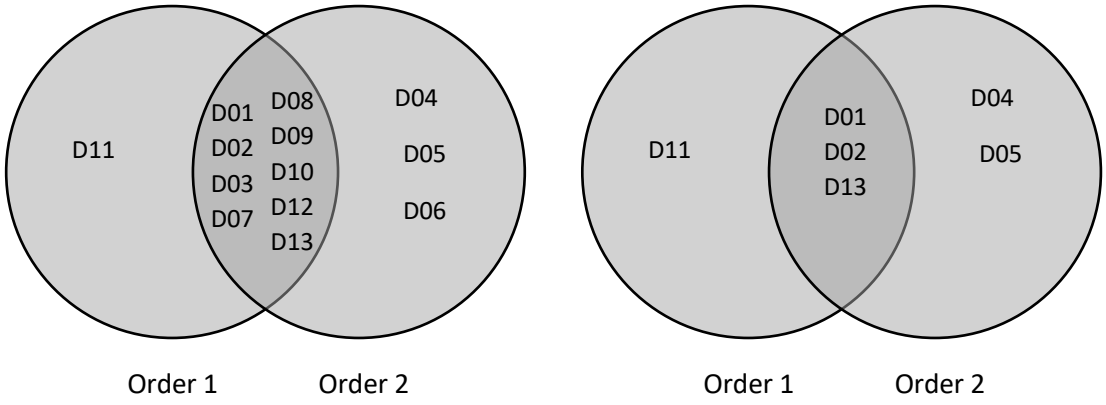
RQ2: ENVIAR found 6 context defects. GPS calibration events contributes to reveal most of them.

Table 6.6: Context Defects and non-context defects

Defect type	Defect IDs	Failure Description	Cause
Context Defects	D01, D04, D11	The AUT included a distant point in the path's route (which should not be part of the route) when the GPS initially out of calibration is calibrated	GPS Calibration
	D02	Did not recalculate the route	GPS Calibration
	D05	Navigation stopped working. The AUT behaved just like if the GPS was turned off.	GPS Calibration
	D13	The execution of AUT become very slow	The event Internet_OFF in a long path
Non-Context Defects	D03, D10	Finished the Navigation Activity	The event SIMULATE_LONG_BACKGROUND
	D06	The AUT restarted	We were unable to identify the reason
	D07	The navigation feature stopped working	The event SIMULATE_LONG_BACKGROUND
	D08	Did not rendered the arrow which indicates the user's position	Activity went to the background or device orientation changed.
	D09	The AUT stopped working abruptly (Crashed)	We were unable to identify the reason
	D12	The application did not respond, but it did not crash	We were unable to identify the reason

From Table 6.5, we can see that defect *D11* was captured only by test cases of order 1, defects *D04*, *D05* and *D06* were captured only by test cases of order 2 and both orders captured *D01*, *D02*, *D03*, *D07*, *D08*, *D09*, *D10*, *D12*, and *D13* (Figure 6.3(a)). Thus, test cases of order 1 were able to find 10 defects in 88 test cases (8.8 test cases by defect), while test cases of order 2 found 12 defects in 183 test cases (15.25 test cases by defect).

For context defects only, we can see that context defect *D11* was captured only by test cases of order 1, context defects *D04* and *D05* were captured only by test cases of order 2 and that the context defects *D01*, *D02* and *D13* were captured by both orders (Figure 6.3(b)). Thus, test cases of order 1 found four context defects in 88 test cases (22 test cases by context defects), while test cases of order 2 found 5 context defects in 183 test cases (36.6 test cases by context defects).



(a) All Defects

(b) Context Defects

Figure 6.3: Defects in Each Order

RQ3: Test cases of orders 1 and 2 may find different defects. The rate of test cases per defect is lower for order 1, but order 2 adds value to the test suite by identifying additional defects.

The average is 20.846 test cases by defect found considering orders 1 and 2 together and 45.167 by context defect found. Thus, order 1 presented a lower average than order 2 or even than order 1 and 2 together, considering all defects and considering only context defects. However, by executing only test cases of order 1, it would not be possible to find defects *D04*, *D05*, and *D06*. In contrast, by executing only test cases of order 2, it would not be possible to find defect *D11*.

6.1.4 Discussion

In this subsection, we discuss some observations of the results.

Factors that contributed to revealing defects.

GPS calibration was the factor that most triggered failures in AUTs. The fact that the GPS is momentarily out of calibration revealed five defects in the tested applications. It is important to highlight that the GPS calibration is a scenario that should be tested in a simulated environment. Thus, it would be impossible or very difficult to test this circumstance in manual testing, which may explain why it was the most found.

To find defect *D13*, it was necessary to use a long route and eliminate the internet connection. This scenario is also challenging to test manually. Therefore, it was so frequent in the Voice GPS Navigation application because it requires an internet connection to calculate the route. To the best of our knowledge, no tool can be found in the literature that simulates the path of short, long, and limit values under adverse situations such as the variation of GPS calibration, fall, and recovery of GPS signal, fall, and recovery of internet connection.

Pairwise Testing and Different Test Orders

Although test cases of order 2 did not found defect *D11*, we cannot say that test cases of order 2 are not able to find defect *D11*. From Definition 4.3, if we had not used pairwise testing, each test case of order 1 would be a subtest case of at least one test case of order 2. Therefore, without pairwise, any defect found in order n test cases could possibly be found in order $n + x$ for every $x > 0$. The pairwise technique eliminates test cases to achieve the smallest set of test cases that ensures that all pairs of events are combined. We have drastically reduced the number of test cases, but we have also eliminated some combinations of event groups.

Defect *D04* can be detected by test case 9 of order 2. The test case starts with setup S3 and, as soon as the tool begins simulating the path, the GPS calibrates again. After a few moments, the GPS is turned off. The failure of this test case occurred shortly after the GPS was calibrated. Thus, the event of turning off the GPS did not influence the failure. For this reason, this defect could be discovered by test cases of order 1 (excluded due to the pairwise technique).

On the other hand, defect *D05* was found in test cases whose setup forced the GPS to be initially turned off. After starting the path simulation, the GPS is turned on, and then it is calibrated. In this scenario, two events were required: GPS_ON and GPS_CALIBRATED. Therefore, no order 1 test case would be able to find this defect.

Regarding defect *D06*, we cannot say for sure whether it would be possible to be detected by test cases of order 1, as it was not possible to determine what motivated the failure.

In summary, we can observe that:

Lower-order test cases are more concise and punctual: the lower the order of the test case, the more punctual is the detection of the defect, that is, order 1 test cases that failed were probably due to the event sent by the test case whereas failures that occur in a test case of order n may be triggered by any (subset) of the n events in the test case or by the combination of the all n events together.

Trade off between execution time and the number of defects found: the lower the order of the test case, the less time it will take to run the test suite, but also fewer defects can be captured. The execution time between orders 1 and 2 more than doubles, but the growth of the time curve is not so great between test cases of orders 2 and 3, 3 and 4, 4 and 5. Taking into account the execution time, the decision of which order to choose will depend on how much time the test team has available to test the AUT and how much they are willing to take the risk of leaving a defect without being captured.

If execution time is not a great problem, execute more than one test order is a good strategy to increase the chances of finding defects and, at the same time, being more punctual in the defects found.

The higher the order of the test case, the more different combinations of events are tested: n order test cases generate combinations that $n - 1$ test cases do not explore. The higher the order, the greater the combination of events generated in the test cases. However, the execution time will be longer, and the accuracy of which event triggered the failure will be smaller.

6.2 Comparing ENVIAR to other tools

Some context-aware applications make use of a constant variation of context data. GPS navigators, for example, must always respond promptly to changes in the user's geographic position. Therefore, context-aware application testing tools must support the constant variation of context data. In addition to this feature, ENVIAR is capable of generating and executing test cases for context-aware applications that use location data type. Thus, in order to make it possible to compare other tools with ENVIAR, the following characteristics are required in the tools:

1. Be able to execute scenarios with large context data variations such as a user's path in a GPS navigator;
2. Be able to generate test cases;
3. Be able to execute test cases;
4. Be available to download.

The systematic mapping (from Chapter 3) brings results from articles published between 2008 and 2018. Thus, we performed new searches in IEEE and ACM, from 2018 to 2020, looking for test case generation tools for context-aware Android applications. We used the same search strings as the systematic mapping. The new search resulted in:

- 5172 articles search engine results;
- 520 articles selected for abstract, introduction and conclusion reading;
- 170 articles selected for full reading;
- 1 result article.

As a result of the new search, we found the DroidMate2 tool. Borges *et al.* [40] present the DroidMate2 tool, which is an evolution of the DroidMate tool. DroidMate2 makes it possible to test applications using random strategies, fitness-based or re-execute previous recorded executions enabling the tester to implement systematic testing strategies on top of a ready to use selection of strategies. DroidMate2 interacts with AUT through events such as click on coordinates (x,y), enable the device wi-fi, Bluetooth, and restart the app from its initial screen.

Monkey is a stress testing tool developed by Google. It sends pseudo-random GUI and system events to the AUT. Some studies attest that, although using a simple strategy, Monkey is one of the tools that can most reveal failures and has more code coverage capability [46], [114], [136]. For this reason, this tool is used by many authors as a comparison tool [90], [46]. We did not find studies exclusively related to study Monkey. However, since Monkey is used as a comparative study by many authors, we decided to choose Monkey for our comparative study.

Table 6.7 illustrates the characteristics presented by each of the ten tools discussed in the Subsection 3.3.2, Monkey, DroidMate2, and ENVIAR among the desired characteristics listed.

The table consists of five columns:

i Column indicating the name of the tool;

ii Column indicating which tools support the execution of test cases with large context data variations scenarios;

iii Column indicating the test case generation strategy used by the tool;

iv Column indicating which tools execute test cases;

v Column indicating which are available for download.

Table 6.7: Desired Characteristics

Tool	Desired characteristics			
	Execution of large context data variations scenarios	Test case generation strategy	Test case execution	Available
Calabash-Android Custom-built version		Model-Based Testing	X	
Context Simulator			X	
Extended AndroidRipper	X	GUI Ripping	X	
ContextDrive		Model-Based Testing	X	X
TestAWARE	X		X	X
CrashScope		GUI Ripping	X	X
MobiPlay	X		X	X
VALERA	X		X	X
RERAN	X		X	X
MBTS4MA		Model-Based Testing	X	X
DroidMate2		Random	X	X
Monkey		Random	X	X
ENVIAR	X	Pairwise Testing	X	X

Looking at the second column, we can see that:

- Except for ENVIAR, no tool that supports the execution of test cases with large context data variations scenarios is capable of generating test cases. The tools that support this type of execution are the Record and Replay tools. These tools allow re-executing previously executed scenarios, but first, they need the data to be executed. This data is generated through a previous real execution of the AUT. In this execution, the sensor data are stored for later re-execution. In the ENVIAR tool, the sensor data is written in a script and provided by the tester. Therefore, ENVIAR does not require the AUT to be executed beforehand. Thus, unlike Record and Replay tools, it is possible to test through ENVIAR scenarios that are difficult to execute manually;

- ENVIAR uses an innovative test case generation strategy. Tools that use Model-Based Testing are widely used in the literature. However, this technique has two inherent problems: the need for familiarity with modeling, and cost to build and maintain the AUT model. Thus, to test an application using this technique, it is necessary first to build the AUT model. Complex applications can add a high cost to their modeling. Also, the models must always be in line with any changes to the AUT specification. Tools that use GUI Ripping have a lower cost than those that use Model-Based Testing because they build the AUT model as they run it. These techniques identify GUI elements through the description of graphic elements in the .xml files of the applications. However, there are graphic elements that are created dynamically and that are not explicitly described in .xml files. Therefore, they may not test specific features of AUT. Random strategies are simple to use and have a low cost but have limitations when we want to test specific features. ENVIAR uses pairwise testing to generate test cases. This strategy allows the creation of execution scenarios by selecting pairs of events that are more likely to find defects and combining them with sets of sensor values. The use of pairwise testing allowed us to generate test cases with viable sizes to be executed. Also, it showed us to be able to find context defects and defects in scenarios of difficult manual execution, Subsection 6.1.3. Besides, unlike the approach that uses Model-Based Testing, the approach implemented in ENVIAR does not require testers' experience in modeling. All that the tester needs is to provide a script with the main geographic points of the path to be simulated and open the AUT in the functionality to be tested;

- Excepting ENVIAR, none of the tools can generate and execute test cases for context-aware applications that make use of a constant variation of context data;
- All the tools mentioned execute test cases;
- Calabash-Android Custom-built version, Context Simulator, and Extended AndroidRipper are not available to download.

To be able to compare the execution of a tool with the ENVIAR, the tool must have all the characteristics mentioned in Table 6.7. Calabash-Android Custom-built version, Context Simulator, and Extended AndroidRipper are not available for download. Thus, we cannot execute them.

TestAWARE, MobiPlay, VALERA, and RERAN are not able to generate test cases. These tools are record and replay tools. One of the paths we used in the exploratory study (Subsection 6.1.1) was a path with approximately 1,000 km. Creating test cases in these tools with this path for each of the four chosen applications (Subsection 6.1.1) is unfeasible for our work scope.

ContextDrive, CrashScope, MBTS4MA, DroidMate2, and Monkey are not able to execute test cases with large context data variations scenarios. Thus, it is not possible to test scenarios that simulate the paths used in the exploratory study of Section 6.1.

None of the tools can generate and execute test cases with large context data variations scenarios. Therefore, we could not compare the execution of any of them with ENVIAR. However, it is possible to execute ENVIAR as a supporting tool allowing the simulation of the paths. Section 6.3 details how we can use ENVIAR as a supporting tool.

6.3 ENVIAR Supporting Other Tools Execution

The ENVIAR tool sends two types of inputs to the AUT: (i) GPS positions, (ii) events more likely to find defects. The sending of GPS positions gives the ENVIAR the ability to simulate the variation of GPS positions and, consequently, simulate a user walking path. We can add this ability to simulate the variation of GPS positions to other tools if we use ENVIAR as a supporting tool. Therefore, we can improve other tools providing them the ability to simulate the constant variation of GPS positions. This improvement makes possible the execution of

scenarios with large GPS data variations, such as a user's path in a GPS navigator. Hence, we can use the ENVIAR tool to simulate the user's path without sending system events. Figure 6.4 illustrates how it is possible to use ENVIAR as a supporting tool for the simulation of paths to be taken.

Initially, we used the ENVIAR tool to create test cases. Each test case contains only each path to be tested. After generating the ENVIAR test cases, the tester prepares the AUT and the other tool. That is, the tester opens the AUT and makes it ready to be tested, and the tester configures the other tool and makes it ready to execute. Once the ENVIAR test cases are generated, and the AUT and the other tool are ready, the tester executes the ENVIAR and the other tool and keeps observing the AUT. If any incorrect behavior occurs, the test case fails. Otherwise, the test case passes. This procedure is performed for each test case generated by ENVIAR. In other words, for each path to be tested, the tester prepares the AUT, prepares the other tool, executes both tools, and observes the AUT.

Some of the mentioned tools in Section 6.2 (CrashScope, MBTS4MA, DroidMate2, and Monkey) can be used as examples for the use of ENVIAR as a supporting tool. These tools generate and execute test cases and could be used as examples, but CrashScope and MBTS4MA could not be used by the following reasons:

- CrashScope is an online tool[3]. The tester needs to uploads the AUT's .akp file to use it. CrashScope executes in a server. Therefore, it is impossible to simulate the user's path through ENVIAR, making it impossible to compare it with our tool;

- Unlike CrashScope, MBTS4MA is a stand-alone tool that makes it possible to use ENVIAR to simulate a path. However, for MBTS4MA to generate test cases, the AUT source code is needed, but we do not have access to the code of the applications used in our experiment. Furthermore, to use MBTS4MA, the AUT source code must be made using Eclipse ADT[4]. Eclipse ADT was discontinued by Google in 2016 and replaced by Android Studio[5] since then. Therefore, it is not possible to compare MBTS4MA with ENVIAR.

[3]https://www.android-dev-tools.com/crashscope

[4]https://marketplace.eclipse.org/content/android-development-tools-eclipse

[5]https://developer.android.com/studio

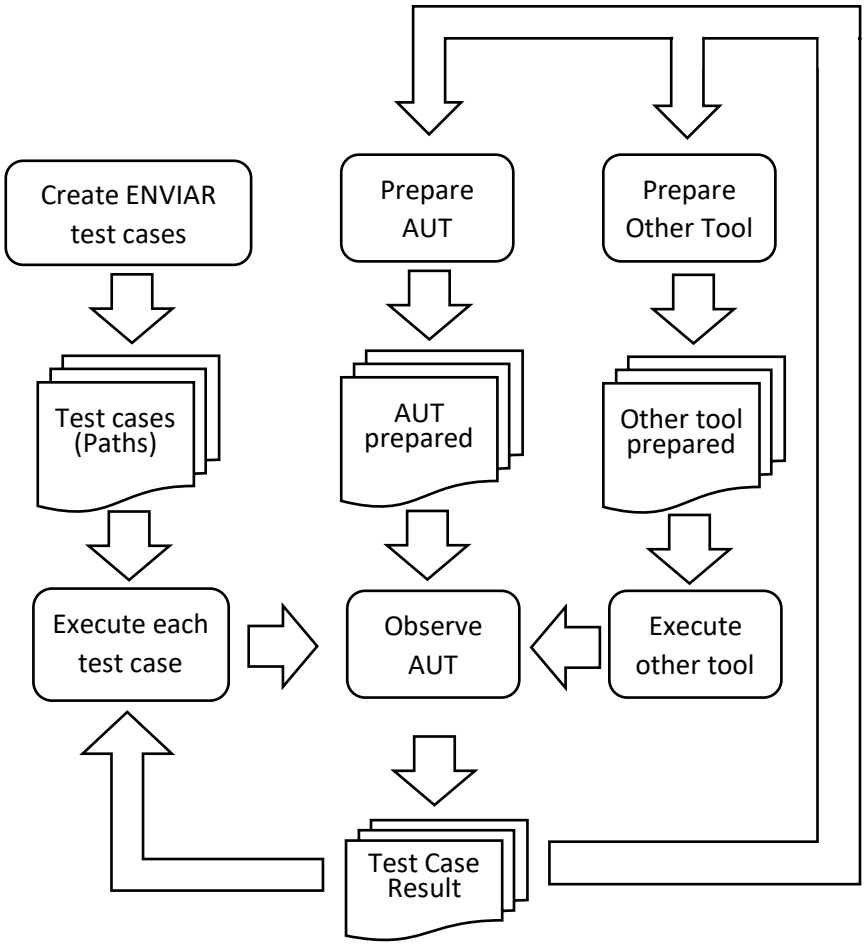


Figure 6.4: ENVIAR Supporting Tool

Therefore, we selected Monkey and DroidMate2 to use as examples for the ENVIAR supporting. We configured these two tools so that we could test the same paths as the test cases of the exploratory study of Section 6.1 in the four selected applications. Hence, we configured Monkey and DroidMate2 to send their random events to AUT while ENVIAR sends the paths' GPS coordinates. In other words, ENVIAR sends geographic positions to the AUT to simulate the walking path, while the compared tools (Monkey and DroidMate2) sends its events. While the test case runs, the tester observes whether the AUTs exhibit any behavior that is considered incorrect and then catalogs the failures of the test cases and looks at the stimuli sent to the AUT.

Patel *et al.* [114] ran the Monkey tool on 15 famous popular commercial apps and affirm that apps fail after entering an average of 7,513 events. Therefore, we set up Monkey to send 10,000 events in the four applications tested.

DroidMate2 does not have the option of choosing how many random events should be sent to the AUT. Therefore, we set up for DroidMate2 to send random events during the time necessary to run the entire path.

We used the same four paths described in Section 6.1.1 and run Monkey and DroidMate2 in the same four applications (Section 6.1.1) from the exploratory study of the Section 6.1.

At the end of the execution, ENVIAR saved files containing the verdict, execution time, and GPS Coordinates. Monkey saved files containing the random seed, event kind percentage, sent events, and some Android system outputs. Finally, the DroidMate2 saved files containing screen-shots, logcats, and UI app states representation. All saved files from these tools are available here[6].

Table 6.8 shows the results of the test cases execution. It shows ten failures and one false crash when running Monkey, 12 inconclusive verdicts when running DroidMate2.

Table 6.8: Tools comparison results.

App	Test Case id	Path	Monkey			DroidMate2		
			Verdict	Context Defect	Observation	Verdict	Context Defect	Observation
OsmAnd	TC01	Small	Pass	No		Inconclusive	--	Need to download map
	TC02	Perfect	Pass	No		Inconclusive	--	Need to download map
	TC03	Limit	Fail	No	The app became too slow	Inconclusive	--	Need to download map
	TC04	Long	Fail	No	Stopped the navigation.	Inconclusive	--	Need to download map
GPS OffLine Navigation	TC05	Small	Fail	No	Crashed	Inconclusive	--	Need to download map
	TC06	Perfect	Fail	No	Crashed	Inconclusive	--	Need to download map
	TC07	Limit	Pass	No	False Crash	Inconclusive	--	Need to download map
	TC08	Long	Fail	No	Crashed	Inconclusive	--	Need to download map
Genius Maps	TC09	Small	Fail	No	Crashed	Inconclusive	--	Need to download map
	TC10	Perfect	Pass	No		Inconclusive	--	Need to download map
	TC11	Limit	Pass	No		Inconclusive	--	Need to download map
	TC12	Long	Pass	No		Inconclusive	--	Need to download map
Voice GPS Navigation	TC13	Small	Fail	No	Crashed	Pass	No	
	TC14	Perfect	Fail	No	Crashed	Pass	No	
	TC15	Limit	Fail	No	Crashed	Pass	No	
	TC16	Long	Fail	No	Crashed	Pass	No	

The OsmAnd application failed in the *TC03* and *TC04*. In the *TC03*, the AUT became slow. After some time (precisely 6 minutes and 18 seconds), OsmAnd showed slowness between the receipt of stimuli and the production of their respective responses. During the *TC04*, the application stopped running navigation after 2 minutes and 28 seconds. That is, OsmAnd was showing the user's current location, but without showing the path that he must take to his destination.

[6]https://bit.ly/2Yvxvj8

The GPS OffLine Navigation application crashed during the *TC05*, *TC06*, and *TC08*. When it crashes, the AUT closed abruptly and had to be restarted manually by the tester. Monkey accused that AUT crashed while executing the *TC07*. However, we observed the execution of the test case and did not see the application showing any incorrect behavior.

The Genius Maps application crashed in the execution of the *TC09* while the Voice GPS Navigation application crashed in the test cases *TC13*, *TC14*, *TC15* and *TC16*.

The output generated by Monkey in each test case describes, in general terms:

- What arguments were used to execute Monkey;
- What was the used seed;
- Percentage of types of events;
- Elapsed time.

Although it is possible to re-run a test case with the same events through the seed of the test case, Monkey does not sufficiently detail which events caused the AUT crash. Thus, it was not possible to identify which reasons caused the AUT's failures and then identify the defect. Also, Monkey cannot run the scenarios that review defects *D01*, *D02*, *D03*, *D04*, *D05*, *D07*, *D11*, and *D13* from Table 6.5 since it focus on GUI interaction. These defects caused failures due to GPS Calibration, Long Background Simulation, and internet connection failure in a long path that accounted for most of the causes of ENVIAR failures. As Monkey does not generate events these events, it was unable to identify context defects in the studied AUTs.

