Faculdade de Engenharia da Universidade do Porto

# Automated Pattern-Based Testing of Mobile Applications

*Author:* Inês Coimbra Morgado

*Supervisor:* Ana C. R. Paiva

*Co-supervisor:* João Pascoal Faria

*A thesis proposal submitted in fulfilment of the requirements for the definitive enrolment in the Doctoral Program in Informatics Engineering Doctor of Informatics Engineering*

April 2014

Universidade do Porto

# *Abstract*

Faculdade de Engenharia
Departmento de Engenharia Informática

Doctor of Informatics Engineering

**Automated Pattern-Based Testing of Mobile Applications**

by Inês Coimbra Morgado

In the last years the necessity to ensure the quality of mobile applications has increased partly due to the exponential increase of critical applications, such as e-banking, and, as for every other type of software, test automation is an important step towards this goal. Testing the Graphical User Interface (GUI) of an application is equally important as it is through it that the user is going to interact with the application. One useful technique is model-based Graphical User Interface (GUI) testing as it automatically generates test cases based on a model of the application's GUI. However, the manual construction of such model is a time consuming error prone task. Reverse engineering can be useful to aid in the automatic generation of part of such model.

One of the goals of this research work is to develop reverse engineering techniques and tools to obtain behavioral models of mobile applications' GUIs. Even though event-driven application, such as mobile ones, are a good target for dynamic exploration, static analysis may also be useful for identifying the widgets that have event handlers associated.

Since GUIs are usually designed based on common GUI patterns (combining structure and behavior), the reverse engineering process will be directed towards finding instances of such patterns. This work is part of The Pattern-Based GUI Testing project which has proved that it is possible to detect patterns on a system in order to ease its modelling and testing as it is possible to define the corresponding test strategy prior to the exploration. As such, this work also intends to test the application on the fly whenever behaviour patterns are identifying by running the corresponding test pattern on the application.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **FSM** | **F**inite **S**tate **M**achine |
| **GUI** | **G**raphical **U**ser **I**nterface |
| **MBGT** | **M**odel **B**ased **G**UI **T**esting |
| **MBT** | **M**odel **B**ased **T**esting |
| **PBGT** | **P**attern **B**ased **G**UI **T**esting |
| **RE** | **R**everse **E**ngineering |
| **SMV** | **S**ymbolic **M**odel **V**erification |
| **UI** | **U**ser **I**nterface |
| **V&V** | **V**erification and **V**alidation |

# Chapter 1

# Introduction

Since the release of the iPhone in 2007 [2] and of the first Android smart phone in 2008 [3], smart phones have started to greatly increase their mobile sales. In fact, in late 2013, smart-phones represented almost 60% of the mobile sales worldwide and, with Android and iPhone representing over 85% of the smart-phone sales [4]. Moreover, in mid-2013, Android's *Google Play* reached one million available applications and the number of downloads has crossed the fifty billion threshold [5] and in late 2013 Apple announced their *App Store* had also reached one million available applications and sixty billion downloads [6]. This market dimension makes it extremely important to ensure the quality of an application as it generates a high level of competitiveness and thus for one to get popular it must be as flawless as possible. Furthermore, there has also been an increase of the critical mobile application, such as banking, which makes it even more important to ensure its correctness.

Mobile applications have, as any other type of application, their own quirks regarding testing, such as the high amount of different events that need to be tested, and, as every other, companies want to spend as little resources as possible in the testing process. As such, it is important to automate this process.

Android already presents a tool to ease testing, Exerciser Monkey [7]. This is a fuzzy testing tool, *i.e.*, it stress tests an Android application and its user interface (UI) by sending a pseudo-random stream of user events into the system. However, this does not ensure the full testing of the application and it does not provide any coverage statistics.

The academics have also been working on this, focusing mainly on Android applications and less on iOS ones. The main reasons for this are the increase of popularity of Android and the availability of open-source applications and frameworks.

The main focus of mobile testing is UI testing, more precisely Graphical UI (GUI) testing, as this is the source of interaction with the user and where most errors occur. There are two main focuses on mobile GUI testing in the literature: automatic test case generation and automatic crawling. The former provides a set of test cases that can be run on the application. The latter tries to exercise as many aspects of the application as possible in order to find errors, such as crashes. One of the most popular techniques for automatic test case generation is Model-based Testing (MBT) [8], *i.e.*, it receives a model of the application's behaviour as input and outputs a test suite to be run on it. The problem of this technique is that it requires a model of the application and its manual construction is a time consuming and error prone task. As such, it is also preferable to automate this step as much as possible. There are several approaches focusing this. However they usually target Desktop [9–14] or web [15–17] applications. Nevertheless, there are a few who apply Model Based GUI Testing (MBGT), *i.e.*, MBT applied to GUIs), to mobile applications [18–20]. Several of these approaches use reverse engineering techniques to ease the extraction of information and its abstraction into a model.

Reverse engineering (RE) was first defined in 1985 by Rekoff [21] as "the process of developing a set of specifications for a complex hardware system by an orderly examination of specimens of that system". Five years later, Chikofsky and Cross [22] adapted this definition to software systems: "Reverse Engineering is the process of analysing a subject system to (1) identify the system's components and interrelationships and (2) to create representations of the system in another form or at a higher level of abstraction". Figure 1.1 depicts their representation of a common reverse engineering process.

Even though nowadays reverse engineering is considered helpful in several areas [23], such as testing, it initially surfaced associated with software maintenance as it eases system comprehension. This was considered extremely important as over 50% of a system's development is occupied with maintenance tasks [24–26] and over 50% of maintenance is dedicated to comprehending the system [27, 28].

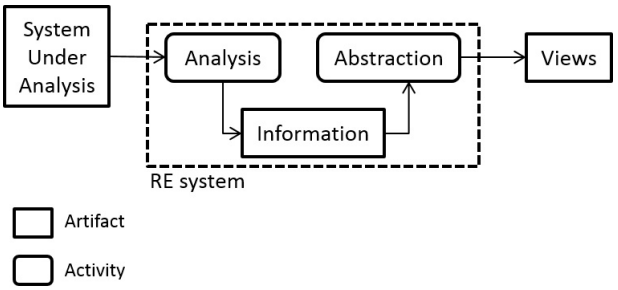


FIGURE 1.1: Architecture of a reverse engineering process

Reverse engineering has also proved to be useful, for instance, in coping with the Y2K problem, with the European currency conversion and with the migration of information systems to the web and towards the electronic commerce [29]. With the exploration of reverse engineering techniques, its usefulness grew from software maintenance to other fields, such as verification and validation and security analysis.

Even though reverse engineering techniques can also be used with malicious intents [30], such as removal of software protection and limitations or allowing unauthorised access to systems/data, developers may also use the same techniques in order to improve the software's safety, *e.g.*, auditing security and vulnerability.

There has been some studies on applying reverse engineering techniques to mobile applications. Hu *et al.* [31] identify bugs in the application, Amalfitano *et al.* extract an event-flow graph of the application considering [20] or not [32] system events, Joorabchi and Mesbah [33] and Yang *et al.* [34] extract a state machine relating the UI states with user interactions.

None of these approaches, however, try to take advantage of the existence of behavioural patterns in the application to facilitate their task. The pattern-based GUI testing (PBGT) project[1] aims at improving current MBGT methods and tools, contributing to construct an effectively applicable testing approach in industry and to contribute to the construction of higher quality GUIs and software systems. It is in the context of the PBGT project that this work arises. It is believed that mobile applications present behavioural patterns and that those ease the modelling and, thus, the testing task. As such, the work proposed in this document aims at extracting the behavioural model of a mobile application and at testing it by defining test strategies to be applied on the

[1] http://paginas.fe.up.pt/ apaiva/pbgtwiki/doku.php

fly as soon as a behavioural pattern is identified. During the exploration, whenever a behavioural pattern is detected a pre-defined test strategy will be applied to test it. Even though it is necessary to manually specify a catalogue of behavioural patterns and the corresponding test strategies, the catalogue is common to every application.

The remaining of this document is structured as follows. Chapter 2 presents the state of the art in reverse engineering classifying it according to an ontology also presented in this chapter. Chapter 3 presents some work that has already been done and the corresponding results. Chapter 4 defines some useful concepts, describes the approach and the research methodology, states the research hypothesis and presents the work plan. Chapter 5 draws some conclusions.