DroidMate2 did not fail in any test case in its executions. The test cases *TC01*, *TC02*, *TC03*, *TC04*, *TC05*, *TC06*, *TC07*, *TC08*, *TC09*, *TC10*, *TC11*, and *TC12* were inconclusive because OsmAnd, GPS OffLine Navigation, and Genius Maps need to download the map of the path's location before starting GPS navigation. For each execution, DroidMate2 uninstalls the AUT and reinstalls it again, so the map is deleted from the AUT database. For this reason, we attribute the inconclusive verdict to these test cases since the AUT did not proceed to the Navigation Activity. The Voice GPS Navigation application does not require a previously downloaded map. Therefore, it was possible to carry out its execution, but there were no failures.

Monkey is a GUI stress testing tool that also sends some system events, so although it has been very effective at finding GUI errors, we cannot say that it is efficient at finding context defects. Even using ENVIAR as a support tool to send GPS positions, we could not say that we found context defects because the outputs of Monkey do not make it clear which events occurred when we identified the failures. As ENVIAR generates the AUT logcat, the access to the AUT code could allow the identification of the defect in the code, and then understand which event caused the failure.

6.4 Concluding Remarks

This chapter carried out an exploratory study in order to evaluate the approach proposed in Chapter 4. The exploratory study identified 13 defects. Among these, 6 are context defects. We also analyzed the results from the perspective of the test case orders by observing the advantages and disadvantages of using order 1 or order 2 test cases.

We compared ENVIAR with other existing tools and exposed why other tools could not be used instead of ENVIAR to test context-aware path-based applications that respond to constant context variation. Finally, we highlighted how ENVIAR can be used as a supporting tool to assist other existing tools in testing context-aware applications.

Chapter 7

Related Works

This chapter presents an overview of relevant work on testing context-aware Android applications. Although research on testing context-aware applications is growing, there are still few studies related to this area. Section 7.1 presents related works on reviewing the state of the art in the area of interest in our work. Section 7.2 presents solutions from other authors related to the research area of our work.

7.1 State of the Art Reviews

This section presents five works presenting state of the art in the area of interest in our work. At the end of this section, the works are compared with the systematic mapping of Chapter 3.

7.1.1 Matalonga *et al.*

A quasi-systematic review was performed by Matalonga *et al.* [97]. The quasi-systematic review aimed to find and characterize methods used for testing Context-Aware Software Systems (named as CASS for short). Among their found results, they argue that there is no complete context-aware test method for testing such software systems. In addition, Matalonga *et al.* emphasize that CASS context information changes continuously. Thus, it is important to test whether the constant variations of context information cause unexpected behavior. However, the identified techniques follow the same strategy: identify predefined

and fixed values for context variables. For example, when testing a context-aware application that uses GPS, the test case will test predefined fixed location values, and these values will not change during the execution of the test case. Therefore, Matalonga *et al.* highlight the importance of testing context diversity in context-aware applications.

7.1.2 Guinea *et al.*

A systematic review to evaluate the different phases of the software development life cycle for ubiquitous systems was conducted by Guinea *et al.* [56]. In their work, the main interests and limitations of each phase of the development cycle were classified. As a result, the systematic review identified 132 approaches that are related to the phases of the ubiquitous software systems development cycle. Among them, ten are related to the testing phase, which Guinea *et al.* argue that perhaps the testing stage is where research is most needed. After classifying the ten approaches by their focus or concern, the authors obtained five dedicated to context-aware testing, including 3 to simulators and 2 to test adequacy:

1. Approach of Wang *et al.* (Subsection 7.2.2);

2. Neill *et al.* [113] present an approach based on prototyping and environment simulation. This approach is more convenient for indoor environments. General approach and not directed to Android;

3. Wang *et al.* [134] present an approach that defines the concept of context diversity and does a study on how this concept may improve the effectiveness of the data flow testing criteria. The authors argue that data flow testing is a very efficient testing approach as it produces high code coverage and has been pointed out by several authors and empirical studies as an efficient form of testing. The study result shows that test cases with higher context diversity results in significantly improvement of existing data flow testing criteria efficiency;

4. Morla and Davies [109] present a solution and test for a ubiquitous remote patient monitoring system. Through a jacket with sensors, the Health-monitoring application monitors and reports location, lung and heart conditions through internet connection.

To test the system, the authors propose the prototyping and emulation of the environment in which they perform atypical situations such as internet connection failure;

5. Huang *et al.* [67] argue that in civil construction, it is common to implement testing of structures on smaller scales. However, time scales are not considered in these tests. Hybrid testing is a solution that integrates physical components of a structure of interest with computational models that can be promising for a computational solution for testing civil structures. Due to a lack of real-time hybrid testing support, the authors developed the design and implementation of novel middleware for their previous work where they developed an initial prototype of a Cyber-physical Instrument for Real-time hybrid Structural Testing.

7.1.3 Shauvik *et al.*

Shauvik *et al.* [46] performed a complete comparison of the main existing test input generation tools for Android. Four metrics were used to evaluate the effectiveness of these tools and their corresponding techniques: (i) code coverage, (ii) ability to detect faults, (iii) ability to work on multiple platforms, and (iv) ease of use. The authors consider that Android applications are event-driven. Therefore, the interaction with applications is through events such as clicks, pinches, text inputs, scrolls, or system events, such as the notification of the battery level. The generation of events (inputs) from these tools can be made randomly or following a systematic exploration strategy. In this sense, several techniques in the literature differ in how they generate inputs, and the strategies and heuristics used to explore AUT's behavior. The results of the tool comparison highlight the strengths and weaknesses of the tools and future research directions. However, none of the analyzes, conclusions, or tools studied took into account context-aware applications.

7.1.4 Santiago *et al.*

Santiago *et al.* [95] carried out a quasi systematic literature review and identified 11 relevant sources that mention 15 problems and 4 proposed solutions. The authors analyzed and classified the data into 3 groups of challenges and strategies for dealing with context-aware software systems testing.

7.1.5 Usman *et al.*

Usman *et al.* [127] performed a search on Scopus and Google Scholar databases between 2010 and 2017 in order to make a comparative study of mobile app testing approaches focusing on context events. The comparison was based on six key points:

1. Events identification;
2. Method of analyzing mobile apps;
3. Testing technique;
4. Classification of context event;
5. Validation method;
6. Evaluation metrics.

Their results indicate that the popular approaches offer limited coverage of mobile app context events.

7.1.6 Comparison

Although relevant, except Usman *et al.* 7.1.5, none of these studies discussed so far have investigated the ability of current testing tools for handling Android context-aware applications. Matalonga *et al.* 7.1.1 presented a quasi-systematic review to characterize methods but did not discuss whether the current tools are capable of testing context-aware applications. Shauvik *et al.* 7.1.3 aimed to perform a complete comparison on the main existing test input generation tools for Android, not focusing at context-awareness. Santiago *et al.* 7.1.4 aimed to identify the available knowledge on testing context-aware software systems. Usman *et al.* 7.1.5 performed a search on Scopus and Google Scholar databases. Therefore, they did not carry out a systematic search as occurs in a systematic mapping. Issues such as research questions, search criteria, number of studies identified, and validity evaluation are not discussed in their work. Also, Usman *et al.* conducted searches between 2010 and 2017 while our SMS (Chapter 3) surveyed studies between 2008 and 2018. Guinea *et al.*

7.1.2 aimed to evaluate the different phases of the software development life cycle for ubiquitous systems. The authors present studies of challenges and solutions, some with tools, but with their respective limitations. The systematic mapping we carry out (Chapter 3) not only allowed us to know tools, but also allowed us to identify:

- The techniques the authors implement;
- Which tools generate test cases;
- Which tools execute test cases;
- Which are the test methods;
- Which tools are available for download;
- Which tools are most commonly used.

7.2 Related Solutions

This section presents the solutions proposed by other authors related to the context of our work.

7.2.1 Sanders and Walcott

Sanders and Walcott [122] have proposed a technique that promises to test existing Espresso test suites easily. The authors developed a tool called TADS (Test Application to Device State) that enables a developer to run Espresso test suites against different changes of state of the device. However, so far, the technique focuses only on Wi-fi and Bluetooth device states.

7.2.2 Wang and *et al.*

Wang *et al.* [137] define the concept of context-aware program points (CAPPS). A CAPPS is a piece of code that identifies context change and, consequently, computes the application's behavior for that change. In its approach, input context data are systematically manipulated

to expose potentially valuable context variations. Thus, the technique performs the following tasks:

- Identify CAPPS;
- Generate variants for existing test cases allowing to test different context sequences;
- Dynamically direct the application's execution to the generated context sequences.

7.2.3 Amalfitano *et al.*

Amalfitano *et al.* [25] present a test solution for context-aware applications based on the assumption that apps are event-driven systems that receive context and GUI events. The authors present approaches based on event-patterns. An event-pattern is a sequence of events that represent a test scenario with a greater chance of failure. Amalfitano *et al.* define three strategies for generating test cases with event-patterns:

1. **Manual technique:** The tester manually chooses one or more event-patterns to define scenario-based test cases;
2. **Mutation-based technique:** The tester inserts event-patterns into existing test cases;
3. **Exploration-based technique:** This technique is used in automatic black-box testing based on dynamic analysis. During the test execution, the exploration technique detects the current set of sensing events of the app and then chooses one or more event-patterns to execute. The choice can be made systematically or in a random manner. Amalfitano *et al.* implemented the exploration-based technique in the Android Ripper tool [26].

7.2.4 Ami *et al.*

Ami *et al.* [32] present a tool called MobiCoMonkey (**Mobi**le **Co**ntextual **Monkey**) capable of automatically testing Android apps against auto-generated (random) or custom contextual scenarios. MobiCoMonkey is also able to create bug reports connecting the bug and contextual factors, making possible the later reproduction of bugs. Despite being a tool for

context-awareness, the tool currently only supports events of changing state of internet connection.

7.2.5 Comparison

Although relevant, none of these works present a solution for testing Android context-aware applications that generates and executes test cases simulating the environment in which the AUT is inserted and system events that are more likely to fail. Amalfitano *et al.* 7.2.3 present approaches based on event-patterns that resemble and inspired the insertion of our system events. However, the authors emphasize the interaction with AUT via GUI while our work focuses on applications that react more to stimuli from the environment than from the user. Also, our work performs the generation of test cases that, through pairwise testing, guarantees the combination of pairs of different system events in the test suite. Our work also presents a simple and efficient way to emulate fake events without the need to change the Android operating system or instrument the AUT code. While Amalfitano *et al.* analyzes their technique through code coverage, our work observes defects that occurred in real applications.

Wang *et al.* 7.2.2 present a white-box approach in which it identifies CAPPS in the AUT code. Our approach is black-box and does not use the AUT code. Also, the result by Wang *et al.* is a more general approach, not specifically for Android.

Sanders and Walcott 7.2.1 propose a technique that extends an Espresso test suite in order to make it possible to test different changes of state of the device. The authors consider only Wi-fi and Bluetooth device states. Our approach not only runs test cases with GPS positions changing, but it also generates test cases.

Ami *et al.* 7.2.4 present a tool that at first resembles ENVIAR. MobiCoMonkey generates bug reports and is able to test Android against auto-generated or custom contextual scenarios automatically. However, MobiCoMonkey only supports events for changing the state of the internet connection.

Table 7.1 illustrates a summary of the related work solutions mentioned above. The Table is divided into 7 columns: (i) authors, (ii) the level of the test, (iii) the name of the tool, (iv) a brief description of how the test activity is carried out on the tool, (v) sensors that the tool supports, (vi) whether the tool generates test cases, and (vii) whether the tool performs test

cases.

Table 7.1: Related Work Comparison

Work	Test Level	Tool	Testing Method	Sensors	Generate Test Case	Execute Test Case
Sanders and Walcott	System	TADS	Extending an existing Espresso test suite to enable the testing of different changes of state of the device.	Wi-fi and Bluetooth	No	Espresso test suite execution
Wang *et al.*	System	No	Generate potential variants for an existing test suite that explore the execution of identified key program points context information can effectively affect the application's behavior.	Any	No	Construction of middleware to simulate the environment. The solution is not specific to Android.
Amalfitano *et al.*	System	Extended Ripper	Combination of both context events and GUI events. The former is based on the definition of reusable event patterns.	Any	Exploration-based technique. The technique runs the application and detects which events the app can sense and react to. From the detected events, the tool generates test cases with event-patterns that explore these detected events.	Execute sequences of event-patterns
Ami *et al.*	System	MobiCoMonkey	The tool injects low-level contextual factors (i.e., GSM Profile, Network Delay, and Network Status) in each Activity.	Internet	Generates random contextual scenarios based on a user provided seed value.	Executes each application's Activity by interacting with its widgets while injects the low-level contextual factors.
Our Work	System	ENVIAR	It tests a reduced set of test cases. The test cases combine (i) large sets of sensor data with (ii) events that are more likely to cause failures.	Any	Use pairwise testing to generate a reduced set of test cases that combine sequences of sensor values with events that are more likely to cause failures.	Executes combinations of sensor values with events that are more likely to cause failures in a simulated environment.

7.3 Concluding Remarks

This chapter presented a review of relevant works to the research of this thesis. The authors present solutions for testing context-aware applications with limitations that do not satisfy the purpose of our work. Therefore, in addition to describing the solution proposed by each author, we discussed a comparison with our work at the end of the chapter.

Chapter 8

Concluding Remarks

This chapter summarizes the main results of this work, discusses some limitations, and presents some suggestions for future work.

8.1 Conclusions

In this work, we present a black-box approach to test context-aware path-based applications at the system level, considering the difficulty and cost of generating and executing test cases. To generate test cases, we use the technique of pairwise testing to select test cases with pairs of events more likely to find defects. The pairwise testing drastically reduces the number of test cases without significantly losing its efficiency to find defects. The execution is done in an emulated environment making possible the execution of scenarios that would be very difficult to run in a real environment.

Considering the research questions exposed in Chapter 1:

Research Question 1: *What are the Android context-aware testing studies and tools published in the literature?*

To answer research question 1, we carried out a systematic mapping of existing techniques in testing context-aware Android applications. The systematic mapping allowed us to know:

i The tools and studies for testing Android applications in the literature;

ii Which techniques are used in the literature;

iii Which studies aim at testing context-aware applications;

iv Which tools are available for download;

v Which are techniques limitations.

This knowledge was necessary for guiding the definition of the approach we use in our work. Thus, we defined that our approach should be able to generate test cases that deal with scenarios that are more prone to failures and test the constant variation of context (Section 4.1). We also defined the generation and execution of test cases that must be feasible to be done.

Research Question 2: *How to create and execute test cases to context-aware path-based Android applications?*

To answer research question 2, we defined that the generation of test cases should receive data sets from the GPS sensor and combine them with sequences of events more prone to failures (Section 4.2). However, the combination of sets of GPS values with sequences of events can take to a large number of test cases. Thus, to make the test case generation more feasible without significantly losing the ability to reveal failures, we used the technique of pairwise testing (Section 2.5) to select a viable number of test cases without significantly losing their ability to reveal defects. We also defined that the execution of the test cases must be done in a simulated environment. Thus, it is possible to execute the test cases in an easier way and in a controlled environment that allows the execution of challenging scenarios to be tested manually.

Research Question 3: *What kind of defects can the approach reveal?*

To answer research question 3, we implemented our approach in the ENVIAR tool (Chapter 5). We defined an exploratory study (Section 6.1) to evaluate the approach proposed in this work. Our results (Section 6.1.3) gave us evidence that our approach is promising in the generation of test cases with the potential to find context defects while drastically reducing the number of test cases that would be generated without pairwise testing. The execution in an emulated environment made it possible to run scenarios that would be very difficult to run in a real environment and, therefore, allowed us to find defects in real applications widely used by Android users.

In summary, this thesis provides the following contributions:

1. A complete review of relevant work throw a conducted systematic mapping;
2. A set of scenarios most likely to fail in path-based applications;
3. A test case generation approach capable of generating feasible and efficient test cases;
4. An execution approach that makes feasible the execution of challenging test scenarios;
5. A tool that implements the black-box testing approach proposed by our work.

8.2 Limitations

Some limitations identified in our work are:

- **The tester needs to follow the execution to define the verdict:** This limitation does not make our execution fully automatic;
- **Lack of automation of the ENVIAR tool in identifying defects:** The tester needs to look at the logs to decide which defect caused the failure in the test case. It would be interesting to automate the identification of possible pieces of code that caused the failure. This automation would reduce the cost of testing and the chance of the tester's wrong decisions;
- **Our approach does not currently test general context-aware applications:** This limitation does not allow us to test applications that use other sensors (e.g., accelerometer, compass, barometer);
- **Maximum of 5 events per test case:** Very long paths can reveal more defects if using more than five events;
- **The tester needs to inform path data:** Our approach does not automatically generate paths; it is necessary for the tester to inform the main points of the route that he wants to test;
- **Race condition when ENVIAR Supporting Other Tools Execution:** The ENVIAR tool stores all inputs that are sent to the AUT. Therefore, it is possible to re-execute faulty scenarios by sending the inputs in the same order as they were sent. However,

when executing ENVIAR combined with another tool (Section X), race condition may occur because we have no control over the thread of the tool to be combined. Therefore, it is not possible to re-execute faulty scenarios under the same conditions.

8.3 Future Work

With the completion of this work, there are several opportunities for future work. Next, some ideas are described:

1. **Extend the approach and tool:** Our approach focus on path-based applications. Thus, we plan to extend the approach and the ENVIAR tool to simulate more types of sensors to allow testing for other types of context-aware Android applications;

2. **Evaluate the ENVIAR tool:** We also plan to perform empirical studies that include more applications and investigate the use of ENVIAR by industry testers;

3. **Facilitate sensor data modeling:** GPS data modeling is currently composed only as a sequence of GPS values. We intend to study how we could easily model this data so that it would be possible to simulate situations such as standing, walking, running, jumping, and so on;

4. **Extend the scenarios most likely to fail:** Currently, the scenarios most likely to fail we cataloged are more focused on GPS based applications. We intend to extend this event catalog to more general context-aware applications;

5. **Testing with more test case orders:** Due to time constraints, we evaluate our technique in test cases of order 1 and order 2. We intend to extend our assessment to orders 3, 4 and 5 to catalog more advantages, disadvantages, and limitations concerning the choice of each test case order;

6. **Propose new combinations of test cases:** Currently, our technique generates test case suites for only one test case order at a time. We intend to evaluate the generation of test cases that combine different orders of test cases by suppressing test cases that are subtest cases;

7. **Propose exhaustive testing:** Currently, our approach supports only combinations of up to 5 events which is adequate to simulate specific context changes. However, we intend to extend the tool also to generate random combinations of more than 5 events to support more exhaustive testing, suitable for long paths.

Bibliography

[1] Cucumber. https://github.com/cucumber/cucumber/wiki/Given-When-Then, 2016 (accessed December 7, 2016).

[2] Statcounter globalstats. https://gs.statcounter.com/os-market-share/mobile/worldwide, Accessed: 20120-03-19.

[3] Android developer. https://developer.android.com, Accessed: 2018-12-20.

[4] Genius maps: Offline gps navigator. https://play.google.com/store/apps/details?id=hr.mireo.arthur, Accessed: 2019.

[5] Gps offline navigation route maps & directions. https://play.google.com/store/apps/details?id=com.offline.routemaps.gps.directionfinder.freer, Accessed: 2019.

[6] Osmand. https://play.google.com/store/apps/details?id=net.osmand, Accessed: 2019.

[7] Voice gps navigation: Live driving direction. https://play.google.com/store/apps/details?id=com.voicenavigation.gps.driving.directions, Accessed: 2019.

[8] Google scholar. https://scholar.google.com, Accessed: 2019-08-10.

[9] Samsung health. https://health.apps.samsung.com, Accessed: 2019-08-10.

[10] Start tool home page. http://lapes.dc.ufscar.br/tools/start_tool, Accessed: 2019-08-10.

[11] Github. https://github.com, Accessed: 2019-09-05.

[12] Logcat. https://developer.android.com/studio/command-line/logcat, Accessed: 2019-09-05.

[13] Android debug bridge. https://developer.android.com/studio/command-line/adb.html, Accessed: 2019-09-09.

[14] Google play store. https://play.google.com, Accessed: 2019-09-09.

[15] Pairwise independent combinatorial tool (pict). https://github.com/microsoft/pict, Accessed: 2019-09-23.

[16] Microsoft home page. https://www.microsoft.com, Accessed: 2020-01-15.

[17] Android emulator. https://developer.android.com/studio/run/emulator, Accessed: 2020-02-01.

[18] Arko flow. https://flowsupport.akvo.org/article/show/33694-calibrate-your-gps, Accessed: 2020-02-11.

[19] Gps.gov. https://www.gps.gov/systems/gps/performance/accuracy, Accessed: 2020-02-11.

[20] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, UK, 1999. Springer-Verlag.

[21] Christoffer Quist Adamsen, Gianluca Mezzetti, and Anders Møller. Systematic execution of android test suites in adverse conditions. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 83–93, New York, NY, USA, 2015. ACM.

[22] Diego R. Almeida, Patrícia D. L. Machado, and Wilkerson L. Andrade. Testing tools for android context-aware applications: a systematic mapping. *Journal of the Brazilian Computer Society*, 25, December 2019.

[23] Diego R. Almeida, Patrícia D. L. Machado, and Wilkerson L. Andrade. Context-aware android applications testing. *SBES'20: 34st Brazilian Symposium on Software Engineering*, 2020.

[24] Diego R. Almeida, Patrícia D. L. Machado, and Wilkerson L. Andrade. Enviar - environment data simulator. *SBES'20: 34st Brazilian Symposium on Software Engineering, Tools Track*, 2020.

[25] D. Amalfitano, A. R. Fasolino, P. Tramontana, and N. Amatucci. Considering context events in event-based testing of mobile applications. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 126–133, March 2013.

[26] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and G. Imparato. A toolset for gui testing of android applications. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 650–653, Sep. 2012.

[27] D. Amalfitano, A. R. Fasolino, P. Tramontana, B. D. Ta, and A. M. Memon. Mobiguitar: Automated model-based testing of mobile apps. *IEEE Software*, 32(5):53–59, Sept 2015.

[28] Domenico Amalfitano, Nicola Amatucci, Anna Rita Fasolino, and Porfirio Tramontana. Agrippin: A novel search based testing technique for android applications. In *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, DeMobile 2015, pages 5–12, New York, NY, USA, 2015. ACM.

[29] Domenico Amalfitano, Anna Rita Fasolino, and Porfirio Tramontana. A gui crawling-based technique for android mobile application testing. In *Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, ICSTW '11, pages 252–261, Washington, DC, USA, 2011. IEEE Computer Society.

[30] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. Using gui ripping for automated testing of android

applications. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 258–261, New York, NY, USA, 2012. ACM.

[31] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Bryan Dzung Ta, and Atif M. Memon. Mobiguitar – a tool for automated model-based testing of mobile apps. *IEEE Software*, NN(N):NN–NN, 2014.

[32] Amit Seal Ami, Md. Mehedi Hasan, Md. Rayhanur Rahman, and Kazi Sakib. Mobicomonkey: Context testing of android apps. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '18, pages 76–79, New York, NY, USA, 2018. ACM.

[33] Saswat Anand, Mayur Naik, Mary Jean Harrold, and Hongseok Yang. Automated concolic testing of smartphone apps. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 59:1–59:11, New York, NY, USA, 2012. ACM.

[34] R. Anbunathan and A. Basu. Data driven architecture based automated test generation for android mobile. In *2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pages 1–5, Dec 2015.

[35] R. Anbunathan and A. Basu. Automation framework for test script generation for android mobile. In *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pages 1914–1918, May 2017.

[36] Tanzirul Azim and Iulian Neamtiu. Targeted and depth-first exploration for systematic testing of android apps. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pages 641–660, New York, NY, USA, 2013. ACM.

[37] Tanzirul Azim and Iulian Neamtiu. Targeted and depth-first exploration for systematic testing of android apps. *SIGPLAN Not.*, 48(10):641–660, October 2013.

[38] Z. Babaei, A. M. Rahmani, and A. Rezaei. Real-time reusable event-driven architecture for context aware systems. In *2016 24th Iranian Conference on Electrical Engineering (ICEE)*, pages 294–299, May 2016.

[39] Rex Black. *Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional*. John Wiley & Sons, Inc., New York, NY, USA, 2007.

[40] Nataniel P. Borges Jr., Jenny Hotzkow, and Andreas Zeller. Droidmate-2: A platform for android test generation. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 916–919, New York, NY, USA, 2018. ACM.

[41] P. J. Brown. The stick-e document: a framework for creating context-aware applications. In *Proceedings of EP'96, Palo Alto*, pages 182–196. also published in it EP–odd, January 1996.

[42] Kevin Burr and William Young. Combinatorial test techniques: Table-based automation, test generation and code coverage. In *Proceedings of the Intl. Conf. on Software Testing Analysis and Review*, pages 503–513. West, 1998.

[43] C. Cao, C. Meng, H. Ge, P. Yu, and X. Ma. Xdroid: Testing android apps with dependency injection. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 214–223, July 2017.

[44] J. Chen, G. Han, S. Guo, and W. Diao. Fragdroid: Automated user interface interaction with activity and fragment analysis in android applications. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 398–409, June 2018.

[45] Wontae Choi, George Necula, and Koushik Sen. Guided gui testing of android apps with minimal restart and approximate learning. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pages 623–640, New York, NY, USA, 2013. ACM.

[46] Shauvik Roy Choudhary, Alessandra Gorla, and Alessandro Orso. Automated test input generation for android: Are we there yet? (e). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 429–440, Washington, DC, USA, 2015. IEEE Computer Society.

[47] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. *IEEE Software*, 13(5):83–88, Sep. 1996.

[48] T.D. Cook and D.T. Campbell. Quasi-experimentation: Design and analysis issue for field settings. 1979.

[49] Riccardo Coppola, Emanuele Raffero, and Marco Torchiano. Automated mobile ui test fragility: An exploratory assessment study on android. In *Proceedings of the 2Nd International Workshop on User Interface Test Automation*, INTUITEST 2016, pages 11–20, New York, NY, USA, 2016. ACM.

[50] Guilherme de Cleva Farto and Andre Takeshi Endo. Reuse of model-based tests in mobile apps. In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, SBES'17, pages 184–193, New York, NY, USA, 2017. ACM.

[51] David Franklin and Joshua Flachsbart. All gadget and no representation makes jack a dull environment. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 155–160, 1998.

[52] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. Reran: Timing- and touch-sensitive record and replay for android. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 72–81, Piscataway, NJ, USA, 2013. IEEE Press.

[53] Tobias Griebe and Volker Gruhn. A model-based approach to test automation for context-aware mobile applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 420–427, New York, NY, USA, 2014. ACM.

[54] Tobias Griebe, Marc Hesenius, and Volker Gruhn. Towards automated ui-tests for sensor-based mobile applications. In *Intelligent Software Methodologies, Tools and*

Techniques - 14th International Conference, SoMeT 2015, Naples, Italy, September 15-17, 2015. Proceedings, pages 3–17, 2015.

[55] F. Gross, G. Fraser, and A. Zeller. Exsyst: Search-based gui testing. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1423–1426, June 2012.

[56] Alejandro Sánchez Guinea, Grégory Nain, and Yves Le Traon. A systematic review on the engineering of software for ubiquitous systems. *Journal of Systems and Software*, 118:251 – 276, 2016.

[57] Shuai Hao, Bin Liu, Suman Nath, William G.J. Halfond, and Ramesh Govindan. Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 204–217, New York, NY, USA, 2014. ACM.

[58] L. V. Haoyin. Automatic android application gui testing - a random walk approach. In *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 72–76, March 2017.

[59] Ville-Veikko Helppi. Calabash 101 - basics, getting started, and advanced tips. http://bitbar.com/new-ebook-calabash-101-basics-getting-started-and-advanced-tips, 2016. [Online; accessed March, 17, 2017].

[60] Clemens Holzmann, Dustin Steiner, Andreas Riegler, and Christian Grossauer. An android toolkit for supporting field studies on mobile devices. In *Proceedings of the 16th International Conference on Mobile and Ubiquitous Multimedia*, MUM '17, pages 473–479, New York, NY, USA, 2017. ACM.

[61] Cuixiong Hu and Iulian Neamtiu. Automating gui testing for android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 77–83, New York, NY, USA, 2011. ACM.

[62] Gang Hu, Xinhao Yuan, Yang Tang, and Junfeng Yang. Efficiently, effectively detecting mobile app bugs with appdoctor. In *Proceedings of the Ninth European Con-*

ference on Computer Systems, EuroSys '14, pages 18:1–18:15, New York, NY, USA, 2014. ACM.

[63] Yongjian Hu, Tanzirul Azim, and Iulian Neamtiu. Improving the android development lifecycle with the valera record-and-replay approach. In *Proceedings of the 3rd International Workshop on Mobile Development Lifecycle*, MobileDeLi 2015, pages 7–8, New York, NY, USA, 2015. ACM.

[64] Yongjian Hu, Tanzirul Azim, and Iulian Neamtiu. Versatile yet lightweight record-and-replay for android. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2015, pages 349–366, New York, NY, USA, 2015. ACM.

[65] Yongjian Hu and Iulian Neamtiu. Fuzzy and cross-app replay for smartphone apps. In *Proceedings of the 11th International Workshop on Automation of Software Test*, AST '16, pages 50–56, New York, NY, USA, 2016. ACM.

[66] Yongjian Hu and Iulian Neamtiu. Valera: An effective and efficient record-and-replay tool for android. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*, MOBILESoft '16, pages 285–286, New York, NY, USA, 2016. ACM.

[67] Huang-Ming Huang, Terry Tidwell, Christopher Gill, Chenyang Lu, Xiuyu Gao, and Shirley Dyke. Cyber-physical systems for real-time hybrid structural testing: A case study. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '10, page 69–78, New York, NY, USA, 2010. Association for Computing Machinery.

[68] R. Hull, P. Neaves, and J. Bedford-Roberts. Towards situated computing. In *Digest of Papers. First International Symposium on Wearable Computers*, pages 146–153, Oct 1997.

[69] Gennaro Imparato. A combined technique of gui ripping and input perturbation testing for android apps. In *Proceedings of the 37th International Conference on Software*

Engineering - Volume 2, ICSE '15, pages 760–762, Piscataway, NJ, USA, 2015. IEEE Press.

[70] Konrad Jamrozik and Andreas Zeller. Droidmate: A robust and extensible test generator for android. In *Proceedings of the International Conference on Mobile Software Engineering and Systems*, MOBILESoft '16, pages 293–294, New York, NY, USA, 2016. ACM.

[71] Casper S. Jensen, Mukul R. Prasad, and Anders Møller. Automated testing with targeted event sequence generation. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ISSTA 2013, pages 67–77, New York, NY, USA, 2013. ACM.

[72] Ajay Kumar Jha, Sunghee Lee, and Woo Jin Lee. Characterizing android-specific crash bugs. In *Proceedings of the 6th International Conference on Mobile Software Engineering and Systems*, MOBILESoft '19, page 111–122. IEEE Press, 2019.

[73] Jouko Kaasila, Denzil Ferreira, Vassilis Kostakos, and Timo Ojala. Testdroid: Automated remote ui testing on android. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, MUM '12, pages 28:1–28:4, New York, NY, USA, 2012. ACM.

[74] Maria Kechagia, Dimitris Mitropoulos, and Diomidis Spinellis. Charting the api minefield using software telemetry data. *Empirical Softw. Engg.*, 20(6):1785–1830, December 2015.

[75] B. Kitchenham and S Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.

[76] Yavuz Koroglu and Alper Sen. Tcm: Test case mutation to improve crash detection in android. In Alessandra Russo and Andy Schürr, editors, *Fundamental Approaches to Software Engineering*, pages 264–280, Cham, 2018. Springer International Publishing.

[77] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering*, 30(6):418–421, June 2004.

[78] Wing Lam, Zhengkai Wu, Dengfeng Li, Wenyu Wang, Haibing Zheng, Hui Luo, Peng Yan, Yuetang Deng, and Tao Xie. Record and replay for android: Are we there yet in industrial cases? In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 854–859, New York, NY, USA, 2017. ACM.

[79] Ang Li, Zishan Qin, Mingsong Chen, and Jing Liu. Adautomation: An activity diagram based automated gui testing framework for smartphone applications. In *Proceedings of the 2014 Eighth International Conference on Software Security and Reliability*, SERE '14, pages 68–77, Washington, DC, USA, 2014. IEEE Computer Society.

[80] X. Li, Y. Jiang, Y. Liu, C. Xu, X. Ma, and J. Lu. User guided automation for testing mobile apps. In *2014 21st Asia-Pacific Software Engineering Conference*, volume 1, pages 27–34, Dec 2014.

[81] Y. Li, Z. Yang, Y. Guo, and X. Chen. Droidbot: A lightweight ui-guided test input generator for android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 23–26, May 2017.

[82] Y. D. Lin, J. F. Rojas, E. T. H. Chu, and Y. C. Lai. On the accuracy, efficiency, and reusability of automated test oracles for android devices. *IEEE Transactions on Software Engineering*, 40(10):957–970, Oct 2014.

[83] Mario Linares-Vásquez. Enabling testing of android apps. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 763–765, Piscataway, NJ, USA, 2015. IEEE Press.

[84] M. Linares-Vásquez, C. Bernal-Cardenas, K. Moran, and D. Poshyvanyk. How do developers test android applications? In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 613–622, Sept 2017.

[85] C. H. Liu, C. Y. Lu, S. J. Cheng, K. Y. Chang, Y. C. Hsiao, and W. M. Chu. Capture-replay testing for android applications. In *2014 International Symposium on Computer, Consumer and Control*, pages 1129–1132, June 2014.

[86] Yepang Liu and Chang Xu. Veridroid: Automating android application verification. In *Proceedings of the 2013 Middleware Doctoral Symposium*, MDS '13, pages 5:1–5:6, New York, NY, USA, 2013. ACM.

[87] Shan Lu, Soyeon Park, Eunsoo Seo, and Yuanyuan Zhou. Learning from mistakes: A comprehensive study on real world concurrency bug characteristics. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XIII, pages 329–339, New York, NY, USA, 2008. ACM.

[88] Chu Luo, Jorge Goncalves, Eduardo Velloso, and Vassilis Kostakos. A survey of context simulation for testing mobile context-aware applications. *ACM Comput. Surv.*, 53(1), February 2020.

[89] Chu Luo, Miikka Kuutila, Simon Klakegg, Denzil Ferreira, Huber Flores, Jorge Goncalves, Mika Mäntylä, and Vassilis Kostakos. Testaware: A laboratory-oriented testing tool for mobile context-aware applications. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 1(3):80:1–80:29, September 2017.

[90] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. Dynodroid: An input generation system for android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 224–234, New York, NY, USA, 2013. ACM.

[91] Riyadh Mahmood, Nariman Mirzaei, and Sam Malek. Evodroid: Segmented evolutionary testing of android apps. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 599–609, New York, NY, USA, 2014. ACM.

[92] Amiya Kumar Maji, Kangli Hao, Salmin Sultana, and Saurabh Bagchi. Characterizing failures in mobile oses: A case study with android and symbian. In *Proceedings of*

the 2010 IEEE 21st International Symposium on Software Reliability Engineering, ISSRE '10, pages 249–258, Washington, DC, USA, 2010. IEEE Computer Society.

[93] Ke Mao, Mark Harman, and Yue Jia. Sapienz: Multi-objective automated testing for android applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA 2016, pages 94–105, New York, NY, USA, 2016. ACM.

[94] L. Mariani, M. Pezze, O. Riganelli, and M. Santoro. Autoblacktest: Automatic black-box testing of interactive applications. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 81–90, April 2012.

[95] Santiago Matalonga, Felyppe Rodrigues, and Guilherme Travassos. Challenges in testing context aware software systems. In *Brazilian Conference on Software: Theory and Practice*, CBSOFT '15, pages 51–60, 09 2015.

[96] Santiago Matalonga, Felyppe Rodrigues, and Guilherme H. Travassos. Matching context aware software testing design techniques to iso/iec/ieee 29119. In Terry Rout, Rory V. O'Connor, and Alec Dorling, editors, *Software Process Improvement and Capability Determination*, pages 33–44, Cham, 2015. Springer International Publishing.

[97] Santiago Matalonga, Felyppe Rodrigues, and Guilherme Horta Travassos. Characterizing testing methods for context-aware software systems: Results from a quasi-systematic literature review. *Journal of Systems and Software*, 131:1 – 21, 2017.

[98] P. McAfee, M. Wiem Mkaouer, and D. E. Krutz. Cate: Concolic android testing using java pathfinder for android applications. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 213–214, May 2017.

[99] P. Mehlitz, O. Tkachuk, and M. Ujma. Jpf-awt: Model checking gui applications. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 584–587, Nov 2011.

[100] A. Memon, I. Banerjee, and A. Nagarajan. Gui ripping: reverse engineering of graphical user interfaces for testing. In *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.*, pages 260–269, Nov 2003.

[101] Atif M. Memon, Martha E. Pollack, and Mary Lou Soffa. Automated test oracles for guis. In *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering: Twenty-first Century Applications*, SIGSOFT '00/FSE-8, pages 30–39, New York, NY, USA, 2000. ACM.

[102] Zhanshuai Meng, Yanyan Jiang, and Chang Xu. Facilitating reusable and scalable automated testing and analysis for android apps. In *Proceedings of the 7th Asia-Pacific Symposium on Internetware*, Internetware '15, pages 166–175, New York, NY, USA, 2015. ACM.

[103] A. M. Mirza and M. N. A. Khan. An automated functional testing framework for context-aware applications. *IEEE Access*, 6:46568–46583, 2018.

[104] N. Mirzaei, H. Bagheri, R. Mahmood, and S. Malek. Sig-droid: Automated system input generation for android applications. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, pages 461–471, Nov 2015.

[105] Nariman Mirzaei, Joshua Garcia, Hamid Bagheri, Alireza Sadeghi, and Sam Malek. Reducing combinatorics in gui testing of android applications. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 559–570, New York, NY, USA, 2016. ACM.

[106] K. Moran, M. Linares-Vasquez, C. Bernal-Cardenas, C. Vendome, and D. Poshyvanyk. Crashscope: A practical tool for automated testing of android applications. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 15–18, May 2017.

[107] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, C. Vendome, and D. Poshyvanyk. Automatically discovering, reporting and reproducing android application crashes. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 33–44, April 2016.

[108] I. C. Morgado and A. C. R. Paiva. The impact tool: Testing ui patterns on mobile applications. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 876–881, Nov 2015.

[109] Ricardo Morla and Nigel Davies. Evaluating a location-based application: A hybrid test and simulation environment. *IEEE Pervasive Computing*, 3(3):48–56, July 2004.

[110] Henry Muccini, Antonio Di Francesco, and Patrizio Esposito. Software testing of mobile applications: Challenges and future research directions. In *Proceedings of the 7th International Workshop on Automation of Software Test*, AST '12, pages 29–35, Piscataway, NJ, USA, 2012. IEEE Press.

[111] Nelson Mariano Leite Neto, Patrícia Vilain, and Ronaldo dos Santos Mello. Segen: Generation of test cases for selenium and selendroid. In *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '16, pages 433–442, New York, NY, USA, 2016. ACM.

[112] Cu D. Nguyen, Alessandro Marchetto, and Paolo Tonella. Combining model-based and combinatorial testing for effective test case generation. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, pages 100–110, New York, NY, USA, 2012. ACM.

[113] Eleanor O'Neill, Owen Conlan, and David Lewis. Situation-based testing for pervasive computing environments. *Pervasive Mob. Comput.*, 9(1):76–97, February 2013.

[114] Priyam Patel, Gokul Srinivasan, Sydur Rahaman, and Iulian Neamtiu. On the effectiveness of random testing for android: Or how i learned to stop worrying and love the monkey. In *Proceedings of the 13th International Workshop on Automation of Software Test*, AST '18, pages 34–37, New York, NY, USA, 2018. ACM.

[115] F. Paulovsky, E. Pavese, and D. Garbervetsky. High-coverage testing of navigation models in android applications. In *2017 IEEE/ACM 12th International Workshop on Automation of Software Testing (AST)*, pages 52–58, May 2017.

[116] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swindon, UK, 2008. BCS Learning & Development Ltd.

[117] C. M. Prathibhan, A. Malini, N. Venkatesh, and K. Sundarakantham. An automated testing framework for testing android mobile applications in the cloud. In *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, pages 1216–1219, May 2014.

[118] Zhengrui Qin, Yutao Tang, Ed Novak, and Qun Li. Mobiplay: A remote execution based record-and-replay tool for mobile applications. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 571–582, New York, NY, USA, 2016. ACM.

[119] Valentim Realinho, Teresa Romão, and A. Eduardo Dias. An event-driven workflow framework to develop context-aware mobile applications. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia*, MUM '12, pages 22:1–22:10, New York, NY, USA, 2012. ACM.

[120] Tom Rodden, Keith Chervest, Nigel Davies, and Alan Dix. Exploiting context in hci design for mobile systems. In *in Workshop on Human Computer Interaction with Mobile Devices*, pages 21–22, 1998.

[121] Ibrahim Anka Salihu and Rosziati Ibrahim. Systematic exploration of android apps' events for automated testing. In *Proceedings of the 14th International Conference on Advances in Mobile Computing and Multi Media*, MoMM '16, pages 50–54, New York, NY, USA, 2016. ACM.

[122] J. Sanders and K. R. Walcott. Tads: Automating device state to android test suite testing. In *Proceedings of the 2018 International Conference on Wireless Networks*, ICWN'18, pages 10–14, 2018.

[123] Aaratee Shrestha, Bettina Biel, Tobias Griebe, and Volker Gruhn. A framework for building and operating context-aware mobile applications. In *Mobile Wireless Middleware, Operating Systems, and Applications - 4th International ICST Conference, Mobilware 2011, London, UK, June 22-24, 2011, Revised Selected Papers*, pages 135–142, 2011.

[124] D. Bernardo Silva, A. T. Endo, M. M. Eler, and V. H. S. Durelli. An analysis of automated tests for mobile android applications. In *2016 XLII Latin American Computing Conference (CLEI)*, pages 1–9, Oct 2016.

[125] Wei Song, Xiangxing Qian, and Jeff Huang. Ehbdroid: Beyond gui testing for android applications. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 27–37, Piscataway, NJ, USA, 2017. IEEE Press.

[126] Ting Su. Fsmdroid: Guided gui testing of android apps. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 689–691, New York, NY, USA, 2016. ACM.

[127] Asmau Usman, Noraini Ibrahim, and Ibrahim Anka Salihu. Comparative study of mobile applications testing techniques for context events. *Advanced Science Letters*, 24(10):7305–7310, 2018.

[128] Asmau Usman, Noraini Ibrahim, and Ibrahim Anka Salihu. Test case generation from android mobile applications focusing on context events. In *Proceedings of the 2018 7th International Conference on Software and Computer Applications*, ICSCA 2018, pages 25–30, New York, NY, USA, 2018. ACM.

[129] Heila van der Merwe, Brink van der Merwe, and Willem Visser. Verifying android applications using java pathfinder. *SIGSOFT Softw. Eng. Notes*, 37(6):1–5, November 2012.

[130] Heila van der Merwe, Brink van der Merwe, and Willem Visser. Execution and property specifications for jpf-android. *SIGSOFT Softw. Eng. Notes*, 39(1):1–5, February 2014.

[131] Vaninha Vieira, Konstantin Holl, and Michael Hassel. A context simulator as testing support for mobile apps. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 535–541, New York, NY, USA, 2015. ACM.

[132] I. K. Villanes, E. A. B. Costa, and A. C. Dias-Neto. Automated mobile testing as

a service (am-taas). In *2015 IEEE World Congress on Services*, pages 79–86, June 2015.

[133] Isabel K. Villanes, Silvia M. Ascate, Josias Gomes, and Arilo Claudio Dias-Neto. What are software engineers asking about android testing on stack overflow? In *Proceedings of the 31st Brazilian Symposium on Software Engineering*, SBES'17, pages 104–113, New York, NY, USA, 2017. ACM.

[134] Huai Wang, W. K. Chan, and T. H. Tse. Improving the effectiveness of testing pervasive software via context diversity. *ACM Trans. Auton. Adapt. Syst.*, 9(2), July 2014.

[135] P. Wang, B. Liang, W. You, J. Li, and W. Shi. Automatic android gui traversal with high coverage. In *2014 Fourth International Conference on Communication Systems and Network Technologies*, pages 1161–1166, April 2014.

[136] Wenyu Wang, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng, and Tao Xie. An empirical study of android test generation tools in industrial cases. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 738–748, New York, NY, USA, 2018. ACM.

[137] Zhimin Wang, Sebastian Elbaum, and David S. Rosenblum. Automated generation of context-aware tests. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 406–415, Washington, DC, USA, 2007. IEEE Computer Society.

[138] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, Oct 1997.

[139] Anthony I. Wasserman. Software engineering issues for mobile application development. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, pages 397–400, New York, NY, USA, 2010. ACM.

[140] H. L. Wen, C. H. Lin, T. H. Hsieh, and C. Z. Yang. Pats: A parallel gui testing framework for android applications. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 2, pages 210–215, July 2015.

[141] Lee White and Husain Almezen. Generating test cases for gui responsibilities using complete interaction sequences. In *Proceedings of the 11th International Symposium on Software Reliability Engineering*, ISSRE '00, pages 110–, Washington, DC, USA, 2000. IEEE Computer Society.

[142] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[143] Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, and Sunghun Kim. Crashlocator: Locating crashing faults based on crash stacks. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA 2014, page 204–214, New York, NY, USA, 2014. Association for Computing Machinery.

[144] Jiwei Yan, Linjie Pan, Yaqi Li, Jun Yan, and Jian Zhang. Land: A user-friendly and customizable test generation tool for android apps. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2018, pages 360–363, New York, NY, USA, 2018. ACM.

[145] Wei Yang, Mukul R. Prasad, and Tao Xie. A grey-box approach for automated gui-model generation of mobile applications. In *Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering*, FASE'13, pages 250–265, Berlin, Heidelberg, 2013. Springer-Verlag.

[146] Hui Ye, Shaoyin Cheng, Lanbo Zhang, and Fan Jiang. Droidfuzzer: Fuzzing the android apps with intent-filter tag. In *Proceedings of International Conference on Advances in Mobile Computing & Multimedia*, MoMM '13, pages 68:68–68:74, New York, NY, USA, 2013. ACM.

[147] Yu Lei and K. C. Tai. In-parameter-order: a test generation strategy for pairwise testing. In *Proceedings Third IEEE International High-Assurance Systems Engineering Symposium (Cat. No.98EX231)*, pages 254–261, Nov 1998.

[148] Songhui Yue, Songqing Yue, and Randy Smith. A survey of testing context-aware software: Challenges and resolution. In *Proceedings of the International Conference*

on Software Engineering Research and Practice (SERP) 2016, pages 102–108, Las Vegas, NE, USA, 2016. IEEE Computer Society.

[149] R. N. Zaeem, M. R. Prasad, and S. Khurshid. Automated generation of oracles for testing user-interaction features of mobile apps. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, pages 183–192, March 2014.

[150] Samer Zein, Norsaremah Salleh, and John Grundy. A systematic mapping study of mobile application testing techniques. *J. Syst. Softw.*, 117(C):334–356, July 2016.

[151] Xia Zeng, Dengfeng Li, Wujie Zheng, Fan Xia, Yuetang Deng, Wing Lam, Wei Yang, and Tao Xie. Automated test input generation for android: Are we really there yet in an industrial case? In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 987–992, New York, NY, USA, 2016. ACM.