# Chapter 2

# Related Work on Reverse Engineering

Two different analysis of related work were performed for this document. The aim of the first one was to provide a general overview of the current state of the art on reverse engineering, henceforth mentioned as software reverse engineering. Given the dimension of this field of study, some restrictions for the selection of approaches were imposed. As such, the selection followed these steps:

1. Selection of top conferences and journals on software engineering in the last six years (2008 to October 2013), *i.e.*, conferences classified as CORE A and Journals classified as CORE A orA*[1] (a total of thirteen venues)

2. Selection of additional venues specific to reverse engineering and program comprehension in the last six years (2008 to October 2013)

3. Selection of papers from the selected journals and conferences based on their titles and abstracts: 104 papers initially selected

4. Refinement of the selection regarding the paper's content: 74 papers selected

The second analysis regarded the state of the art on mobile reverse engineering. The main differences from the software RE research methodology are: 1) only approaches

[1] classification according to CORE's (Computing Research and Education) ranking (http://core.edu.au/)

targeting mobile applications were considered (approaches for mobile versions of web applications were not taken into consideration); 2) an approach was considered regardless of where it had been published; and 3) no time restriction was imposed. Regardless of the latter, all the approaches are recent as iOS and Android smart phones are also recent: the first iPhone was released in mid 2007 [2] and the first smart phone running Android was released in late 2008 [3].

The final selection of papers for each venue is displayed in Tables A.1 and A.2 in Appendix A.

Tables B.1 and B.2 in Appendix B present the geographical distribution of software and mobile reverse engineering research, respectively.

## 2.1 Ontology for Classifying Reverse Engineering Approaches

A careful analysis was performed to extract the most important aspects to classify the selected approaches. In order to do so an ontology was defined. This ontology was based in the one presented by Cornelissen *et al.* [35]. The final list of the aspects considered for the classification consists on the following:

- *goal*: what is the main purpose of the approach?
- *target*: what is the target platform in which the approach works?
- *method*: what is the reverse engineering method used: static, dynamic, or hybrid?
- *information*: what type of information is extracted? (only for the mobile reverse engineering approaches)
- *output*: how is the obtained information represented to the user?
- *validation*: how is the proposed approach validated?

In order to thoroughly classify all the different approaches, some aspects of the ontology were refined for each of the selections (software and mobile reverse engineering): 1) the *goal*, *method*, *output* and *validation* aspects are common to both classifications; 2) the *target* aspect has different possible values (in software RE it only matters if the target

application is, for instance, web or mobile while in mobile RE it verifies if the application is for the iOS or the Android system); 3) the *information* aspect is only present in the mobile RE analysis as most approaches do not specify which specific information they are interested in even though such was stated in the mobile RE ones.

For each of the mentioned aspects, there are different sub-classifications. The next sections define each of these classifications.

### 2.1.1 Goal

Feature Location: A feature is usually any specific scenario of a system that is triggered by an external user [36]. A feature location approach extracts information on which methods and classes implement a certain feature. Most feature location approaches distinguish between feature specific methods, *i.e.*, methods that are only used by one feature, and omnipresent methods, *i.e.*, methods that are used by several features.

Pattern Identification: The concept of pattern was first defined by an architect, Christopher Alexander [37], as a representation of the "current best guess as to what arrangement of the physical environment will work to solve the problem presented". In the software engineering area, the definition of a pattern is conceptually the same but applied to the context of software engineering. Usually, design patterns are the ones considered [38]. Nevertheless, some approaches explore other types of patterns, such as behavioural or communication ones.

Model Recovery: Model recovery intends to recover a model, *i.e.*, a representation of (part of) a system. There are usually three kinds of information represented in these models: software architecture, business processes and system configuration. Nevertheless, it is possible to find other approaches extracting information such as layout or behavioural dependencies. Every approach whose sole purpose is to recover (extract) a model of a system without revealing any further purposes fits into this category.