[152] Y. Zhauniarovich, A. Philippov, O. Gadyatskaya, B. Crispo, and F. Massacci. Towards black box testing of android apps. In *2015 10th International Conference on Availability, Reliability and Security*, pages 501–510, Aug 2015.

[153] D. Zun, T. Qi, and L. Chen. Research on automated testing framework for multi-platform mobile applications. In *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pages 82–87, Aug 2016.

Appendix A

PICT Rules

Table A.1: Order 1 test rules

```
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} THEN [First_Event] <> "GPS_ON";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_OFF";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} THEN [First_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S1", "S2", "S7", "S8"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} THEN [First_Event] <> "INTERNET_ON";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} THEN [First_Event] <> "INTERNET_OFF";
```

Table A.2: Order 2 test rules

```
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} THEN [First_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" THEN
[Second_Event] <> "GPS_ON";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" THEN [Second_Event] <> "GPS_OFF";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} THEN [First_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" THEN [Second_Event]
<> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" THEN [Second_Event]
<> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S1", "S2", "S7", "S8"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" THEN [Second_Event] <>
"GPS_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" THEN
[Second_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" THEN [Second_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} THEN [First_Event] <> "INTERNET_ON";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" THEN [Second_Event] <>
"INTERNET_ON";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} THEN [First_Event] <> "INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" THEN [Second_Event] <>
"INTERNET_OFF";
IF [First_event] = "GPS_ON" THEN [Second_Event] <> "GPS_ON";
IF [First_event] = "GPS_OFF" THEN [Second_Event] <> "GPS_OFF";
IF [First_event] = "SIMULATE_LONG_BACKGROUND" THEN [Second_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [First_event] = "TAKE_A_PICTURE" THEN [Second_Event] <> "TAKE_A_PICTURE";
IF [First_event] = "ORIENTATION_PORTRAIT" THEN [Second_Event] <> "ORIENTATION_PORTRAIT";
IF [First_event] = "ORIENTATION_LANDSCAPE" THEN [Second_Event] <> "ORIENTATION_LANDSCAPE";
IF [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "RECEIVE_CALL";
IF [First_event] = "GPS_CALIBRATED" THEN [Second_Event] <> "GPS_CALIBRATED";
IF [First_event] = "GPS_NOT_CALIBRATED" THEN [Second_Event] <> "GPS_NOT_CALIBRATED";
IF [First_event] = "INTERNET_ON" THEN [Second_Event] <> "INTERNET_ON";
```

```
IF [First_event] = "INTERNET_OFF" THEN [Second_Event] <> "INTERNET_OFF";
IF NOT [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "ACCEPT_CALL";
IF NOT [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "CANCEL_CALL";
```

Table A.3: Order 3 test rules

```
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} THEN [First_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" THEN
[Second_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" AND
[Second_Event] <> "GPS_OFF" THEN [Third_Event] <> "GPS_ON";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" THEN [Second_Event] <> "GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" THEN [Third_Event] <>
"GPS_OFF";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} THEN [First_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" THEN [Second_Event]
<> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" AND [Second_Event] <>
"ORIENTATION_LANDSCAPE"  THEN [Third_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" THEN [Second_Event]
<> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" AND [Second_Event]
<> "ORIENTATION_PORTRAIT" THEN [Third_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S1", "S2", "S7", "S8"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" THEN [Second_Event] <>
"GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" AND [Second_Event] <>
"GPS_NOT_CALIBRATED" THEN [Third_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" THEN
[Second_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" AND
[Second_Event] <> "GPS_CALIBRATED" THEN [Third_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" THEN [Second_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" THEN [Third_Event] <>
"GPS_CALIBRATED";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} THEN [First_Event] <> "INTERNET_ON";
```

```
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" THEN [Second_Event] <>
"INTERNET_ON";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" AND [Second_Event] <>
"INTERNET_OFF" THEN [Third_Event] <> "INTERNET_ON";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} THEN [First_Event] <> "INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" THEN [Second_Event] <>
"INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" AND [Second_Event] <>
"INTERNET_ON" THEN [Third_Event] <> "INTERNET_OFF";
IF [First_event] = "GPS_ON" THEN [Second_Event] <> "GPS_ON";
IF [Second_Event] = "GPS_ON" THEN [Third_Event] <> "GPS_ON";
IF [First_event] = "GPS_OFF" THEN [Second_Event] <> "GPS_OFF";
IF [Second_Event] = "GPS_OFF" THEN [Third_Event] <> "GPS_OFF";
IF [First_event] = "SIMULATE_LONG_BACKGROUND" THEN [Second_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [Second_Event] = "SIMULATE_LONG_BACKGROUND" THEN [Third_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [First_event] = "TAKE_A_PICTURE" THEN [Second_Event] <> "TAKE_A_PICTURE";
IF [Second_Event] = "TAKE_A_PICTURE" THEN [Third_Event] <> "TAKE_A_PICTURE";
IF [First_event] = "ORIENTATION_PORTRAIT" THEN [Second_Event] <> "ORIENTATION_PORTRAIT";
IF [Second_Event] = "ORIENTATION_PORTRAIT" THEN [Third_Event] <> "ORIENTATION_PORTRAIT";
IF [First_event] = "ORIENTATION_LANDSCAPE" THEN [Second_Event] <> "ORIENTATION_LANDSCAPE";
IF [Second_Event] = "ORIENTATION_LANDSCAPE" THEN [Third_Event] <> "ORIENTATION_LANDSCAPE";
IF [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "RECEIVE_CALL";
IF [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "RECEIVE_CALL";
IF [First_event] = "GPS_CALIBRATED" THEN [Second_Event] <> "GPS_CALIBRATED";
IF [Second_Event] = "GPS_CALIBRATED" THEN [Third_Event] <> "GPS_CALIBRATED";
IF [First_event] = "GPS_NOT_CALIBRATED" THEN [Second_Event] <> "GPS_NOT_CALIBRATED";
IF [Second_Event] = "GPS_NOT_CALIBRATED" THEN [Third_Event] <> "GPS_NOT_CALIBRATED";
IF [First_event] = "INTERNET_ON" THEN [Second_Event] <> "INTERNET_ON";
IF [Second_Event] = "INTERNET_ON" THEN [Third_Event] <> "INTERNET_ON";
IF [First_event] = "INTERNET_OFF" THEN [Second_Event] <> "INTERNET_OFF";
IF [Second_Event] = "INTERNET_OFF" THEN [Third_Event] <> "INTERNET_OFF";
IF NOT [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "ACCEPT_CALL";
IF NOT [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "ACCEPT_CALL";
IF NOT [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "CANCEL_CALL";
IF NOT [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "CANCEL_CALL";
```

Table A.4: Order 4 test rules

```
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} THEN [First_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" THEN
[Second_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" AND
[Second_Event] <> "GPS_OFF" THEN [Third_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" AND
[Second_Event] <> "GPS_OFF" AND [Third_Event] <> "GPS_OFF" THEN [Fourth_Event] <> "GPS_ON";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" THEN [Second_Event] <> "GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" THEN [Third_Event] <>
"GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" AND [Third_Event] <>
"GPS_ON" THEN [Fourth_Event] <> "GPS_OFF";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} THEN [First_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" THEN [Second_Event]
<> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" AND [Second_Event] <>
"ORIENTATION_LANDSCAPE"  THEN [Third_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" AND [Second_Event] <>
"ORIENTATION_LANDSCAPE"  AND [Third_Event] <> "ORIENTATION_LANDSCAPE"  THEN [Fourth_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" THEN [Second_Event]
<> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" AND [Second_Event]
<> "ORIENTATION_PORTRAIT" THEN [Third_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" AND [Second_Event]
<> "ORIENTATION_PORTRAIT" AND [Third_Event] <> "ORIENTATION_PORTRAIT" THEN [Fourth_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S1", "S2", "S7", "S8"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" THEN [Second_Event] <>
"GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" AND [Second_Event] <>
"GPS_NOT_CALIBRATED" THEN [Third_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" AND [Second_Event] <>
"GPS_NOT_CALIBRATED" AND [Third_Event] <> "GPS_NOT_CALIBRATED" THEN [Fourth_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "GPS_NOT_CALIBRATED";
```

```
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" THEN
[Second_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" AND
[Second_Event] <> "GPS_CALIBRATED" THEN [Third_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" AND
[Second_Event] <> "GPS_CALIBRATED" AND [Third_Event] <> "GPS_CALIBRATED" THEN [Fourth_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" THEN [Second_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" THEN [Third_Event] <>
"GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" AND [Third_Event] <>
"GPS_ON" THEN [Fourth_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} THEN [First_Event] <> "INTERNET_ON";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" THEN [Second_Event] <>
"INTERNET_ON";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" AND [Second_Event] <>
"INTERNET_OFF" THEN [Third_Event] <> "INTERNET_ON";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" AND [Second_Event] <>
"INTERNET_OFF" AND [Third_Event] <> "INTERNET_OFF" THEN [Fourth_Event] <> "INTERNET_ON";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} THEN [First_Event] <> "INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" THEN [Second_Event] <>
"INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" AND [Second_Event] <>
"INTERNET_ON" THEN [Third_Event] <> "INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" AND [Second_Event] <>
"INTERNET_ON" AND [Third_Event] <> "INTERNET_ON" THEN [Fourth_Event] <> "INTERNET_OFF";
IF [First_event] = "GPS_ON" THEN [Second_Event] <> "GPS_ON";
IF [Second_Event] = "GPS_ON" THEN [Third_Event] <> "GPS_ON";
IF [Third_Event] = "GPS_ON" THEN [Fourth_Event] <> "GPS_ON";
IF [First_event] = "GPS_OFF" THEN [Second_Event] <> "GPS_OFF";
IF [Second_Event] = "GPS_OFF" THEN [Third_Event] <> "GPS_OFF";
IF [Third_Event] = "GPS_OFF" THEN [Fourth_Event] <> "GPS_OFF";
IF [First_event] = "SIMULATE_LONG_BACKGROUND" THEN [Second_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [Second_Event] = "SIMULATE_LONG_BACKGROUND" THEN [Third_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [Third_Event] = "SIMULATE_LONG_BACKGROUND" THEN [Fourth_Event] <> "SIMULATE_LONG_BACKGROUND";
```

```
IF [First_event] = "TAKE_A_PICTURE" THEN [Second_Event] <> "TAKE_A_PICTURE";
IF [Second_Event] = "TAKE_A_PICTURE" THEN [Third_Event] <> "TAKE_A_PICTURE";
IF [Third_Event] = "TAKE_A_PICTURE" THEN [Fourth_Event] <> "TAKE_A_PICTURE";
IF [First_event] = "ORIENTATION_PORTRAIT" THEN [Second_Event] <> "ORIENTATION_PORTRAIT";
IF [Second_Event] = "ORIENTATION_PORTRAIT" THEN [Third_Event] <> "ORIENTATION_PORTRAIT";
IF [Third_Event] = "ORIENTATION_PORTRAIT" THEN [Fourth_Event] <> "ORIENTATION_PORTRAIT";
IF [First_event] = "ORIENTATION_LANDSCAPE" THEN [Second_Event] <> "ORIENTATION_LANDSCAPE";
IF [Second_Event] = "ORIENTATION_LANDSCAPE" THEN [Third_Event] <> "ORIENTATION_LANDSCAPE";
IF [Third_Event] = "ORIENTATION_LANDSCAPE" THEN [Fourth_Event] <> "ORIENTATION_LANDSCAPE";
IF [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "RECEIVE_CALL";
IF [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "RECEIVE_CALL";
IF [Third_Event] = "RECEIVE_CALL" THEN [Fourth_Event] <> "RECEIVE_CALL";
IF [First_event] = "GPS_CALIBRATED" THEN [Second_Event] <> "GPS_CALIBRATED";
IF [Second_Event] = "GPS_CALIBRATED" THEN [Third_Event] <> "GPS_CALIBRATED";
IF [Third_Event] = "GPS_CALIBRATED" THEN [Fourth_Event] <> "GPS_CALIBRATED";
IF [First_event] = "GPS_NOT_CALIBRATED" THEN [Second_Event] <> "GPS_NOT_CALIBRATED";
IF [Second_Event] = "GPS_NOT_CALIBRATED" THEN [Third_Event] <> "GPS_NOT_CALIBRATED";
IF [Third_Event] = "GPS_NOT_CALIBRATED" THEN [Fourth_Event] <> "GPS_NOT_CALIBRATED";
IF [First_event] = "INTERNET_ON" THEN [Second_Event] <> "INTERNET_ON";
IF [Second_Event] = "INTERNET_ON" THEN [Third_Event] <> "INTERNET_ON";
IF [Third_Event] = "INTERNET_ON" THEN [Fourth_Event] <> "INTERNET_ON";
IF [First_event] = "INTERNET_OFF" THEN [Second_Event] <> "INTERNET_OFF";
IF [Second_Event] = "INTERNET_OFF" THEN [Third_Event] <> "INTERNET_OFF";
IF [Third_Event] = "INTERNET_OFF" THEN [Fourth_Event] <> "INTERNET_OFF";
IF NOT [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "ACCEPT_CALL";
IF NOT [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "ACCEPT_CALL";
IF NOT [Third_Event] = "RECEIVE_CALL" THEN [Fourth_Event] <> "ACCEPT_CALL";
IF NOT [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "CANCEL_CALL";
IF NOT [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "CANCEL_CALL";
IF NOT [Third_Event] = "RECEIVE_CALL" THEN [Fourth_Event] <> "CANCEL_CALL";
```

Table A.5: Order 5 test rules

```
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} THEN [First_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" THEN
[Second_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" AND
[Second_Event] <> "GPS_OFF" THEN [Third_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" AND
[Second_Event] <> "GPS_OFF" AND [Third_Event] <> "GPS_OFF" THEN [Fourth_Event] <> "GPS_ON";
IF [Setup] in {"S1", "S2", "S3", "S4", "S6", "S7", "S8", "S9", "S10"} AND [First_Event] <> "GPS_OFF" AND
[Second_Event] <> "GPS_OFF" AND [Third_Event] <> "GPS_OFF" AND [Fourth_Event] <> "GPS_OFF" THEN [Fifth_Event] <>
"GPS_ON";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" THEN [Second_Event] <> "GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" THEN [Third_Event] <>
"GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" AND [Third_Event] <>
"GPS_ON" THEN [Fourth_Event] <> "GPS_OFF";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" AND [Third_Event] <>
"GPS_ON" AND [Fourth_Event] <> "GPS_ON" THEN [Fifth_Event] <> "GPS_OFF";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} THEN [First_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" THEN [Second_Event]
<> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" AND [Second_Event] <>
"ORIENTATION_LANDSCAPE"  THEN [Third_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" AND [Second_Event] <>
"ORIENTATION_LANDSCAPE"  AND [Third_Event] <> "ORIENTATION_LANDSCAPE" THEN [Fourth_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S1", "S2", "S3", "S4", "S5", "S6"} AND [First_Event] <> "ORIENTATION_LANDSCAPE" AND [Second_Event] <>
"ORIENTATION_LANDSCAPE"  AND [Third_Event] <> "ORIENTATION_LANDSCAPE" AND [Fourth_Event] <> "ORIENTATION_LANDSCAPE"
THEN [Fifth_Event] <> "ORIENTATION_PORTRAIT";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" THEN [Second_Event]
<> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" AND [Second_Event]
<> "ORIENTATION_PORTRAIT" THEN [Third_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" AND [Second_Event]
<> "ORIENTATION_PORTRAIT" AND [Third_Event] <> "ORIENTATION_PORTRAIT" THEN [Fourth_Event] <> "ORIENTATION_LANDSCAPE";
```

```
IF [Setup] in {"S7", "S8", "S9", "S10", "S11", "S12"} AND [First_Event] <> "ORIENTATION_PORTRAIT" AND [Second_Event]
<> "ORIENTATION_PORTRAIT" AND [Third_Event] <> "ORIENTATION_PORTRAIT" AND [Fourth_Event] <> "ORIENTATION_PORTRAIT"
THEN [Fifth_Event] <> "ORIENTATION_LANDSCAPE";
IF [Setup] in {"S1", "S2", "S7", "S8"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" THEN [Second_Event] <>
"GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" AND [Second_Event] <>
"GPS_NOT_CALIBRATED" THEN [Third_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" AND [Second_Event] <>
"GPS_NOT_CALIBRATED" AND [Third_Event] <> "GPS_NOT_CALIBRATED" THEN [Fourth_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S2", "S7", "S8"} AND [First_Event] <> "GPS_NOT_CALIBRATED" AND [Second_Event] <>
"GPS_NOT_CALIBRATED" AND [Third_Event] <> "GPS_NOT_CALIBRATED" AND [Fourth_Event] <> "GPS_NOT_CALIBRATED" THEN
[Fifth_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} THEN [First_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" THEN
[Second_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" AND
[Second_Event] <> "GPS_CALIBRATED" THEN [Third_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" AND
[Second_Event] <> "GPS_CALIBRATED" AND [Third_Event] <> "GPS_CALIBRATED" THEN [Fourth_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S3", "S4", "S5", "S6", "S9", "S10", "S11", "S12"} AND [First_Event] <> "GPS_CALIBRATED" AND
[Second_Event] <> "GPS_CALIBRATED" AND [Third_Event] <> "GPS_CALIBRATED" AND [Fourth_Event] <> "GPS_CALIBRATED" THEN
[Fifth_Event] <> "GPS_NOT_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} THEN [First_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" THEN [Second_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" THEN [Third_Event] <>
"GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" AND [Third_Event] <>
"GPS_ON" THEN [Fourth_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S5", "S11", "S12"} AND [First_Event] <> "GPS_ON" AND [Second_Event] <> "GPS_ON" AND [Third_Event] <>
"GPS_ON" AND [Fourth_Event] <> "GPS_ON" THEN [Third_Event] <> "GPS_CALIBRATED";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} THEN [First_Event] <> "INTERNET_ON";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" THEN [Second_Event] <>
"INTERNET_ON";
```

```
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" AND [Second_Event] <>
"INTERNET_OFF" THEN [Third_Event] <> "INTERNET_ON";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" AND [Second_Event] <>
"INTERNET_OFF" AND [Third_Event] <> "INTERNET_OFF" THEN [Fourth_Event] <> "INTERNET_ON";
IF [Setup] in {"S1", "S3", "S5", "S7", "S10", "S12"} AND [First_Event] <> "INTERNET_OFF" AND [Second_Event] <>
"INTERNET_OFF" AND [Third_Event] <> "INTERNET_OFF" AND [Fourth_Event] <> "INTERNET_OFF" THEN [Fifth_Event] <>
"INTERNET_ON";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} THEN [First_Event] <> "INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" THEN [Second_Event] <>
"INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" AND [Second_Event] <>
"INTERNET_ON" THEN [Third_Event] <> "INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" AND [Second_Event] <>
"INTERNET_ON" AND [Third_Event] <> "INTERNET_ON" THEN [Fourth_Event] <> "INTERNET_OFF";
IF [Setup] in {"S2", "S4", "S6", "S8", "S9", "S11"} AND [First_Event] <> "INTERNET_ON" AND [Second_Event] <>
"INTERNET_ON" AND [Third_Event] <> "INTERNET_ON" AND [Fourth_Event] <> "INTERNET_ON" THEN [Fifth_Event] <>
"INTERNET_OFF";
IF [First_event] = "GPS_ON" THEN [Second_Event] <> "GPS_ON";
IF [Second_Event] = "GPS_ON" THEN [Third_Event] <> "GPS_ON";
IF [Third_Event] = "GPS_ON" THEN [Fourth_Event] <> "GPS_ON";
IF [Fourth_Event] = "GPS_ON" THEN [Fifth_Event] <> "GPS_ON";
IF [First_event] = "GPS_OFF" THEN [Second_Event] <> "GPS_OFF";
IF [Second_Event] = "GPS_OFF" THEN [Third_Event] <> "GPS_OFF";
IF [Third_Event] = "GPS_OFF" THEN [Fourth_Event] <> "GPS_OFF";
IF [Fourth_Event] = "GPS_OFF" THEN [Fifth_Event] <> "GPS_OFF";
IF [First_event] = "SIMULATE_LONG_BACKGROUND" THEN [Second_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [Second_Event] = "SIMULATE_LONG_BACKGROUND" THEN [Third_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [Third_Event] = "SIMULATE_LONG_BACKGROUND" THEN [Fourth_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [Fourth_Event] = "SIMULATE_LONG_BACKGROUND" THEN [Fifth_Event] <> "SIMULATE_LONG_BACKGROUND";
IF [First_event] = "TAKE_A_PICTURE" THEN [Second_Event] <> "TAKE_A_PICTURE";
IF [Second_Event] = "TAKE_A_PICTURE" THEN [Third_Event] <> "TAKE_A_PICTURE";
IF [Third_Event] = "TAKE_A_PICTURE" THEN [Fourth_Event] <> "TAKE_A_PICTURE";
IF [Fourth_Event] = "TAKE_A_PICTURE" THEN [Fifth_Event] <> "TAKE_A_PICTURE";
IF [First_event] = "ORIENTATION_PORTRAIT" THEN [Second_Event] <> "ORIENTATION_PORTRAIT";
IF [Second_Event] = "ORIENTATION_PORTRAIT" THEN [Third_Event] <> "ORIENTATION_PORTRAIT";
```

```
IF [Third_Event] = "ORIENTATION_PORTRAIT" THEN [Fourth_Event] <> "ORIENTATION_PORTRAIT";
IF [Fourth_Event] = "ORIENTATION_PORTRAIT" THEN [Fifth_Event] <> "ORIENTATION_PORTRAIT";
IF [First_event] = "ORIENTATION_LANDSCAPE" THEN [Second_Event] <> "ORIENTATION_LANDSCAPE";
IF [Second_Event] = "ORIENTATION_LANDSCAPE" THEN [Third_Event] <> "ORIENTATION_LANDSCAPE";
IF [Third_Event] = "ORIENTATION_LANDSCAPE" THEN [Fourth_Event] <> "ORIENTATION_LANDSCAPE";
IF [Fourth_Event] = "ORIENTATION_LANDSCAPE" THEN [Fifth_Event] <> "ORIENTATION_LANDSCAPE";
IF [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "RECEIVE_CALL";
IF [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "RECEIVE_CALL";
IF [Third_Event] = "RECEIVE_CALL" THEN [Fourth_Event] <> "RECEIVE_CALL";
IF [Fourth_Event] = "RECEIVE_CALL" THEN [Fifth_Event] <> "RECEIVE_CALL";
IF [First_event] = "GPS_CALIBRATED" THEN [Second_Event] <> "GPS_CALIBRATED";
IF [Second_Event] = "GPS_CALIBRATED" THEN [Third_Event] <> "GPS_CALIBRATED";
IF [Third_Event] = "GPS_CALIBRATED" THEN [Fourth_Event] <> "GPS_CALIBRATED";
IF [Fourth_Event] = "GPS_CALIBRATED" THEN [Fifth_Event] <> "GPS_CALIBRATED";
IF [First_event] = "GPS_NOT_CALIBRATED" THEN [Second_Event] <> "GPS_NOT_CALIBRATED";
IF [Second_Event] = "GPS_NOT_CALIBRATED" THEN [Third_Event] <> "GPS_NOT_CALIBRATED";
IF [Third_Event] = "GPS_NOT_CALIBRATED" THEN [Fourth_Event] <> "GPS_NOT_CALIBRATED";
IF [Fourth_Event] = "GPS_NOT_CALIBRATED" THEN [Fifth_Event] <> "GPS_NOT_CALIBRATED";
IF [First_event] = "INTERNET_ON" THEN [Second_Event] <> "INTERNET_ON";
IF [Second_Event] = "INTERNET_ON" THEN [Third_Event] <> "INTERNET_ON";
IF [Third_Event] = "INTERNET_ON" THEN [Fourth_Event] <> "INTERNET_ON";
IF [Fourth_Event] = "INTERNET_ON" THEN [Fifth_Event] <> "INTERNET_ON";
IF [First_event] = "INTERNET_OFF" THEN [Second_Event] <> "INTERNET_OFF";
IF [Second_Event] = "INTERNET_OFF" THEN [Third_Event] <> "INTERNET_OFF";
IF [Third_Event] = "INTERNET_OFF" THEN [Fourth_Event] <> "INTERNET_OFF";
IF [Fourth_Event] = "INTERNET_OFF" THEN [Fifth_Event] <> "INTERNET_OFF";
IF NOT [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "ACCEPT_CALL";
IF NOT [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "ACCEPT_CALL";
IF NOT [Third_Event] = "RECEIVE_CALL" THEN [Fourth_Event] <> "ACCEPT_CALL";
IF NOT [Fourth_Event] = "RECEIVE_CALL" THEN [Fifth_Event] <> "ACCEPT_CALL";
IF NOT [First_event] = "RECEIVE_CALL" THEN [Second_Event] <> "CANCEL_CALL";
IF NOT [Second_Event] = "RECEIVE_CALL" THEN [Third_Event] <> "CANCEL_CALL";
IF NOT [Third_Event] = "RECEIVE_CALL" THEN [Fourth_Event] <> "CANCEL_CALL";
IF NOT [Fourth_Event] = "RECEIVE_CALL" THEN [Fifth_Event] <> "CANCEL_CALL";
```

Appendix B

Test Cases

Table B.1: Order 1 test cases

	Path	Setup	First Delay	First event
1	Small	S8	NOT_WAIT	ORIENTATION_PORTRAIT
2	Small	S11	WAIT	RECEIVE_CALL
3	Small	S2	WAIT	LONG_BACKGROUND
4	Small	S9	NOT_WAIT	INTERNET_ON
5	Small	S5	NOT_WAIT	INTERNET_OFF
6	Small	S5	WAIT	TAKE_A_PICTURE
7	Small	S10	NOT_WAIT	ORIENTATION_PORTRAIT
8	Small	S6	WAIT	ORIENTATION_LANDSCAPE
9	Small	S9	NOT_WAIT	RECEIVE_CALL
10	Small	S12	WAIT	INTERNET_OFF
11	Small	S7	NOT_WAIT	ORIENTATION_PORTRAIT
12	Small	S4	WAIT	ORIENTATION_LANDSCAPE
13	Small	S8	WAIT	TAKE_A_PICTURE
14	Small	S3	WAIT	INTERNET_OFF
15	Small	S4	WAIT	LONG_BACKGROUND
16	Small	S3	WAIT	GPS_CALIBRATED
17	Small	S4	WAIT	GPS_CALIBRATED
18	Small	S4	NOT_WAIT	GPS_OFF
19	Small	S11	WAIT	TAKE_A_PICTURE
20	Small	S1	NOT_WAIT	RECEIVE_CALL
21	Small	S1	WAIT	GPS_OFF
22	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED
23	Small	S1	WAIT	TAKE_A_PICTURE
24	Small	S11	NOT_WAIT	GPS_ON
25	Small	S2	NOT_WAIT	GPS_NOT_CALIBRATED
26	Small	S12	NOT_WAIT	LONG_BACKGROUND
27	Small	S2	NOT_WAIT	TAKE_A_PICTURE
28	Small	S11	NOT_WAIT	LONG_BACKGROUND
29	Small	S7	WAIT	GPS_OFF
30	Small	S7	NOT_WAIT	LONG_BACKGROUND

31	Perfect	S11	NOT_WAIT	INTERNET_ON
32	Perfect	S1	WAIT	GPS_NOT_CALIBRATED
33	Perfect	S4	NOT_WAIT	TAKE_A_PICTURE
34	Perfect	S10	WAIT	INTERNET_OFF
35	Perfect	S8	WAIT	LONG_BACKGROUND
36	Perfect	S6	WAIT	GPS_CALIBRATED
37	Perfect	S12	NOT_WAIT	GPS_ON
38	Perfect	S5	NOT_WAIT	ORIENTATION_LANDSCAPE
39	Perfect	S10	NOT_WAIT	TAKE_A_PICTURE
40	Perfect	S9	WAIT	GPS_CALIBRATED
41	Perfect	S9	NOT_WAIT	GPS_OFF
42	Perfect	S7	WAIT	TAKE_A_PICTURE
43	Perfect	S2	NOT_WAIT	GPS_OFF
44	Perfect	S12	WAIT	ORIENTATION_PORTRAIT
45	Perfect	S3	WAIT	ORIENTATION_LANDSCAPE
46	Perfect	S9	WAIT	RECEIVE_CALL
47	Limit	S9	WAIT	ORIENTATION_PORTRAIT
48	Limit	S3	NOT_WAIT	GPS_OFF
49	Limit	S10	WAIT	LONG_BACKGROUND
50	Limit	S8	NOT_WAIT	RECEIVE_CALL
51	Limit	S2	WAIT	INTERNET_ON
52	Limit	S6	NOT_WAIT	TAKE_A_PICTURE
53	Limit	S12	WAIT	TAKE_A_PICTURE
54	Limit	S4	WAIT	INTERNET_ON
55	Limit	S5	NOT_WAIT	ORIENTATION_LANDSCAPE
56	Limit	S7	WAIT	INTERNET_OFF
57	Limit	S9	WAIT	GPS_CALIBRATED
58	Limit	S1	WAIT	GPS_NOT_CALIBRATED
59	Limit	S11	WAIT	GPS_ON
60	Long	S3	WAIT	TAKE_A_PICTURE
61	Long	S10	NOT_WAIT	GPS_CALIBRATED

62	Long	S6	NOT_WAIT	RECEIVE_CALL
63	Long	S8	WAIT	INTERNET_ON
64	Long	S1	NOT_WAIT	LONG_BACKGROUND
65	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT
66	Long	S9	NOT_WAIT	LONG_BACKGROUND
67	Long	S10	WAIT	RECEIVE_CALL
68	Long	S9	NOT_WAIT	TAKE_A_PICTURE
69	Long	S2	NOT_WAIT	RECEIVE_CALL
70	Long	S4	WAIT	RECEIVE_CALL
71	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT
72	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE
73	Long	S6	WAIT	GPS_OFF
74	Long	S5	WAIT	GPS_ON
75	Long	S7	WAIT	INTERNET_OFF
76	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED
77	Long	S5	WAIT	LONG_BACKGROUND
78	Long	S12	NOT_WAIT	RECEIVE_CALL
79	Long	S6	WAIT	INTERNET_ON
80	Long	S10	NOT_WAIT	GPS_OFF
81	Long	S6	NOT_WAIT	LONG_BACKGROUND
82	Long	S1	WAIT	INTERNET_OFF
83	Long	S5	WAIT	RECEIVE_CALL
84	Long	S3	NOT_WAIT	RECEIVE_CALL
85	Long	S1	NOT_WAIT	ORIENTATION_LANDSCAPE
86	Long	S3	WAIT	LONG_BACKGROUND
87	Long	S7	WAIT	RECEIVE_CALL
88	Long	S8	WAIT	GPS_OFF

Table B.2: Order 2 test cases

	Path	Setup	First Delay	First event	Second Delay	Second event
1	Small	S1	WAIT	INTERNET_OFF	NOT_WAIT	TAKE_A_PICTURE
2	Small	S10	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_PORTRAIT
3	Small	S12	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	RECEIVE_CALL
4	Small	S9	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	ACCEPT_CALL
5	Small	S7	WAIT	GPS_OFF	NOT_WAIT	GPS_ON
6	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	LONG_BACKGROUND
7	Small	S2	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_NOT_CALIBRATED
8	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF
9	Small	S3	NOT_WAIT	GPS_CALIBRATED	WAIT	GPS_OFF
10	Small	S4	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL
11	Small	S5	NOT_WAIT	GPS_ON	NOT_WAIT	RECEIVE_CALL
12	Small	S9	WAIT	LONG_BACKGROUND	WAIT	GPS_CALIBRATED
13	Small	S6	NOT_WAIT	INTERNET_ON	NOT_WAIT	INTERNET_OFF
14	Small	S6	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	TAKE_A_PICTURE
15	Small	S10	WAIT	ORIENTATION_PORTRAIT	WAIT	TAKE_A_PICTURE
16	Small	S6	WAIT	TAKE_A_PICTURE	WAIT	ORIENTATION_LANDSCAPE
17	Small	S5	WAIT	INTERNET_OFF	WAIT	INTERNET_ON
18	Small	S9	WAIT	GPS_OFF	WAIT	GPS_ON
19	Small	S4	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_CALIBRATED
20	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	INTERNET_ON
21	Small	S9	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_ON
22	Small	S5	WAIT	GPS_ON	WAIT	GPS_CALIBRATED
23	Small	S4	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT
24	Small	S2	WAIT	INTERNET_ON	WAIT	GPS_OFF
25	Small	S10	WAIT	TAKE_A_PICTURE	WAIT	INTERNET_OFF
26	Small	S5	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT
27	Small	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_OFF
28	Small	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	TAKE_A_PICTURE
29	Small	S11	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_ON
30	Small	S1	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE
31	Small	S6	WAIT	RECEIVE_CALL	NOT_WAIT	GPS_CALIBRATED
32	Small	S7	WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_OFF
33	Small	S10	WAIT	INTERNET_OFF	WAIT	LONG_BACKGROUND
34	Small	S11	WAIT	GPS_ON	WAIT	TAKE_A_PICTURE
35	Small	S7	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL
36	Small	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_OFF
37	Small	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_ON
38	Small	S3	WAIT	RECEIVE_CALL	NOT_WAIT	INTERNET_OFF
39	Small	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT
40	Small	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_CALIBRATED
41	Small	S4	WAIT	INTERNET_ON	NOT_WAIT	LONG_BACKGROUND
42	Small	S7	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_OFF
43	Small	S11	NOT_WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL

44	Small	S11	WAIT	GPS_ON	NOT_WAIT	INTERNET_ON
45	Small	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	INTERNET_OFF
46	Small	S7	NOT_WAIT	TAKE_A_PICTURE	WAIT	LONG_BACKGROUND
47	Small	S3	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_OFF
48	Small	S2	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
49	Small	S4	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_CALIBRATED
50	Small	S1	NOT_WAIT	INTERNET_OFF	WAIT	GPS_NOT_CALIBRATED
51	Small	S11	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL
52	Small	S7	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_NOT_CALIBRATED
53	Small	S2	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED
54	Small	S8	WAIT	GPS_OFF	WAIT	GPS_ON
55	Small	S6	WAIT	LONG_BACKGROUND	WAIT	ORIENTATION_LANDSCAPE
56	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON
57	Small	S2	NOT_WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
58	Small	S2	WAIT	GPS_OFF	WAIT	INTERNET_ON
59	Small	S2	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_NOT_CALIBRATED
60	Small	S12	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE
61	Small	S12	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
62	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	GPS_ON
63	Small	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_ON
64	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
65	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	RECEIVE_CALL
66	Perfect	S7	NOT_WAIT	INTERNET_OFF	WAIT	INTERNET_ON
67	Perfect	S3	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	GPS_CALIBRATED
68	Perfect	S9	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_PORTRAIT
69	Perfect	S9	NOT_WAIT	GPS_OFF	WAIT	LONG_BACKGROUND
70	Perfect	S6	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON
71	Perfect	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	RECEIVE_CALL
72	Perfect	S8	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	ACCEPT_CALL
73	Perfect	S1	NOT_WAIT	LONG_BACKGROUND	WAIT	GPS_OFF
74	Perfect	S2	NOT_WAIT	RECEIVE_CALL	WAIT	LONG_BACKGROUND
75	Perfect	S5	WAIT	GPS_ON	WAIT	ORIENTATION_LANDSCAPE
76	Perfect	S11	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE
77	Perfect	S5	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE
78	Perfect	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_OFF
79	Perfect	S7	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
80	Perfect	S9	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF
81	Perfect	S9	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED
82	Perfect	S12	WAIT	GPS_ON	WAIT	GPS_OFF
83	Perfect	S4	WAIT	GPS_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE
84	Perfect	S8	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	INTERNET_ON
85	Perfect	S12	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	ORIENTATION_PORTRAIT
86	Perfect	S5	WAIT	ORIENTATION_LANDSCAPE	WAIT	LONG_BACKGROUND
87	Perfect	S8	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE

88	Perfect	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT
89	Limit	S3	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE
90	Limit	S9	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
91	Limit	S12	WAIT	GPS_ON	NOT_WAIT	INTERNET_OFF
92	Limit	S9	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE
93	Limit	S4	NOT_WAIT	GPS_OFF	WAIT	GPS_ON
94	Limit	S1	WAIT	GPS_NOT_CALIBRATED	WAIT	LONG_BACKGROUND
95	Limit	S3	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT
96	Limit	S3	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL
97	Limit	S2	WAIT	INTERNET_ON	WAIT	RECEIVE_CALL
98	Limit	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_NOT_CALIBRATED
99	Limit	S7	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE
100	Limit	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON
101	Limit	S7	NOT_WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
102	Limit	S5	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON
103	Limit	S10	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED
104	Limit	S2	WAIT	INTERNET_ON	WAIT	INTERNET_OFF
105	Limit	S1	WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_NOT_CALIBRATED
106	Limit	S5	NOT_WAIT	INTERNET_OFF	WAIT	GPS_ON
107	Limit	S8	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_NOT_CALIBRATED
108	Limit	S6	WAIT	GPS_OFF	NOT_WAIT	RECEIVE_CALL
109	Limit	S6	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_OFF
110	Limit	S12	WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
111	Limit	S12	NOT_WAIT	GPS_ON	WAIT	GPS_CALIBRATED
112	Limit	S1	WAIT	INTERNET_OFF	WAIT	ORIENTATION_LANDSCAPE
113	Limit	S11	WAIT	GPS_ON	NOT_WAIT	GPS_OFF
114	Limit	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT
115	Limit	S11	NOT_WAIT	GPS_ON	WAIT	ORIENTATION_PORTRAIT
116	Limit	S5	NOT_WAIT	GPS_ON	WAIT	INTERNET_OFF
117	Limit	S8	NOT_WAIT	LONG_BACKGROUND	WAIT	INTERNET_ON
118	Limit	S10	WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE
119	Long	S2	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_LANDSCAPE
120	Long	S9	WAIT	TAKE_A_PICTURE	NOT_WAIT	GPS_OFF
121	Long	S2	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT
122	Long	S3	WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
123	Long	S6	WAIT	GPS_CALIBRATED	WAIT	LONG_BACKGROUND
124	Long	S8	NOT_WAIT	INTERNET_ON	WAIT	TAKE_A_PICTURE
125	Long	S8	WAIT	LONG_BACKGROUND	NOT_WAIT	RECEIVE_CALL
126	Long	S3	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
127	Long	S3	WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON
128	Long	S1	NOT_WAIT	GPS_OFF	WAIT	GPS_ON
129	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	LONG_BACKGROUND
130	Long	S10	NOT_WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
131	Long	S4	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF

132	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_OFF
133	Long	S10	WAIT	INTERNET_OFF	WAIT	GPS_CALIBRATED
134	Long	S8	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED
135	Long	S1	WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
136	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
137	Long	S1	WAIT	INTERNET_OFF	WAIT	INTERNET_ON
138	Long	S11	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
139	Long	S5	WAIT	GPS_ON	NOT_WAIT	GPS_OFF
140	Long	S7	WAIT	RECEIVE_CALL	WAIT	GPS_NOT_CALIBRATED
141	Long	S1	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	GPS_CALIBRATED
142	Long	S4	WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
143	Long	S10	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
144	Long	S11	WAIT	GPS_ON	NOT_WAIT	GPS_CALIBRATED
145	Long	S8	WAIT	INTERNET_ON	WAIT	INTERNET_OFF
146	Long	S4	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_ON
147	Long	S10	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON
148	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED
149	Long	S8	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL
150	Long	S9	WAIT	RECEIVE_CALL	NOT_WAIT	TAKE_A_PICTURE
151	Long	S6	WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
152	Long	S10	WAIT	INTERNET_OFF	NOT_WAIT	GPS_OFF
153	Long	S1	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL
154	Long	S11	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	ORIENTATION_PORTRAIT
155	Long	S1	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL
156	Long	S3	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED
157	Long	S12	NOT_WAIT	INTERNET_OFF	WAIT	TAKE_A_PICTURE
158	Long	S8	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_PORTRAIT
159	Long	S2	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE
160	Long	S4	WAIT	GPS_OFF	WAIT	TAKE_A_PICTURE
161	Long	S6	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED
162	Long	S7	NOT_WAIT	INTERNET_OFF	NOT_WAIT	ORIENTATION_PORTRAIT
163	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_ON
164	Long	S3	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	LONG_BACKGROUND
165	Long	S2	WAIT	GPS_OFF	NOT_WAIT	GPS_ON
166	Long	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON
167	Long	S9	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL
168	Long	S11	NOT_WAIT	GPS_ON	WAIT	LONG_BACKGROUND
169	Long	S10	WAIT	GPS_CALIBRATED	WAIT	TAKE_A_PICTURE
170	Long	S9	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	GPS_CALIBRATED
171	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	ACCEPT_CALL
172	Long	S4	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED
173	Long	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	RECEIVE_CALL
174	Long	S3	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_LANDSCAPE
175	Long	S11	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_ON

176	Long	S7	WAIT	ORIENTATION_PORTRAIT	WAIT	INTERNET_OFF
177	Long	S10	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE
178	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	RECEIVE_CALL
179	Long	S6	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_LANDSCAPE
180	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_ON
181	Long	S6	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT
182	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	INTERNET_ON
183	Long	S12	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON

Appendix C

Raw Results

Table C.1: Order 1 test result for OsmAnd application

OsmAnd						
	Path	Setup	First Delay	First event	Verdict	Crash
1	Small	S8	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
2	Small	S11	WAIT	RECEIVE_CALL	Pass	
3	Small	S2	WAIT	LONG_BACKGROUND	Pass	
4	Small	S9	NOT_WAIT	INTERNET_ON	Pass	
5	Small	S5	NOT_WAIT	INTERNET_OFF	Pass	
6	Small	S5	WAIT	TAKE_A_PICTURE	Pass	
7	Small	S10	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
8	Small	S6	WAIT	ORIENTATION_LANDSCAPE	Pass	
9	Small	S9	NOT_WAIT	RECEIVE_CALL	Pass	
10	Small	S12	WAIT	INTERNET_OFF	Pass	
11	Small	S7	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
12	Small	S4	WAIT	ORIENTATION_LANDSCAPE	Pass	
13	Small	S8	WAIT	TAKE_A_PICTURE	Pass	
14	Small	S3	WAIT	INTERNET_OFF	Pass	
15	Small	S4	WAIT	LONG_BACKGROUND	Pass	
16	Small	S3	WAIT	GPS_CALIBRATED	Fail	
17	Small	S4	WAIT	GPS_CALIBRATED	Pass	
18	Small	S4	NOT_WAIT	GPS_OFF	Pass	
19	Small	S11	WAIT	TAKE_A_PICTURE	Pass	
20	Small	S1	NOT_WAIT	RECEIVE_CALL	Pass	
21	Small	S1	WAIT	GPS_OFF	Pass	
22	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
23	Small	S1	WAIT	TAKE_A_PICTURE	Pass	
24	Small	S11	NOT_WAIT	GPS_ON	Pass	
25	Small	S2	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
26	Small	S12	NOT_WAIT	LONG_BACKGROUND	Pass	
27	Small	S2	NOT_WAIT	TAKE_A_PICTURE	Pass	
28	Small	S11	NOT_WAIT	LONG_BACKGROUND	Pass	
29	Small	S7	WAIT	GPS_OFF	Pass	
30	Small	S7	NOT_WAIT	LONG_BACKGROUND	Pass	
31	Perfect	S11	NOT_WAIT	INTERNET_ON	Pass	
32	Perfect	S1	WAIT	GPS_NOT_CALIBRATED	Pass	
33	Perfect	S4	NOT_WAIT	TAKE_A_PICTURE	Pass	
34	Perfect	S10	WAIT	INTERNET_OFF	Pass	
35	Perfect	S8	WAIT	LONG_BACKGROUND	Pass	
36	Perfect	S6	WAIT	GPS_CALIBRATED	Pass	
37	Perfect	S12	NOT_WAIT	GPS_ON	Pass	
38	Perfect	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
39	Perfect	S10	NOT_WAIT	TAKE_A_PICTURE	Pass	
40	Perfect	S9	WAIT	GPS_CALIBRATED	Fail	
41	Perfect	S9	NOT_WAIT	GPS_OFF	Pass	
42	Perfect	S7	WAIT	TAKE_A_PICTURE	Pass	
43	Perfect	S2	NOT_WAIT	GPS_OFF	Pass	
44	Perfect	S12	WAIT	ORIENTATION_PORTRAIT	Pass	
45	Perfect	S3	WAIT	ORIENTATION_LANDSCAPE	Pass	
46	Perfect	S9	WAIT	RECEIVE_CALL	Pass	
47	Limit	S9	WAIT	ORIENTATION_PORTRAIT	Pass	
48	Limit	S3	NOT_WAIT	GPS_OFF	Pass	

49	Limit	S10	WAIT	LONG_BACKGROUND	Pass	
50	Limit	S8	NOT_WAIT	RECEIVE_CALL	Pass	
51	Limit	S2	WAIT	INTERNET_ON	Pass	
52	Limit	S6	NOT_WAIT	TAKE_A_PICTURE	Pass	
53	Limit	S12	WAIT	TAKE_A_PICTURE	Pass	
54	Limit	S4	WAIT	INTERNET_ON	Pass	
55	Limit	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
56	Limit	S7	WAIT	INTERNET_OFF	Pass	
57	Limit	S9	WAIT	GPS_CALIBRATED	Fail	
58	Limit	S1	WAIT	GPS_NOT_CALIBRATED	Pass	
59	Limit	S11	WAIT	GPS_ON	Pass	
60	Long	S3	WAIT	TAKE_A_PICTURE	Pass	
61	Long	S10	NOT_WAIT	GPS_CALIBRATED	Fail	
62	Long	S6	NOT_WAIT	RECEIVE_CALL	Pass	
63	Long	S8	WAIT	INTERNET_ON	Pass	
64	Long	S1	NOT_WAIT	LONG_BACKGROUND	Pass	
65	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
66	Long	S9	NOT_WAIT	LONG_BACKGROUND	Pass	
67	Long	S10	WAIT	RECEIVE_CALL	Pass	
68	Long	S9	NOT_WAIT	TAKE_A_PICTURE	Pass	
69	Long	S2	NOT_WAIT	RECEIVE_CALL	Pass	
70	Long	S4	WAIT	RECEIVE_CALL	Pass	
71	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
72	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
73	Long	S6	WAIT	GPS_OFF	Pass	
74	Long	S5	WAIT	GPS_ON	Pass	
75	Long	S7	WAIT	INTERNET_OFF	Pass	
76	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
77	Long	S5	WAIT	LONG_BACKGROUND	Pass	
78	Long	S12	NOT_WAIT	RECEIVE_CALL	Pass	
79	Long	S6	WAIT	INTERNET_ON	Pass	
80	Long	S10	NOT_WAIT	GPS_OFF	Pass	
81	Long	S6	NOT_WAIT	LONG_BACKGROUND	Pass	
82	Long	S1	WAIT	INTERNET_OFF	Pass	
83	Long	S5	WAIT	RECEIVE_CALL	Pass	
84	Long	S3	NOT_WAIT	RECEIVE_CALL	Pass	
85	Long	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
86	Long	S3	WAIT	LONG_BACKGROUND	Pass	
87	Long	S7	WAIT	RECEIVE_CALL	Pass	
88	Long	S8	WAIT	GPS_OFF	Pass	