Verification and Validation: Verification and Validation (V&V) processes are used to determine whether the development products of a given activity conform to the requirements of that activity and whether the product satisfies its intended use and user needs. Verification and Validation life cycle process requirements are specified for different integrity levels. Its scope encompasses systems, software, and hardware, and it includes

their interfaces [39]. In reverse engineering approaches, the two main sub-fields of V&V focused are Testing, such as generating a test suite, and Security as in ensuring, for instance, the application does not unwillingly access any user data.

Migration: Migration involves moving a set of instructions or programs from one platform to another. It may be motivated by different reasons such as the obsolescence of a technology, the pressure of users or the need to build a single coherent information system when merging companies [40]. This is usually important when dealing with legacy systems as it is often too costly to manually migrate the system or to build a new one. This classification was given to approaches which stated it as such.

Maintenance: Software maintenance is the modification of a software product after delivery to correct faults or to improve performance or other attributes [41]. Although in many cases the maintenance of a system may be the ultimate goal of a RE approach, they were only classified as such if that was explicitly mentioned.

Program Comprehension: One can argue that every RE approach must involve program comprehension (of part or of the complete system). Nevertheless, in this work only the approaches which did not fall under any of the other categories were considered as program comprehension, in particular, system's re-documentation, behaviour prediction (*e.g.*, how the system behaves in a different environment) and all the others which explicitly stated that is their goal.

### 2.1.2 Target

As mentioned, this aspect has different sub-classifications for software RE and for mobile RE approaches. However, the sub-classifications of this aspect in the mobile RE approaches (Android, iOS) are self explanatory and, as such, only the ones for the software RE aspect are defined here.

Mobile: Targeting mobile applications is very recent in the history of reverse engineering as it is itself a recent platform. Even though mobile phones have been around for the last three decades, smart phones just became common around 2003. In 2013 Android and iOS smart phones represented 90% of the global handset sales [42].

Web: On the other hand, web applications have been around for many years and are often the subject of reverse engineering. In this category, all kinds of web systems are acceptable, from simple web pages to Rich Internet Applications and web services.

Desktop: A desktop application is one that runs on the computer, *i.e.*, it does not run on mobiles nor on the web. However, they were only considered as such if they were not considered a legacy system and the focus of the authors was not on Java applications as in those cases they would be classified as such.

Other targets: There are some approaches which have specific targets that are not consistent with any of the ones previously defined, such as software line products, object-oriented systems and multi-threaded systems. These fall under the *other* category.

### 2.1.3 Method

The *method* aspect of the ontology classifies the approach according to its type of reverse engineering. There are four types of reverse engineering: static [43], in which the information is extracted from the source code or from the binary code; dynamic [44], in which the information is extracted from the system under execution; hybrid, a combination of both static and dynamic techniques; and historical [45], which obtains information on the evolution of the system kept in version control systems, such as SVN[2] or GIT[3]. The historical reverse engineering approaches have characteristics that are very different from the others and will not be subject of analysis in this document.

Static: Static reverse engineering is performed without actually executing the code using, for instance, parsers (usually returning a parse tree or an abstract syntax tree) or symbolic execution (simulating the execution according to a model, for instance). One of the advantages of using a static analysis approach is the possibility of extracting a complete sequence diagram of the program, *i.e.*, how processes and events of a system operate with one another and in what order [46]. Obtaining this diagram can provide 100% recall, *i.e.*, all the behaviour is extracted (no false negatives [47]), at the price of a low precision, *i.e.*, not all the extracted behaviour is real (high number of false positives [47]) [48]. Without disregarding the importance of the information extracted by static approaches, they are unable to extract information on the system's real behaviour, *i.e.*,

[2] svn.apache.org
[3] git-scm.com

its behaviour during runtime, and the effort of statically analysing dynamic types of object references is not conceivable for large programs [49, 50]. Moreover, conceptually related code is usually scattered over different source artefacts, making it a handful task to extract and analyse all the relevant information [51]. Another possible drawback is the necessity of availability of the source code, which is not always possible (when analysing grey-box components, for instance) [52].