Table C.2: Order 1 test result for GPS Offline Navigation

GPS Offline Navigation Route Maps & Directions						
	Path	Setup	First Delay	First event	Verdict	Crash
1	Small	S8	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
2	Small	S11	WAIT	RECEIVE_CALL	Pass	
3	Small	S2	WAIT	LONG_BACKGROUND	Fail	
4	Small	S9	NOT_WAIT	INTERNET_ON	Pass	
5	Small	S5	NOT_WAIT	INTERNET_OFF	Pass	
6	Small	S5	WAIT	TAKE_A_PICTURE	Pass	
7	Small	S10	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
8	Small	S6	WAIT	ORIENTATION_LANDSCAPE	Pass	
9	Small	S9	NOT_WAIT	RECEIVE_CALL	Pass	
10	Small	S12	WAIT	INTERNET_OFF	Pass	
11	Small	S7	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
12	Small	S4	WAIT	ORIENTATION_LANDSCAPE	Pass	
13	Small	S8	WAIT	TAKE_A_PICTURE	Pass	
14	Small	S3	WAIT	INTERNET_OFF	Pass	
15	Small	S4	WAIT	LONG_BACKGROUND	Fail	
16	Small	S3	WAIT	GPS_CALIBRATED	Pass	
17	Small	S4	WAIT	GPS_CALIBRATED	Pass	
18	Small	S4	NOT_WAIT	GPS_OFF	Pass	
19	Small	S11	WAIT	TAKE_A_PICTURE	Pass	
20	Small	S1	NOT_WAIT	RECEIVE_CALL	Pass	
21	Small	S1	WAIT	GPS_OFF	Pass	
22	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
23	Small	S1	WAIT	TAKE_A_PICTURE	Pass	
24	Small	S11	NOT_WAIT	GPS_ON	Pass	
25	Small	S2	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
26	Small	S12	NOT_WAIT	LONG_BACKGROUND	Fail	
27	Small	S2	NOT_WAIT	TAKE_A_PICTURE	Pass	
28	Small	S11	NOT_WAIT	LONG_BACKGROUND	Fail	
29	Small	S7	WAIT	GPS_OFF	Pass	
30	Small	S7	NOT_WAIT	LONG_BACKGROUND	Fail	
31	Perfect	S11	NOT_WAIT	INTERNET_ON	Pass	
32	Perfect	S1	WAIT	GPS_NOT_CALIBRATED	Pass	
33	Perfect	S4	NOT_WAIT	TAKE_A_PICTURE	Pass	
34	Perfect	S10	WAIT	INTERNET_OFF	Pass	
35	Perfect	S8	WAIT	LONG_BACKGROUND	Pass	
36	Perfect	S6	WAIT	GPS_CALIBRATED	Pass	
37	Perfect	S12	NOT_WAIT	GPS_ON	Pass	
38	Perfect	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
39	Perfect	S10	NOT_WAIT	TAKE_A_PICTURE	Pass	
40	Perfect	S9	WAIT	GPS_CALIBRATED	Pass	
41	Perfect	S9	NOT_WAIT	GPS_OFF	Pass	
42	Perfect	S7	WAIT	TAKE_A_PICTURE	Pass	
43	Perfect	S2	NOT_WAIT	GPS_OFF	Pass	
44	Perfect	S12	WAIT	ORIENTATION_PORTRAIT	Pass	
45	Perfect	S3	WAIT	ORIENTATION_LANDSCAPE	Pass	
46	Perfect	S9	WAIT	RECEIVE_CALL	Pass	
47	Limit	S9	WAIT	ORIENTATION_PORTRAIT	Pass	
48	Limit	S3	NOT_WAIT	GPS_OFF	Pass	

49	Limit	S10	WAIT	LONG_BACKGROUND	Pass	
50	Limit	S8	NOT_WAIT	RECEIVE_CALL	Pass	
51	Limit	S2	WAIT	INTERNET_ON	Pass	
52	Limit	S6	NOT_WAIT	TAKE_A_PICTURE	Pass	
53	Limit	S12	WAIT	TAKE_A_PICTURE	Pass	
54	Limit	S4	WAIT	INTERNET_ON	Pass	
55	Limit	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
56	Limit	S7	WAIT	INTERNET_OFF	Pass	
57	Limit	S9	WAIT	GPS_CALIBRATED	Pass	
58	Limit	S1	WAIT	GPS_NOT_CALIBRATED	Pass	
59	Limit	S11	WAIT	GPS_ON	Pass	
60	Long	S3	WAIT	TAKE_A_PICTURE	Pass	
61	Long	S10	NOT_WAIT	GPS_CALIBRATED	Pass	
62	Long	S6	NOT_WAIT	RECEIVE_CALL	Pass	
63	Long	S8	WAIT	INTERNET_ON	Pass	
64	Long	S1	NOT_WAIT	LONG_BACKGROUND	Fail	
65	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
66	Long	S9	NOT_WAIT	LONG_BACKGROUND	Fail	
67	Long	S10	WAIT	RECEIVE_CALL	Pass	
68	Long	S9	NOT_WAIT	TAKE_A_PICTURE	Pass	
69	Long	S2	NOT_WAIT	RECEIVE_CALL	Pass	
70	Long	S4	WAIT	RECEIVE_CALL	Pass	
71	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
72	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
73	Long	S6	WAIT	GPS_OFF	Pass	
74	Long	S5	WAIT	GPS_ON	Pass	
75	Long	S7	WAIT	INTERNET_OFF	Pass	
76	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
77	Long	S5	WAIT	LONG_BACKGROUND	Fail	
78	Long	S12	NOT_WAIT	RECEIVE_CALL	Pass	
79	Long	S6	WAIT	INTERNET_ON	Pass	
80	Long	S10	NOT_WAIT	GPS_OFF	Pass	
81	Long	S6	NOT_WAIT	LONG_BACKGROUND	Fail	
82	Long	S1	WAIT	INTERNET_OFF	Pass	
83	Long	S5	WAIT	RECEIVE_CALL	Pass	
84	Long	S3	NOT_WAIT	RECEIVE_CALL	Pass	
85	Long	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
86	Long	S3	WAIT	LONG_BACKGROUND	Pass	
87	Long	S7	WAIT	RECEIVE_CALL	Pass	
88	Long	S8	WAIT	GPS_OFF	Pass	

Table C.3: Order 1 test result for Genius Maps application

Genius Maps: Offline GPS Navigator						
	Path	Setup	First Delay	First event	Verdict	Crash
1	Small	S8	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
2	Small	S11	WAIT	RECEIVE_CALL	Pass	
3	Small	S2	WAIT	LONG_BACKGROUND	Fail	
4	Small	S9	NOT_WAIT	INTERNET_ON	Pass	
5	Small	S5	NOT_WAIT	INTERNET_OFF	Pass	
6	Small	S5	WAIT	TAKE_A_PICTURE	Pass	
7	Small	S10	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
8	Small	S6	WAIT	ORIENTATION_LANDSCAPE	Pass	
9	Small	S9	NOT_WAIT	RECEIVE_CALL	Pass	
10	Small	S12	WAIT	INTERNET_OFF	Pass	
11	Small	S7	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
12	Small	S4	WAIT	ORIENTATION_LANDSCAPE	Pass	
13	Small	S8	WAIT	TAKE_A_PICTURE	Fail	
14	Small	S3	WAIT	INTERNET_OFF	Pass	
15	Small	S4	WAIT	LONG_BACKGROUND	Pass	
16	Small	S3	WAIT	GPS_CALIBRATED	Pass	
17	Small	S4	WAIT	GPS_CALIBRATED	Pass	
18	Small	S4	NOT_WAIT	GPS_OFF	Pass	
19	Small	S11	WAIT	TAKE_A_PICTURE	Pass	
20	Small	S1	NOT_WAIT	RECEIVE_CALL	Pass	
21	Small	S1	WAIT	GPS_OFF	Pass	
22	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
23	Small	S1	WAIT	TAKE_A_PICTURE	Pass	
24	Small	S11	NOT_WAIT	GPS_ON	Pass	
25	Small	S2	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
26	Small	S12	NOT_WAIT	LONG_BACKGROUND	Pass	
27	Small	S2	NOT_WAIT	TAKE_A_PICTURE	Pass	
28	Small	S11	NOT_WAIT	LONG_BACKGROUND	Pass	
29	Small	S7	WAIT	GPS_OFF	Pass	
30	Small	S7	NOT_WAIT	LONG_BACKGROUND	Fail	
31	Perfect	S11	NOT_WAIT	INTERNET_ON	Pass	
32	Perfect	S1	WAIT	GPS_NOT_CALIBRATED	Pass	
33	Perfect	S4	NOT_WAIT	TAKE_A_PICTURE	Pass	
34	Perfect	S10	WAIT	INTERNET_OFF	Pass	
35	Perfect	S8	WAIT	LONG_BACKGROUND	Pass	
36	Perfect	S6	WAIT	GPS_CALIBRATED	Pass	
37	Perfect	S12	NOT_WAIT	GPS_ON	Pass	
38	Perfect	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
39	Perfect	S10	NOT_WAIT	TAKE_A_PICTURE	Pass	
40	Perfect	S9	WAIT	GPS_CALIBRATED	Pass	
41	Perfect	S9	NOT_WAIT	GPS_OFF	Pass	
42	Perfect	S7	WAIT	TAKE_A_PICTURE	Pass	
43	Perfect	S2	NOT_WAIT	GPS_OFF	Pass	
44	Perfect	S12	WAIT	ORIENTATION_PORTRAIT	Pass	
45	Perfect	S3	WAIT	ORIENTATION_LANDSCAPE	Pass	
46	Perfect	S9	WAIT	RECEIVE_CALL	Pass	
47	Limit	S9	WAIT	ORIENTATION_PORTRAIT	Pass	
48	Limit	S3	NOT_WAIT	GPS_OFF	Pass	

49	Limit	S10	WAIT	LONG_BACKGROUND	Pass	
50	Limit	S8	NOT_WAIT	RECEIVE_CALL	Pass	
51	Limit	S2	WAIT	INTERNET_ON	Pass	
52	Limit	S6	NOT_WAIT	TAKE_A_PICTURE	Pass	
53	Limit	S12	WAIT	TAKE_A_PICTURE	Pass	
54	Limit	S4	WAIT	INTERNET_ON	Pass	
55	Limit	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
56	Limit	S7	WAIT	INTERNET_OFF	Pass	
57	Limit	S9	WAIT	GPS_CALIBRATED	Pass	
58	Limit	S1	WAIT	GPS_NOT_CALIBRATED	Pass	
59	Limit	S11	WAIT	GPS_ON	Pass	
60	Long	S3	WAIT	TAKE_A_PICTURE	Pass	
61	Long	S10	NOT_WAIT	GPS_CALIBRATED	Pass	
62	Long	S6	NOT_WAIT	RECEIVE_CALL	Pass	
63	Long	S8	WAIT	INTERNET_ON	Pass	
64	Long	S1	NOT_WAIT	LONG_BACKGROUND	Pass	
65	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
66	Long	S9	NOT_WAIT	LONG_BACKGROUND	Pass	
67	Long	S10	WAIT	RECEIVE_CALL	Pass	
68	Long	S9	NOT_WAIT	TAKE_A_PICTURE	Pass	
69	Long	S2	NOT_WAIT	RECEIVE_CALL	Pass	
70	Long	S4	WAIT	RECEIVE_CALL	Pass	
71	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
72	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
73	Long	S6	WAIT	GPS_OFF	Pass	
74	Long	S5	WAIT	GPS_ON	Pass	
75	Long	S7	WAIT	INTERNET_OFF	Pass	
76	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
77	Long	S5	WAIT	LONG_BACKGROUND	Pass	
78	Long	S12	NOT_WAIT	RECEIVE_CALL	Pass	
79	Long	S6	WAIT	INTERNET_ON	Pass	
80	Long	S10	NOT_WAIT	GPS_OFF	Pass	
81	Long	S6	NOT_WAIT	LONG_BACKGROUND	Pass	
82	Long	S1	WAIT	INTERNET_OFF	Pass	
83	Long	S5	WAIT	RECEIVE_CALL	Pass	
84	Long	S3	NOT_WAIT	RECEIVE_CALL	Pass	
85	Long	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
86	Long	S3	WAIT	LONG_BACKGROUND	Pass	
87	Long	S7	WAIT	RECEIVE_CALL	Pass	
88	Long	S8	WAIT	GPS_OFF	Pass	

Table C.4: Order 1 test result for Voice GPS Navigation application

Voice GPS Navigation: Live Driving Direction						
	Path	Setup	First Delay	First event	Verdict	Crash
1	Small	S8	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
2	Small	S11	WAIT	RECEIVE_CALL	Pass	
3	Small	S2	WAIT	LONG_BACKGROUND	Pass	
4	Small	S9	NOT_WAIT	INTERNET_ON	Fail	X
5	Small	S5	NOT_WAIT	INTERNET_OFF	Pass	
6	Small	S5	WAIT	TAKE_A_PICTURE	Pass	
7	Small	S10	NOT_WAIT	ORIENTATION_PORTRAIT	Fail	X
8	Small	S6	WAIT	ORIENTATION_LANDSCAPE	Pass	
9	Small	S9	NOT_WAIT	RECEIVE_CALL	Pass	
10	Small	S12	WAIT	INTERNET_OFF	Pass	
11	Small	S7	NOT_WAIT	ORIENTATION_PORTRAIT	Fail	
12	Small	S4	WAIT	ORIENTATION_LANDSCAPE	Pass	
13	Small	S8	WAIT	TAKE_A_PICTURE	Fail	
14	Small	S3	WAIT	INTERNET_OFF	Pass	
15	Small	S4	WAIT	LONG_BACKGROUND	Pass	
16	Small	S3	WAIT	GPS_CALIBRATED	Fail	
17	Small	S4	WAIT	GPS_CALIBRATED	Pass	
18	Small	S4	NOT_WAIT	GPS_OFF	Pass	
19	Small	S11	WAIT	TAKE_A_PICTURE	Pass	
20	Small	S1	NOT_WAIT	RECEIVE_CALL	Pass	
21	Small	S1	WAIT	GPS_OFF	Fail	X
22	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
23	Small	S1	WAIT	TAKE_A_PICTURE	Pass	
24	Small	S11	NOT_WAIT	GPS_ON	Pass	
25	Small	S2	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
26	Small	S12	NOT_WAIT	LONG_BACKGROUND	Pass	
27	Small	S2	NOT_WAIT	TAKE_A_PICTURE	Pass	
28	Small	S11	NOT_WAIT	LONG_BACKGROUND	Pass	
29	Small	S7	WAIT	GPS_OFF	Pass	
30	Small	S7	NOT_WAIT	LONG_BACKGROUND	Pass	
31	Perfect	S11	NOT_WAIT	INTERNET_ON	Pass	
32	Perfect	S1	WAIT	GPS_NOT_CALIBRATED	Pass	
33	Perfect	S4	NOT_WAIT	TAKE_A_PICTURE	Pass	
34	Perfect	S10	WAIT	INTERNET_OFF	Pass	
35	Perfect	S8	WAIT	LONG_BACKGROUND	Pass	
36	Perfect	S6	WAIT	GPS_CALIBRATED	Pass	
37	Perfect	S12	NOT_WAIT	GPS_ON	Pass	
38	Perfect	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
39	Perfect	S10	NOT_WAIT	TAKE_A_PICTURE	Pass	
40	Perfect	S9	WAIT	GPS_CALIBRATED	Pass	
41	Perfect	S9	NOT_WAIT	GPS_OFF	Pass	
42	Perfect	S7	WAIT	TAKE_A_PICTURE	Pass	
43	Perfect	S2	NOT_WAIT	GPS_OFF	Pass	
44	Perfect	S12	WAIT	ORIENTATION_PORTRAIT	Pass	
45	Perfect	S3	WAIT	ORIENTATION_LANDSCAPE	Pass	
46	Perfect	S9	WAIT	RECEIVE_CALL	Pass	
47	Limit	S9	WAIT	ORIENTATION_PORTRAIT	Pass	
48	Limit	S3	NOT_WAIT	GPS_OFF	Pass	

49	Limit	S10	WAIT	LONG_BACKGROUND	Pass	
50	Limit	S8	NOT_WAIT	RECEIVE_CALL	Pass	
51	Limit	S2	WAIT	INTERNET_ON	Fail	X
52	Limit	S6	NOT_WAIT	TAKE_A_PICTURE	Pass	
53	Limit	S12	WAIT	TAKE_A_PICTURE	Pass	
54	Limit	S4	WAIT	INTERNET_ON	Pass	
55	Limit	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
56	Limit	S7	WAIT	INTERNET_OFF	Pass	
57	Limit	S9	WAIT	GPS_CALIBRATED	Pass	
58	Limit	S1	WAIT	GPS_NOT_CALIBRATED	Pass	
59	Limit	S11	WAIT	GPS_ON	Pass	
60	Long	S3	WAIT	TAKE_A_PICTURE	Pass	
61	Long	S10	NOT_WAIT	GPS_CALIBRATED	Fail	X
62	Long	S6	NOT_WAIT	RECEIVE_CALL	Pass	
63	Long	S8	WAIT	INTERNET_ON	Fail	
64	Long	S1	NOT_WAIT	LONG_BACKGROUND	Pass	
65	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
66	Long	S9	NOT_WAIT	LONG_BACKGROUND	Pass	
67	Long	S10	WAIT	RECEIVE_CALL	Pass	
68	Long	S9	NOT_WAIT	TAKE_A_PICTURE	Pass	
69	Long	S2	NOT_WAIT	RECEIVE_CALL	Fail	
70	Long	S4	WAIT	RECEIVE_CALL	Pass	
71	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
72	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	Fail	
73	Long	S6	WAIT	GPS_OFF	Pass	
74	Long	S5	WAIT	GPS_ON	Pass	
75	Long	S7	WAIT	INTERNET_OFF	Fail	
76	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	Fail	X
77	Long	S5	WAIT	LONG_BACKGROUND	Pass	
78	Long	S12	NOT_WAIT	RECEIVE_CALL	Pass	
79	Long	S6	WAIT	INTERNET_ON	Pass	
80	Long	S10	NOT_WAIT	GPS_OFF	Pass	
81	Long	S6	NOT_WAIT	LONG_BACKGROUND	Pass	
82	Long	S1	WAIT	INTERNET_OFF	Fail	X
83	Long	S5	WAIT	RECEIVE_CALL	Pass	
84	Long	S3	NOT_WAIT	RECEIVE_CALL	Fail	X
85	Long	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	Fail	
86	Long	S3	WAIT	LONG_BACKGROUND	Pass	
87	Long	S7	WAIT	RECEIVE_CALL	Fail	
88	Long	S8	WAIT	GPS_OFF	Pass	

Table C.5: Order 2 test result for OsmAnd application

OsmAnd								
	Path	Setup	First Delay	First event	Second Delay	Second event	Verdict	Crash
1	Small	S1	WAIT	INTERNET_OFF	NOT_WAIT	TAKE_A_PICTURE	Pass	
2	Small	S10	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_PORTRAIT	Pass	
3	Small	S12	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	RECEIVE_CALL	Pass	
4	Small	S9	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	Pass_CALL	Pass	
5	Small	S7	WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
6	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	LONG_BACKGROUND	Pass	
7	Small	S2	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_NOT_CALIBRATED	Pass	
8	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
9	Small	S3	NOT_WAIT	GPS_CALIBRATED	WAIT	GPS_OFF	Fail	
10	Small	S4	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL	Pass	
11	Small	S5	NOT_WAIT	GPS_ON	NOT_WAIT	RECEIVE_CALL	Pass	
12	Small	S9	WAIT	LONG_BACKGROUND	WAIT	GPS_CALIBRATED	Fail	
13	Small	S6	NOT_WAIT	INTERNET_ON	NOT_WAIT	INTERNET_OFF	Pass	
14	Small	S6	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	TAKE_A_PICTURE	Pass	
15	Small	S10	WAIT	ORIENTATION_PORTRAIT	WAIT	TAKE_A_PICTURE	Pass	
16	Small	S6	WAIT	TAKE_A_PICTURE	WAIT	ORIENTATION_LANDSCAPE	Pass	
17	Small	S5	WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
18	Small	S9	WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
19	Small	S4	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_CALIBRATED	Pass	
20	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	INTERNET_ON	Pass	
21	Small	S9	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_ON	Fail	
22	Small	S5	WAIT	GPS_ON	WAIT	GPS_CALIBRATED	Pass	
23	Small	S4	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
24	Small	S2	WAIT	INTERNET_ON	WAIT	GPS_OFF	Pass	
25	Small	S10	WAIT	TAKE_A_PICTURE	WAIT	INTERNET_OFF	Pass	
26	Small	S5	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
27	Small	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_OFF	Pass	
28	Small	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	TAKE_A_PICTURE	Pass	
29	Small	S11	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_ON	Pass	
30	Small	S1	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE	Pass	
31	Small	S6	WAIT	RECEIVE_CALL	NOT_WAIT	GPS_CALIBRATED	Fail	
32	Small	S7	WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_OFF	Pass	
33	Small	S10	WAIT	INTERNET_OFF	WAIT	LONG_BACKGROUND	Pass	
34	Small	S11	WAIT	GPS_ON	WAIT	TAKE_A_PICTURE	Pass	
35	Small	S7	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
36	Small	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_OFF	Pass	
37	Small	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_ON	Pass	
38	Small	S3	WAIT	RECEIVE_CALL	NOT_WAIT	INTERNET_OFF	Pass	
39	Small	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT	Fail	
40	Small	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_CALIBRATED	Fail	
41	Small	S4	WAIT	INTERNET_ON	NOT_WAIT	LONG_BACKGROUND	Pass	
42	Small	S7	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_OFF	Pass	
43	Small	S11	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
44	Small	S11	WAIT	GPS_ON	NOT_WAIT	INTERNET_ON	Pass	
45	Small	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	INTERNET_OFF	Pass	
46	Small	S7	NOT_WAIT	TAKE_A_PICTURE	WAIT	LONG_BACKGROUND	Pass	
47	Small	S3	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_OFF	Fail	
48	Small	S2	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
49	Small	S4	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_CALIBRATED	Fail	
50	Small	S1	NOT_WAIT	INTERNET_OFF	WAIT	GPS_NOT_CALIBRATED	Pass	
51	Small	S11	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
52	Small	S7	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
53	Small	S2	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Pass	
54	Small	S8	WAIT	GPS_OFF	WAIT	GPS_ON	Pass	

55	Small	S6	WAIT	LONG_BACKGROUND	WAIT	ORIENTATION_LANDSCAPE	Pass	
56	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
57	Small	S2	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
58	Small	S2	WAIT	GPS_OFF	WAIT	INTERNET_ON	Pass	
59	Small	S2	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
60	Small	S12	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE	Pass	
61	Small	S12	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
62	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	GPS_ON	Pass	
63	Small	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_ON	Pass	
64	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
65	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	RECEIVE_CALL	Pass	
66	Perfect	S7	NOT_WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
67	Perfect	S3	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	GPS_CALIBRATED	Fail	
68	Perfect	S9	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
69	Perfect	S9	NOT_WAIT	GPS_OFF	WAIT	LONG_BACKGROUND	Pass	
70	Perfect	S6	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
71	Perfect	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	RECEIVE_CALL	Fail	
72	Perfect	S8	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	Pass_CALL	Pass	
73	Perfect	S1	NOT_WAIT	LONG_BACKGROUND	WAIT	GPS_OFF	Pass	
74	Perfect	S2	NOT_WAIT	RECEIVE_CALL	WAIT	LONG_BACKGROUND	Pass	
75	Perfect	S5	WAIT	GPS_ON	WAIT	ORIENTATION_LANDSCAPE	Pass	
76	Perfect	S11	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
77	Perfect	S5	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE	Pass	
78	Perfect	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_OFF	Pass	
79	Perfect	S7	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
80	Perfect	S9	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
81	Perfect	S9	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Fail	
82	Perfect	S12	WAIT	GPS_ON	WAIT	GPS_OFF	Pass	
83	Perfect	S4	WAIT	GPS_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE	Fail	
84	Perfect	S8	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	INTERNET_ON	Pass	
85	Perfect	S12	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
86	Perfect	S5	WAIT	ORIENTATION_LANDSCAPE	WAIT	LONG_BACKGROUND	Pass	
87	Perfect	S8	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
88	Perfect	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
89	Limit	S3	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE	Pass	
90	Limit	S9	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
91	Limit	S12	WAIT	GPS_ON	NOT_WAIT	INTERNET_OFF	Pass	
92	Limit	S9	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE	Pass	
93	Limit	S4	NOT_WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
94	Limit	S1	WAIT	GPS_NOT_CALIBRATED	WAIT	LONG_BACKGROUND	Pass	
95	Limit	S3	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
96	Limit	S3	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
97	Limit	S2	WAIT	INTERNET_ON	WAIT	RECEIVE_CALL	Pass	
98	Limit	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
99	Limit	S7	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
100	Limit	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
101	Limit	S7	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
102	Limit	S5	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON	Pass	
103	Limit	S10	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
104	Limit	S2	WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
105	Limit	S1	WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_NOT_CALIBRATED	Pass	
106	Limit	S5	NOT_WAIT	INTERNET_OFF	WAIT	GPS_ON	Pass	
107	Limit	S8	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
108	Limit	S6	WAIT	GPS_OFF	NOT_WAIT	RECEIVE_CALL	Pass	
109	Limit	S6	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_OFF	Pass	
110	Limit	S12	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	

111	Limit	S12	NOT_WAIT	GPS_ON	WAIT	GPS_CALIBRATED	Pass	
112	Limit	S1	WAIT	INTERNET_OFF	WAIT	ORIENTATION_LANDSCAPE	Pass	
113	Limit	S11	WAIT	GPS_ON	NOT_WAIT	GPS_OFF	Pass	
114	Limit	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
115	Limit	S11	NOT_WAIT	GPS_ON	WAIT	ORIENTATION_PORTRAIT	Pass	
116	Limit	S5	NOT_WAIT	GPS_ON	WAIT	INTERNET_OFF	Pass	
117	Limit	S8	NOT_WAIT	LONG_BACKGROUND	WAIT	INTERNET_ON	Pass	
118	Limit	S10	WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE	Pass	
119	Long	S2	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
120	Long	S9	WAIT	TAKE_A_PICTURE	NOT_WAIT	GPS_OFF	Pass	
121	Long	S2	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
122	Long	S3	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
123	Long	S6	WAIT	GPS_CALIBRATED	WAIT	LONG_BACKGROUND	Fail	
124	Long	S8	NOT_WAIT	INTERNET_ON	WAIT	TAKE_A_PICTURE	Pass	
125	Long	S8	WAIT	LONG_BACKGROUND	NOT_WAIT	RECEIVE_CALL	Pass	
126	Long	S3	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
127	Long	S3	WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
128	Long	S1	NOT_WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
129	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	LONG_BACKGROUND	Pass	
130	Long	S10	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
131	Long	S4	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
132	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_OFF	Pass	
133	Long	S10	WAIT	INTERNET_OFF	WAIT	GPS_CALIBRATED	Fail	
134	Long	S8	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Fail	
135	Long	S1	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
136	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
137	Long	S1	WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
138	Long	S11	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
139	Long	S5	WAIT	GPS_ON	NOT_WAIT	GPS_OFF	Pass	
140	Long	S7	WAIT	RECEIVE_CALL	WAIT	GPS_NOT_CALIBRATED	Pass	
141	Long	S1	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	GPS_CALIBRATED	Pass	
142	Long	S4	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
143	Long	S10	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
144	Long	S11	WAIT	GPS_ON	NOT_WAIT	GPS_CALIBRATED	Fail	
145	Long	S8	WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
146	Long	S4	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_ON	Pass	
147	Long	S10	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
148	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Fail	
149	Long	S8	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
150	Long	S9	WAIT	RECEIVE_CALL	NOT_WAIT	TAKE_A_PICTURE	Pass	
151	Long	S6	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
152	Long	S10	WAIT	INTERNET_OFF	NOT_WAIT	GPS_OFF	Pass	
153	Long	S1	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
154	Long	S11	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
155	Long	S1	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL	Pass	
156	Long	S3	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
157	Long	S12	NOT_WAIT	INTERNET_OFF	WAIT	TAKE_A_PICTURE	Pass	
158	Long	S8	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
159	Long	S2	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE	Pass	
160	Long	S4	WAIT	GPS_OFF	WAIT	TAKE_A_PICTURE	Pass	
161	Long	S6	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
162	Long	S7	NOT_WAIT	INTERNET_OFF	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
163	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_ON	Pass	
164	Long	S3	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	LONG_BACKGROUND	Pass	
165	Long	S2	WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
166	Long	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	

167	Long	S9	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
168	Long	S11	NOT_WAIT	GPS_ON	WAIT	LONG_BACKGROUND	Pass	
169	Long	S10	WAIT	GPS_CALIBRATED	WAIT	TAKE_A_PICTURE	Fail	
170	Long	S9	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	GPS_CALIBRATED	Fail	
171	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
172	Long	S4	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
173	Long	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	RECEIVE_CALL	Pass	
174	Long	S3	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_LANDSCAPE	Pass	
175	Long	S11	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_ON	Pass	
176	Long	S7	WAIT	ORIENTATION_PORTRAIT	WAIT	INTERNET_OFF	Pass	
177	Long	S10	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
178	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	RECEIVE_CALL	Pass	
179	Long	S6	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
180	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_ON	Pass	
181	Long	S6	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
182	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	INTERNET_ON	Pass	
183	Long	S12	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON	Pass	

Table C.6: Order 2 test result for GPS Offline Navigation application

GPS Offline Navigation Route Maps & Directions								
	Path	Setup	First Delay	First event	Second Delay	Second event	Verdict	Crash
1	Small	S1	WAIT	INTERNET_OFF	NOT_WAIT	TAKE_A_PICTURE	Pass	
2	Small	S10	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_PORTRAIT	Pass	
3	Small	S12	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	RECEIVE_CALL	Pass	
4	Small	S9	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	Pass_CALL	Pass	
5	Small	S7	WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
6	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	LONG_BACKGROUND	Pass	
7	Small	S2	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_NOT_CALIBRATED	Pass	
8	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
9	Small	S3	NOT_WAIT	GPS_CALIBRATED	WAIT	GPS_OFF	Fail	
10	Small	S4	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL	Pass	
11	Small	S5	NOT_WAIT	GPS_ON	NOT_WAIT	RECEIVE_CALL	Pass	
12	Small	S9	WAIT	LONG_BACKGROUND	WAIT	GPS_CALIBRATED	Pass	
13	Small	S6	NOT_WAIT	INTERNET_ON	NOT_WAIT	INTERNET_OFF	Pass	
14	Small	S6	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	TAKE_A_PICTURE	Pass	
15	Small	S10	WAIT	ORIENTATION_PORTRAIT	WAIT	TAKE_A_PICTURE	Pass	
16	Small	S6	WAIT	TAKE_A_PICTURE	WAIT	ORIENTATION_LANDSCAPE	Pass	
17	Small	S5	WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
18	Small	S9	WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
19	Small	S4	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_CALIBRATED	Pass	
20	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	INTERNET_ON	Pass	
21	Small	S9	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_ON	Pass	
22	Small	S5	WAIT	GPS_ON	WAIT	GPS_CALIBRATED	Fail	
23	Small	S4	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
24	Small	S2	WAIT	INTERNET_ON	WAIT	GPS_OFF	Pass	
25	Small	S10	WAIT	TAKE_A_PICTURE	WAIT	INTERNET_OFF	Pass	
26	Small	S5	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
27	Small	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_OFF	Pass	
28	Small	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	TAKE_A_PICTURE	Pass	
29	Small	S11	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_ON	Pass	
30	Small	S1	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE	Pass	
31	Small	S6	WAIT	RECEIVE_CALL	NOT_WAIT	GPS_CALIBRATED	Pass	
32	Small	S7	WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_OFF	Pass	
33	Small	S10	WAIT	INTERNET_OFF	WAIT	LONG_BACKGROUND	Pass	
34	Small	S11	WAIT	GPS_ON	WAIT	TAKE_A_PICTURE	Pass	
35	Small	S7	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
36	Small	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_OFF	Pass	
37	Small	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_ON	Pass	
38	Small	S3	WAIT	RECEIVE_CALL	NOT_WAIT	INTERNET_OFF	Pass	
39	Small	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
40	Small	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_CALIBRATED	Pass	
41	Small	S4	WAIT	INTERNET_ON	NOT_WAIT	LONG_BACKGROUND	Pass	
42	Small	S7	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_OFF	Pass	
43	Small	S11	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
44	Small	S11	WAIT	GPS_ON	NOT_WAIT	INTERNET_ON	Pass	
45	Small	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	INTERNET_OFF	Pass	
46	Small	S7	NOT_WAIT	TAKE_A_PICTURE	WAIT	LONG_BACKGROUND	Pass	
47	Small	S3	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_OFF	Pass	
48	Small	S2	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
49	Small	S4	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_CALIBRATED	Pass	
50	Small	S1	NOT_WAIT	INTERNET_OFF	WAIT	GPS_NOT_CALIBRATED	Pass	
51	Small	S11	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
52	Small	S7	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
53	Small	S2	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Pass	
54	Small	S8	WAIT	GPS_OFF	WAIT	GPS_ON	Pass	

55	Small	S6	WAIT	LONG_BACKGROUND	WAIT	ORIENTATION_LANDSCAPE	Pass	
56	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
57	Small	S2	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
58	Small	S2	WAIT	GPS_OFF	WAIT	INTERNET_ON	Pass	
59	Small	S2	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
60	Small	S12	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE	Pass	
61	Small	S12	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
62	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	GPS_ON	Pass	
63	Small	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_ON	Pass	
64	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
65	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	RECEIVE_CALL	Pass	
66	Perfect	S7	NOT_WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
67	Perfect	S3	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	GPS_CALIBRATED	Pass	
68	Perfect	S9	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
69	Perfect	S9	NOT_WAIT	GPS_OFF	WAIT	LONG_BACKGROUND	Pass	
70	Perfect	S6	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
71	Perfect	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	RECEIVE_CALL	Pass	
72	Perfect	S8	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	Pass_CALL	Pass	
73	Perfect	S1	NOT_WAIT	LONG_BACKGROUND	WAIT	GPS_OFF	Pass	
74	Perfect	S2	NOT_WAIT	RECEIVE_CALL	WAIT	LONG_BACKGROUND	Pass	
75	Perfect	S5	WAIT	GPS_ON	WAIT	ORIENTATION_LANDSCAPE	Pass	
76	Perfect	S11	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
77	Perfect	S5	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE	Pass	
78	Perfect	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_OFF	Pass	
79	Perfect	S7	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
80	Perfect	S9	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
81	Perfect	S9	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
82	Perfect	S12	WAIT	GPS_ON	WAIT	GPS_OFF	Pass	
83	Perfect	S4	WAIT	GPS_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE	Pass	
84	Perfect	S8	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	INTERNET_ON	Pass	
85	Perfect	S12	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	ORIENTATION_PORTRAIT	Fail	
86	Perfect	S5	WAIT	ORIENTATION_LANDSCAPE	WAIT	LONG_BACKGROUND	Pass	
87	Perfect	S8	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
88	Perfect	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
89	Limit	S3	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE	Pass	
90	Limit	S9	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
91	Limit	S12	WAIT	GPS_ON	NOT_WAIT	INTERNET_OFF	Pass	
92	Limit	S9	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE	Pass	
93	Limit	S4	NOT_WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
94	Limit	S1	WAIT	GPS_NOT_CALIBRATED	WAIT	LONG_BACKGROUND	Pass	
95	Limit	S3	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
96	Limit	S3	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
97	Limit	S2	WAIT	INTERNET_ON	WAIT	RECEIVE_CALL	Pass	
98	Limit	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
99	Limit	S7	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
100	Limit	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
101	Limit	S7	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
102	Limit	S5	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON	Pass	
103	Limit	S10	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
104	Limit	S2	WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
105	Limit	S1	WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_NOT_CALIBRATED	Pass	
106	Limit	S5	NOT_WAIT	INTERNET_OFF	WAIT	GPS_ON	Pass	
107	Limit	S8	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
108	Limit	S6	WAIT	GPS_OFF	NOT_WAIT	RECEIVE_CALL	Pass	
109	Limit	S6	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_OFF	Pass	
110	Limit	S12	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	

111	Limit	S12	NOT_WAIT	GPS_ON	WAIT	GPS_CALIBRATED	Pass	
112	Limit	S1	WAIT	INTERNET_OFF	WAIT	ORIENTATION_LANDSCAPE	Pass	
113	Limit	S11	WAIT	GPS_ON	NOT_WAIT	GPS_OFF	Pass	
114	Limit	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
115	Limit	S11	NOT_WAIT	GPS_ON	WAIT	ORIENTATION_PORTRAIT	Pass	
116	Limit	S5	NOT_WAIT	GPS_ON	WAIT	INTERNET_OFF	Pass	
117	Limit	S8	NOT_WAIT	LONG_BACKGROUND	WAIT	INTERNET_ON	Pass	
118	Limit	S10	WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE	Pass	
119	Long	S2	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
120	Long	S9	WAIT	TAKE_A_PICTURE	NOT_WAIT	GPS_OFF	Pass	
121	Long	S2	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
122	Long	S3	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
123	Long	S6	WAIT	GPS_CALIBRATED	WAIT	LONG_BACKGROUND	Fail	
124	Long	S8	NOT_WAIT	INTERNET_ON	WAIT	TAKE_A_PICTURE	Pass	
125	Long	S8	WAIT	LONG_BACKGROUND	NOT_WAIT	RECEIVE_CALL	Pass	
126	Long	S3	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
127	Long	S3	WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
128	Long	S1	NOT_WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
129	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	LONG_BACKGROUND	Pass	
130	Long	S10	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
131	Long	S4	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
132	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_OFF	Pass	
133	Long	S10	WAIT	INTERNET_OFF	WAIT	GPS_CALIBRATED	Pass	
134	Long	S8	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Pass	
135	Long	S1	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
136	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
137	Long	S1	WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
138	Long	S11	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
139	Long	S5	WAIT	GPS_ON	NOT_WAIT	GPS_OFF	Pass	
140	Long	S7	WAIT	RECEIVE_CALL	WAIT	GPS_NOT_CALIBRATED	Pass	
141	Long	S1	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	GPS_CALIBRATED	Pass	
142	Long	S4	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
143	Long	S10	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
144	Long	S11	WAIT	GPS_ON	NOT_WAIT	GPS_CALIBRATED	Fail	
145	Long	S8	WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
146	Long	S4	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_ON	Pass	
147	Long	S10	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
148	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Pass	
149	Long	S8	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
150	Long	S9	WAIT	RECEIVE_CALL	NOT_WAIT	TAKE_A_PICTURE	Pass	
151	Long	S6	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
152	Long	S10	WAIT	INTERNET_OFF	NOT_WAIT	GPS_OFF	Pass	
153	Long	S1	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
154	Long	S11	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
155	Long	S1	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL	Pass	
156	Long	S3	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
157	Long	S12	NOT_WAIT	INTERNET_OFF	WAIT	TAKE_A_PICTURE	Pass	
158	Long	S8	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
159	Long	S2	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE	Pass	
160	Long	S4	WAIT	GPS_OFF	WAIT	TAKE_A_PICTURE	Fail	
161	Long	S6	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
162	Long	S7	NOT_WAIT	INTERNET_OFF	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
163	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_ON	Pass	
164	Long	S3	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	LONG_BACKGROUND	Pass	
165	Long	S2	WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
166	Long	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	

167	Long	S9	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
168	Long	S11	NOT_WAIT	GPS_ON	WAIT	LONG_BACKGROUND	Fail	
169	Long	S10	WAIT	GPS_CALIBRATED	WAIT	TAKE_A_PICTURE	Pass	
170	Long	S9	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	GPS_CALIBRATED	Pass	
171	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
172	Long	S4	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
173	Long	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	RECEIVE_CALL	Pass	
174	Long	S3	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_LANDSCAPE	Pass	
175	Long	S11	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_ON	Pass	
176	Long	S7	WAIT	ORIENTATION_PORTRAIT	WAIT	INTERNET_OFF	Pass	
177	Long	S10	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
178	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	RECEIVE_CALL	Pass	
179	Long	S6	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_LANDSCAPE	Fail	
180	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_ON	Pass	
181	Long	S6	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
182	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	INTERNET_ON	Pass	
183	Long	S12	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON	Pass	

Table C.7: Order 2 test result for Genius Maps application

Genius Maps: Offline GPS Navigator								
	Path	Setup	First Delay	First event	Second Delay	Second event	Verdict	Crash
1	Small	S1	WAIT	INTERNET_OFF	NOT_WAIT	TAKE_A_PICTURE	Pass	
2	Small	S10	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_PORTRAIT	Pass	
3	Small	S12	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	RECEIVE_CALL	Pass	
4	Small	S9	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	Pass_CALL	Pass	
5	Small	S7	WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
6	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	LONG_BACKGROUND	Pass	
7	Small	S2	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_NOT_CALIBRATED	Pass	
8	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
9	Small	S3	NOT_WAIT	GPS_CALIBRATED	WAIT	GPS_OFF	Pass	
10	Small	S4	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL	Pass	
11	Small	S5	NOT_WAIT	GPS_ON	NOT_WAIT	RECEIVE_CALL	Pass	
12	Small	S9	WAIT	LONG_BACKGROUND	WAIT	GPS_CALIBRATED	Pass	
13	Small	S6	NOT_WAIT	INTERNET_ON	NOT_WAIT	INTERNET_OFF	Pass	
14	Small	S6	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	TAKE_A_PICTURE	Pass	
15	Small	S10	WAIT	ORIENTATION_PORTRAIT	WAIT	TAKE_A_PICTURE	Pass	
16	Small	S6	WAIT	TAKE_A_PICTURE	WAIT	ORIENTATION_LANDSCAPE	Pass	
17	Small	S5	WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
18	Small	S9	WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
19	Small	S4	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_CALIBRATED	Pass	
20	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	INTERNET_ON	Pass	
21	Small	S9	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_ON	Pass	
22	Small	S5	WAIT	GPS_ON	WAIT	GPS_CALIBRATED	Pass	
23	Small	S4	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
24	Small	S2	WAIT	INTERNET_ON	WAIT	GPS_OFF	Pass	
25	Small	S10	WAIT	TAKE_A_PICTURE	WAIT	INTERNET_OFF	Pass	
26	Small	S5	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
27	Small	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_OFF	Pass	
28	Small	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	TAKE_A_PICTURE	Pass	
29	Small	S11	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_ON	Pass	
30	Small	S1	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE	Pass	
31	Small	S6	WAIT	RECEIVE_CALL	NOT_WAIT	GPS_CALIBRATED	Pass	
32	Small	S7	WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_OFF	Fail	
33	Small	S10	WAIT	INTERNET_OFF	WAIT	LONG_BACKGROUND	Pass	
34	Small	S11	WAIT	GPS_ON	WAIT	TAKE_A_PICTURE	Pass	
35	Small	S7	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
36	Small	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_OFF	Pass	
37	Small	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_ON	Pass	
38	Small	S3	WAIT	RECEIVE_CALL	NOT_WAIT	INTERNET_OFF	Pass	
39	Small	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
40	Small	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_CALIBRATED	Pass	
41	Small	S4	WAIT	INTERNET_ON	NOT_WAIT	LONG_BACKGROUND	Pass	
42	Small	S7	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_OFF	Pass	
43	Small	S11	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
44	Small	S11	WAIT	GPS_ON	NOT_WAIT	INTERNET_ON	Pass	
45	Small	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	INTERNET_OFF	Pass	
46	Small	S7	NOT_WAIT	TAKE_A_PICTURE	WAIT	LONG_BACKGROUND	Pass	
47	Small	S3	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_OFF	Fail	
48	Small	S2	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
49	Small	S4	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_CALIBRATED	Pass	
50	Small	S1	NOT_WAIT	INTERNET_OFF	WAIT	GPS_NOT_CALIBRATED	Pass	
51	Small	S11	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
52	Small	S7	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
53	Small	S2	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Pass	
54	Small	S8	WAIT	GPS_OFF	WAIT	GPS_ON	Pass	

55	Small	S6	WAIT	LONG_BACKGROUND	WAIT	ORIENTATION_LANDSCAPE	Pass	
56	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
57	Small	S2	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
58	Small	S2	WAIT	GPS_OFF	WAIT	INTERNET_ON	Pass	
59	Small	S2	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
60	Small	S12	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE	Fail	
61	Small	S12	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Fail	
62	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	GPS_ON	Pass	
63	Small	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_ON	Pass	
64	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
65	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	RECEIVE_CALL	Pass	
66	Perfect	S7	NOT_WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
67	Perfect	S3	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	GPS_CALIBRATED	Pass	
68	Perfect	S9	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
69	Perfect	S9	NOT_WAIT	GPS_OFF	WAIT	LONG_BACKGROUND	Pass	
70	Perfect	S6	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
71	Perfect	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	RECEIVE_CALL	Pass	
72	Perfect	S8	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	Pass_CALL	Pass	
73	Perfect	S1	NOT_WAIT	LONG_BACKGROUND	WAIT	GPS_OFF	Pass	
74	Perfect	S2	NOT_WAIT	RECEIVE_CALL	WAIT	LONG_BACKGROUND	Pass	
75	Perfect	S5	WAIT	GPS_ON	WAIT	ORIENTATION_LANDSCAPE	Pass	
76	Perfect	S11	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
77	Perfect	S5	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE	Pass	
78	Perfect	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_OFF	Pass	
79	Perfect	S7	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
80	Perfect	S9	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
81	Perfect	S9	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
82	Perfect	S12	WAIT	GPS_ON	WAIT	GPS_OFF	Pass	
83	Perfect	S4	WAIT	GPS_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE	Pass	
84	Perfect	S8	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	INTERNET_ON	Pass	
85	Perfect	S12	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
86	Perfect	S5	WAIT	ORIENTATION_LANDSCAPE	WAIT	LONG_BACKGROUND	Pass	
87	Perfect	S8	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
88	Perfect	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
89	Limit	S3	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE	Pass	
90	Limit	S9	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
91	Limit	S12	WAIT	GPS_ON	NOT_WAIT	INTERNET_OFF	Pass	
92	Limit	S9	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE	Pass	
93	Limit	S4	NOT_WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
94	Limit	S1	WAIT	GPS_NOT_CALIBRATED	WAIT	LONG_BACKGROUND	Pass	
95	Limit	S3	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
96	Limit	S3	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
97	Limit	S2	WAIT	INTERNET_ON	WAIT	RECEIVE_CALL	Pass	
98	Limit	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
99	Limit	S7	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
100	Limit	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
101	Limit	S7	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
102	Limit	S5	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON	Pass	
103	Limit	S10	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
104	Limit	S2	WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
105	Limit	S1	WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_NOT_CALIBRATED	Pass	
106	Limit	S5	NOT_WAIT	INTERNET_OFF	WAIT	GPS_ON	Pass	
107	Limit	S8	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
108	Limit	S6	WAIT	GPS_OFF	NOT_WAIT	RECEIVE_CALL	Pass	
109	Limit	S6	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_OFF	Pass	
110	Limit	S12	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	

111	Limit	S12	NOT_WAIT	GPS_ON	WAIT	GPS_CALIBRATED	Pass	
112	Limit	S1	WAIT	INTERNET_OFF	WAIT	ORIENTATION_LANDSCAPE	Pass	
113	Limit	S11	WAIT	GPS_ON	NOT_WAIT	GPS_OFF	Pass	
114	Limit	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
115	Limit	S11	NOT_WAIT	GPS_ON	WAIT	ORIENTATION_PORTRAIT	Pass	
116	Limit	S5	NOT_WAIT	GPS_ON	WAIT	INTERNET_OFF	Pass	
117	Limit	S8	NOT_WAIT	LONG_BACKGROUND	WAIT	INTERNET_ON	Pass	
118	Limit	S10	WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE	Pass	
119	Long	S2	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
120	Long	S9	WAIT	TAKE_A_PICTURE	NOT_WAIT	GPS_OFF	Pass	
121	Long	S2	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
122	Long	S3	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
123	Long	S6	WAIT	GPS_CALIBRATED	WAIT	LONG_BACKGROUND	Pass	
124	Long	S8	NOT_WAIT	INTERNET_ON	WAIT	TAKE_A_PICTURE	Pass	
125	Long	S8	WAIT	LONG_BACKGROUND	NOT_WAIT	RECEIVE_CALL	Pass	
126	Long	S3	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
127	Long	S3	WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
128	Long	S1	NOT_WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
129	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	LONG_BACKGROUND	Fail	
130	Long	S10	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
131	Long	S4	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
132	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_OFF	Pass	
133	Long	S10	WAIT	INTERNET_OFF	WAIT	GPS_CALIBRATED	Pass	
134	Long	S8	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Pass	
135	Long	S1	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
136	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
137	Long	S1	WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
138	Long	S11	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
139	Long	S5	WAIT	GPS_ON	NOT_WAIT	GPS_OFF	Pass	
140	Long	S7	WAIT	RECEIVE_CALL	WAIT	GPS_NOT_CALIBRATED	Pass	
141	Long	S1	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	GPS_CALIBRATED	Pass	
142	Long	S4	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
143	Long	S10	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
144	Long	S11	WAIT	GPS_ON	NOT_WAIT	GPS_CALIBRATED	Pass	
145	Long	S8	WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
146	Long	S4	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_ON	Pass	
147	Long	S10	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
148	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Pass	
149	Long	S8	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
150	Long	S9	WAIT	RECEIVE_CALL	NOT_WAIT	TAKE_A_PICTURE	Pass	
151	Long	S6	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
152	Long	S10	WAIT	INTERNET_OFF	NOT_WAIT	GPS_OFF	Pass	
153	Long	S1	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
154	Long	S11	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
155	Long	S1	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL	Pass	
156	Long	S3	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
157	Long	S12	NOT_WAIT	INTERNET_OFF	WAIT	TAKE_A_PICTURE	Pass	
158	Long	S8	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
159	Long	S2	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE	Pass	
160	Long	S4	WAIT	GPS_OFF	WAIT	TAKE_A_PICTURE	Pass	
161	Long	S6	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
162	Long	S7	NOT_WAIT	INTERNET_OFF	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
163	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_ON	Pass	
164	Long	S3	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	LONG_BACKGROUND	Pass	
165	Long	S2	WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
166	Long	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	

167	Long	S9	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
168	Long	S11	NOT_WAIT	GPS_ON	WAIT	LONG_BACKGROUND	Pass	
169	Long	S10	WAIT	GPS_CALIBRATED	WAIT	TAKE_A_PICTURE	Pass	
170	Long	S9	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	GPS_CALIBRATED	Pass	
171	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
172	Long	S4	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
173	Long	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	RECEIVE_CALL	Pass	
174	Long	S3	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_LANDSCAPE	Pass	
175	Long	S11	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_ON	Pass	
176	Long	S7	WAIT	ORIENTATION_PORTRAIT	WAIT	INTERNET_OFF	Pass	
177	Long	S10	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
178	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	RECEIVE_CALL	Pass	
179	Long	S6	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
180	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_ON	Pass	
181	Long	S6	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
182	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	INTERNET_ON	Pass	
183	Long	S12	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON	Pass	

Table C.8: Order 2 test result for Voice GPS Navigation application

Voice GPS Navigation: Live Driving Direction								
	Path	Setup	First Delay	First event	Second Delay	Second event	Verdict	Crash
1	Small	S1	WAIT	INTERNET_OFF	NOT_WAIT	TAKE_A_PICTURE	Pass	
2	Small	S10	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_PORTRAIT	Pass	
3	Small	S12	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	RECEIVE_CALL	Pass	
4	Small	S9	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	Pass_CALL	Pass	
5	Small	S7	WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
6	Small	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	LONG_BACKGROUND	Pass	
7	Small	S2	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_NOT_CALIBRATED	Pass	
8	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
9	Small	S3	NOT_WAIT	GPS_CALIBRATED	WAIT	GPS_OFF	Pass	
10	Small	S4	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL	Pass	
11	Small	S5	NOT_WAIT	GPS_ON	NOT_WAIT	RECEIVE_CALL	Pass	
12	Small	S9	WAIT	LONG_BACKGROUND	WAIT	GPS_CALIBRATED	Pass	
13	Small	S6	NOT_WAIT	INTERNET_ON	NOT_WAIT	INTERNET_OFF	Pass	
14	Small	S6	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	TAKE_A_PICTURE	Pass	
15	Small	S10	WAIT	ORIENTATION_PORTRAIT	WAIT	TAKE_A_PICTURE	Pass	
16	Small	S6	WAIT	TAKE_A_PICTURE	WAIT	ORIENTATION_LANDSCAPE	Pass	
17	Small	S5	WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Pass	
18	Small	S9	WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
19	Small	S4	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_CALIBRATED	Pass	
20	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	INTERNET_ON	Pass	
21	Small	S9	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_ON	Pass	
22	Small	S5	WAIT	GPS_ON	WAIT	GPS_CALIBRATED	Pass	
23	Small	S4	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
24	Small	S2	WAIT	INTERNET_ON	WAIT	GPS_OFF	Pass	
25	Small	S10	WAIT	TAKE_A_PICTURE	WAIT	INTERNET_OFF	Pass	
26	Small	S5	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
27	Small	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_OFF	Pass	
28	Small	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	TAKE_A_PICTURE	Pass	
29	Small	S11	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_ON	Pass	
30	Small	S1	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE	Fail	X
31	Small	S6	WAIT	RECEIVE_CALL	NOT_WAIT	GPS_CALIBRATED	Pass	
32	Small	S7	WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_OFF	Pass	
33	Small	S10	WAIT	INTERNET_OFF	WAIT	LONG_BACKGROUND	Pass	
34	Small	S11	WAIT	GPS_ON	WAIT	TAKE_A_PICTURE	Pass	
35	Small	S7	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Fail	X
36	Small	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_OFF	Pass	
37	Small	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_ON	Fail	X
38	Small	S3	WAIT	RECEIVE_CALL	NOT_WAIT	INTERNET_OFF	Pass	
39	Small	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT	Fail	
40	Small	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_CALIBRATED	Fail	X
41	Small	S4	WAIT	INTERNET_ON	NOT_WAIT	LONG_BACKGROUND	Pass	
42	Small	S7	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	INTERNET_OFF	Pass	
43	Small	S11	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
44	Small	S11	WAIT	GPS_ON	NOT_WAIT	INTERNET_ON	Fail	X
45	Small	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	INTERNET_OFF	Pass	
46	Small	S7	NOT_WAIT	TAKE_A_PICTURE	WAIT	LONG_BACKGROUND	Pass	
47	Small	S3	NOT_WAIT	GPS_CALIBRATED	NOT_WAIT	INTERNET_OFF	Pass	
48	Small	S2	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
49	Small	S4	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_CALIBRATED	Pass	
50	Small	S1	NOT_WAIT	INTERNET_OFF	WAIT	GPS_NOT_CALIBRATED	Pass	
51	Small	S11	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
52	Small	S7	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
53	Small	S2	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Pass	
54	Small	S8	WAIT	GPS_OFF	WAIT	GPS_ON	Pass	

55	Small	S6	WAIT	LONG_BACKGROUND	WAIT	ORIENTATION_LANDSCAPE	Pass	
56	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
57	Small	S2	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
58	Small	S2	WAIT	GPS_OFF	WAIT	INTERNET_ON	Fail	X
59	Small	S2	WAIT	LONG_BACKGROUND	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
60	Small	S12	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE	Pass	
61	Small	S12	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
62	Small	S11	NOT_WAIT	INTERNET_ON	WAIT	GPS_ON	Pass	
63	Small	S5	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_ON	Pass	
64	Small	S6	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
65	Small	S12	NOT_WAIT	INTERNET_OFF	NOT_WAIT	RECEIVE_CALL	Pass	
66	Perfect	S7	NOT_WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Fail	X
67	Perfect	S3	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	GPS_CALIBRATED	Pass	
68	Perfect	S9	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
69	Perfect	S9	NOT_WAIT	GPS_OFF	WAIT	LONG_BACKGROUND	Pass	
70	Perfect	S6	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
71	Perfect	S10	WAIT	GPS_CALIBRATED	NOT_WAIT	RECEIVE_CALL	Fail	X
72	Perfect	S8	NOT_WAIT	RECEIVE_CALL	NOT_WAIT	Pass_CALL	Pass	
73	Perfect	S1	NOT_WAIT	LONG_BACKGROUND	WAIT	GPS_OFF	Pass	
74	Perfect	S2	NOT_WAIT	RECEIVE_CALL	WAIT	LONG_BACKGROUND	Pass	
75	Perfect	S5	WAIT	GPS_ON	WAIT	ORIENTATION_LANDSCAPE	Pass	
76	Perfect	S11	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
77	Perfect	S5	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE	Pass	
78	Perfect	S7	WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_OFF	Pass	
79	Perfect	S7	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
80	Perfect	S9	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
81	Perfect	S9	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
82	Perfect	S12	WAIT	GPS_ON	WAIT	GPS_OFF	Pass	
83	Perfect	S4	WAIT	GPS_CALIBRATED	WAIT	ORIENTATION_LANDSCAPE	Pass	
84	Perfect	S8	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	INTERNET_ON	Pass	
85	Perfect	S12	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
86	Perfect	S5	WAIT	ORIENTATION_LANDSCAPE	WAIT	LONG_BACKGROUND	Pass	
87	Perfect	S8	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
88	Perfect	S1	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Fail	X
89	Limit	S3	WAIT	LONG_BACKGROUND	WAIT	TAKE_A_PICTURE	Fail	X
90	Limit	S9	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
91	Limit	S12	WAIT	GPS_ON	NOT_WAIT	INTERNET_OFF	Pass	
92	Limit	S9	WAIT	ORIENTATION_PORTRAIT	WAIT	ORIENTATION_LANDSCAPE	Pass	
93	Limit	S4	NOT_WAIT	GPS_OFF	WAIT	GPS_ON	Pass	
94	Limit	S1	WAIT	GPS_NOT_CALIBRATED	WAIT	LONG_BACKGROUND	Pass	
95	Limit	S3	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
96	Limit	S3	WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
97	Limit	S2	WAIT	INTERNET_ON	WAIT	RECEIVE_CALL	Pass	
98	Limit	S8	WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
99	Limit	S7	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
100	Limit	S3	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	
101	Limit	S7	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
102	Limit	S5	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON	Fail	X
103	Limit	S10	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Fail	
104	Limit	S2	WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
105	Limit	S1	WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_NOT_CALIBRATED	Pass	
106	Limit	S5	NOT_WAIT	INTERNET_OFF	WAIT	GPS_ON	Pass	
107	Limit	S8	NOT_WAIT	INTERNET_ON	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
108	Limit	S6	WAIT	GPS_OFF	NOT_WAIT	RECEIVE_CALL	Pass	
109	Limit	S6	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_OFF	Pass	
110	Limit	S12	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	

111	Limit	S12	NOT_WAIT	GPS_ON	WAIT	GPS_CALIBRATED	Fail	X
112	Limit	S1	WAIT	INTERNET_OFF	WAIT	ORIENTATION_LANDSCAPE	Pass	
113	Limit	S11	WAIT	GPS_ON	NOT_WAIT	GPS_OFF	Pass	
114	Limit	S8	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
115	Limit	S11	NOT_WAIT	GPS_ON	WAIT	ORIENTATION_PORTRAIT	Pass	
116	Limit	S5	NOT_WAIT	GPS_ON	WAIT	INTERNET_OFF	Pass	
117	Limit	S8	NOT_WAIT	LONG_BACKGROUND	WAIT	INTERNET_ON	Pass	
118	Limit	S10	WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE	Pass	
119	Long	S2	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
120	Long	S9	WAIT	TAKE_A_PICTURE	NOT_WAIT	GPS_OFF	Pass	
121	Long	S2	WAIT	ORIENTATION_LANDSCAPE	WAIT	ORIENTATION_PORTRAIT	Pass	
122	Long	S3	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
123	Long	S6	WAIT	GPS_CALIBRATED	WAIT	LONG_BACKGROUND	Pass	
124	Long	S8	NOT_WAIT	INTERNET_ON	WAIT	TAKE_A_PICTURE	Fail	X
125	Long	S8	WAIT	LONG_BACKGROUND	NOT_WAIT	RECEIVE_CALL	Fail	
126	Long	S3	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
127	Long	S3	WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
128	Long	S1	NOT_WAIT	GPS_OFF	WAIT	GPS_ON	Fail	
129	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	LONG_BACKGROUND	Pass	
130	Long	S10	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
131	Long	S4	NOT_WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Pass	
132	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	WAIT	GPS_OFF	Pass	
133	Long	S10	WAIT	INTERNET_OFF	WAIT	GPS_CALIBRATED	Fail	
134	Long	S8	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Fail	
135	Long	S1	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Fail	X
136	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
137	Long	S1	WAIT	INTERNET_OFF	WAIT	INTERNET_ON	Fail	
138	Long	S11	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
139	Long	S5	WAIT	GPS_ON	NOT_WAIT	GPS_OFF	Pass	
140	Long	S7	WAIT	RECEIVE_CALL	WAIT	GPS_NOT_CALIBRATED	Fail	
141	Long	S1	NOT_WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	GPS_CALIBRATED	Fail	
142	Long	S4	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
143	Long	S10	WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Pass	
144	Long	S11	WAIT	GPS_ON	NOT_WAIT	GPS_CALIBRATED	Fail	
145	Long	S8	WAIT	INTERNET_ON	WAIT	INTERNET_OFF	Fail	
146	Long	S4	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	INTERNET_ON	Pass	
147	Long	S10	NOT_WAIT	INTERNET_OFF	NOT_WAIT	INTERNET_ON	Pass	
148	Long	S7	NOT_WAIT	GPS_NOT_CALIBRATED	WAIT	GPS_CALIBRATED	Fail	X
149	Long	S8	NOT_WAIT	RECEIVE_CALL	WAIT	CANCEL_CALL	Fail	
150	Long	S9	WAIT	RECEIVE_CALL	NOT_WAIT	TAKE_A_PICTURE	Pass	
151	Long	S6	WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
152	Long	S10	WAIT	INTERNET_OFF	NOT_WAIT	GPS_OFF	Pass	
153	Long	S1	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Fail	X
154	Long	S11	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
155	Long	S1	WAIT	RECEIVE_CALL	NOT_WAIT	CANCEL_CALL	Pass	
156	Long	S3	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED	Fail	X
157	Long	S12	NOT_WAIT	INTERNET_OFF	WAIT	TAKE_A_PICTURE	Pass	
158	Long	S8	NOT_WAIT	GPS_OFF	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
159	Long	S2	NOT_WAIT	LONG_BACKGROUND	NOT_WAIT	TAKE_A_PICTURE	Fail	
160	Long	S4	WAIT	GPS_OFF	WAIT	TAKE_A_PICTURE	Pass	
161	Long	S6	WAIT	GPS_CALIBRATED	NOT_WAIT	GPS_NOT_CALIBRATED	Pass	
162	Long	S7	NOT_WAIT	INTERNET_OFF	NOT_WAIT	ORIENTATION_PORTRAIT	Fail	
163	Long	S12	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	GPS_ON	Pass	
164	Long	S3	NOT_WAIT	TAKE_A_PICTURE	NOT_WAIT	LONG_BACKGROUND	Pass	
165	Long	S2	WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Fail	
166	Long	S10	NOT_WAIT	GPS_OFF	NOT_WAIT	GPS_ON	Pass	

167	Long	S9	NOT_WAIT	TAKE_A_PICTURE	WAIT	RECEIVE_CALL	Pass	
168	Long	S11	NOT_WAIT	GPS_ON	WAIT	LONG_BACKGROUND	Pass	
169	Long	S10	WAIT	GPS_CALIBRATED	WAIT	TAKE_A_PICTURE	Pass	
170	Long	S9	NOT_WAIT	ORIENTATION_PORTRAIT	WAIT	GPS_CALIBRATED	Fail	
171	Long	S5	NOT_WAIT	RECEIVE_CALL	WAIT	Pass_CALL	Pass	
172	Long	S4	WAIT	GPS_CALIBRATED	WAIT	GPS_NOT_CALIBRATED	Pass	
173	Long	S2	WAIT	GPS_NOT_CALIBRATED	NOT_WAIT	RECEIVE_CALL	Pass	
174	Long	S3	NOT_WAIT	RECEIVE_CALL	WAIT	ORIENTATION_LANDSCAPE	Pass	
175	Long	S11	NOT_WAIT	RECEIVE_CALL	WAIT	GPS_ON	Pass	
176	Long	S7	WAIT	ORIENTATION_PORTRAIT	WAIT	INTERNET_OFF	Fail	X
177	Long	S10	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
178	Long	S4	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	RECEIVE_CALL	Pass	
179	Long	S6	WAIT	INTERNET_ON	NOT_WAIT	ORIENTATION_LANDSCAPE	Pass	
180	Long	S2	NOT_WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	INTERNET_ON	Fail	X
181	Long	S6	WAIT	ORIENTATION_LANDSCAPE	NOT_WAIT	ORIENTATION_PORTRAIT	Pass	
182	Long	S11	NOT_WAIT	ORIENTATION_PORTRAIT	NOT_WAIT	INTERNET_ON	Pass	
183	Long	S12	NOT_WAIT	TAKE_A_PICTURE	WAIT	GPS_ON	Pass	

Appendix D

Defect Report

Table D.1: Defect Report

Defect	Observed behavior	Expected behavior	Demonstration video
D01	The AUT starts initially with GPS not calibrated, as determined by the test case setup. Afterward, long background simulation is performed. The application normally returns to navigation. However, when the AUT recalculates the route to the destination (3:43 of the video) the route includes the location that simulates a possible inaccurate location as an intermediate destination.	The AUT starts initially with GPS not calibrated, as determined by the test case setup. Afterward, long background simulation is performed. The application normally returns to navigation. Then the AUT recalculates the route normally to the destination without including unwanted intermediate destinations in the route.	https://youtu.be/Kyb_GvtZIa0
D02	The AUT starts initially with GPS not calibrated, as determined by the test case setup. After 42 seconds of the video, the GPS is calibrated again. The application starts showing the user's position on the map but does not recalculate the route as expected until the end of the test case.	The AUT starts initially with GPS not calibrated, as determined by the test case setup. When the GPS is calibrated again, the application must again show the user's position on the map and recalculate the route from the current position to the destination.	https://youtu.be/3dCyGv5A0rU
D03	As soon as the AUT starts displaying the GPS navigation, the test case executes the long background simulation. The application resumes at 2:33 of the video. However, navigation is closed, and the Activity of choosing which map the user wants to use is reopened.	As soon as the AUT starts displaying the GPS navigation, the test case executes the long background simulation. The application resumes at 2:33 of the video. The application should continue to perform navigation normally.	https://youtu.be/YPeVHCesdq8
D04	The AUT starts initially with GPS not calibrated, as determined by the test case setup. Immediately when the test case starts to run, the GPS is calibrated. As soon as it returns to the route position (27 seconds of the video), the application includes in the route the location that simulates a	The AUT starts initially with GPS not calibrated, as determined by the test case setup. Immediately when the test case starts to run, the GPS is calibrated. As soon as it returns to the route position, the application must trace the route to the destination without including	https://youtu.be/LNUpD_2YKh4

	possible inaccurate location as an intermediate destination.	unwanted intermediate destinations in the route.	
D05	AUT starts initially with GPS off, as determined by the test case setup. As soon as the GPS is turned on at 28 seconds of the video, the navigation map is not rendered.	AUT starts initially with GPS off, as determined by the test case setup. As soon as the GPS is turned on, the application should indicate a message or show the path from the position received by the sensor to the destination.	https://youtu.be/qcyQvyaCNEc
D06	The AUT was running the test case normally until the ENVIAR tool simulated a long background. After resuming the execution of the application, navigation was interrupted at 3:51 of the video.	The AUT should normally execute throughout the test case.	https://youtu.be/WWdebzKautc
D07	AUT executes the test case normally until the ENVIAR tool simulates the device's orientation change (at 21 seconds of the video). From that moment on, the application no longer displays the arrow indicating the user's position.	The AUT should normally execute throughout the test case.	https://youtu.be/HZNwk10hX6k
D08	At 31 seconds of the video, AUT started calculating the route. The route was recalculated several times, as we can see at 52 seconds, at 01:06 and 01:16 of the video. The application was calculating the route until 2:15 when the application closed abruptly and displayed an error message.	The AUT should be able to recover from events sent by the ENVIAR tool without ending abruptly.	https://youtu.be/91Nnrlru8Dg
D09	During the setup of the test case, the ENVIAR tool leaves the AUT in the background. When starting the test case, the AUT returns to the foreground. At that moment (at 20 seconds of the video), the AUT ended the Navigation Activity.	During the setup of the test case, the ENVIAR tool leaves the AUT in the background. When starting the test case, the AUT returns to the foreground. The AUT should continue navigation normally.	https://youtu.be/kEt3SmhAUt4

D10	The AUT starts initially with GPS not calibrated, as determined by the test case setup. When the GPS is calibrated again (1:11 of the video), the application includes in the route a location that simulates a possible inaccurate location as an intermediate destination.	The AUT should not include an intermediate destination without the user's request.	https://youtu.be/fN2qexfHsDU
D11	After starting the test case at 25 seconds of the video, the application presented a slowness that made it impossible to use it.	The AUT should perform efficiently in order to monitor the user's displacement in real-time.	https://youtu.be/CY1tY8JdKoQ
D12	The AUT starts with the internet turned off according to the test case setup. After starting the execution (28 seconds of the video), the AUT was recalculating the route and no longer updated the user's position on the map even when the internet was turned on again.	A AUT inicia com a internet desligada conforme o setup do caso de teste. Após iniciar a execução, a AUT deveria continuar executando normalmente.	https://youtu.be/pbUmuBHjOtw
D13	The AUT starts with an internet connection, as specified in the test case setup. Although showing some slowness, AUT updates the user's position on the map. After dropping the internet connection, AUT is no longer able to recalculate the route to the destination. At approximately 4:30 in the video, AUT stops responding to stimuli in the environment.	It was expected that it would be possible to experience slowdowns because the speed programmed for this test case was very high (1000 km/h). It was expected that the AUT could not calculate the route to the destination without the internet because it needs the internet for this task. However, we did not expect the AUT to stop responding to stimuli from the environment.	https://youtu.be/iyLzHuiRTR8