Dynamic: Dynamic analysis consists in extracting information from the program in run time, such as data-dependent execution and late binding. This provides better precision and worst recall than static analysis as the number of false positives is reduced but the number of false negatives increases [48]. Dynamic analysis is usually more complex than static analysis. For instance, in one of the most recent works, Amalfitano *et al.* [32] explain the difficulty involved in dynamically extracting a model of an Android application, such as the unfamiliarity of the application's developers with the development framework due to its novelty (Activity, Service, Content Provider, *etc.*), with reports stating that bugs frequently appear due to incorrect management of the the Activity component [53]. Moreover, some of the challenges in automatically exploring an Android application are common to the exploration of other GUIs, such as the order in which the events are found and fired, preconditions of the application and of its running environment, when events are fired, when to stop the exploration (for instance, when a sufficient amount of the model is extracted) and how to represent the initial state of the application. One of the most popular techniques used by dynamic approaches is instrumentation, *i.e.*, the ability to monitor or measure the level of a product's performance, to diagnose errors and to obtain trace information, implemented in the form of code instructions that monitor specific components in a system [54]. These approaches usually define scenarios to be executed resulting in execution traces, which can be analysed in order to extract the information relevant to the work at hands. Considering this, it is not surprising that most dynamic approaches opt to represent information as sequence diagrams, even if just as an internal and intermediate representation and thus not considered in the *output* aspect.

Hybrid: Despite the advantages of the dynamic approaches over the static ones, dynamic analyses do not provide the behaviour of the whole program [48], as they do not provide 100% recall, but only a high precision. Hybrid analysis improve the completeness, scope and precision of the analysis [44] as they bring together the advantages of both static and

dynamic analysis. However, alike static approaches, they require access to the source or byte code. The techniques used in hybrid approaches are the combination of the ones used in both static (*e.g.*, parsing) and dynamic (*e.g.*, instrumentation) approaches as well as concolic execution.

### 2.1.4 Information

Crashes: A crash is an unavoidable event in which an application ceases to function properly. Part of testing an application is detecting these situations in order to enable the developers to correct them.

Bugs: A bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result or to behave in unintended ways. Most bugs arise from problems in the program's source code implementation. There can also be bugs related to the malfunction of frameworks or operating systems. However, in this context the bugs are considered to be in the application itself.

Event Sequence: As the name itself indicates, an event sequence is a set of user actions which, in a given order, provide testing or exercising of some features or functionalities of the application under analysis.

Inputs: In order to properly explore an application it is sometimes necessary to use a specific set of input values to enable a full coverage. These are values the user must use in certain states of the application in order to access a specific functionality.

Malicious Functionalities: Some problems found in applications may be related not only to behaviour wrongly implemented but also to functionalities with malicious intents, such as access to user data without the actual content of the end user.

Runtime Behaviour: Bugs, crashes and malicious functionalities are all part of an application's runtime behaviour even though they are unwanted behaviour. However, the information from and approach was classified as runtime behaviour if it simply intended to extract the general behaviour without specifying further.

### 2.1.5 Output

Sequence Diagram: The main purpose of a sequence diagram is to describe the interactions between objects and elements of a program or application in the sequential order that those interactions occur. However, their initial purpose was to provide a transition from requirements expressed as use cases to the next and more formal level of refinement. Use cases are often refined into one or more sequence diagrams [46].

Event Flow Graph: A simple way of representing the flow of events of an application's component, *i.e.*, in which order the events may occur, is an event-flow graph, which represents all possible interactions among the events in that component [55].

Control Flow Graph: A control flow graph is, as defined by Francis E. Allen [56], a representation of all paths that might be traversed through a program during its execution.

Finite State Machine: A finite state machine consists in a set of states and a transition function that maps each state to another one. When using these to represent a program's behaviour, it usually represents how an event modifies the state of the application under analysis, *i.e.*, the transitions represents events on the application and the states represent stages of the application.

Other Graphs: This classification was used when the output was a graph but not any of the already described.

Test Suits: A test suit is a set of tests required to verify and/or validate an application. This is the result of the approaches focusing on the automatic test cases generation, which is an important aid to assure an application's quality.

Report: Some approaches opt to output their information in the form of a report. This is to be expected, for instance, in feature location and pattern identification approaches or even when the clustering technique is used.

Other: This classification was used when no other sufficed, such as source code, in the case of migration-oriented approaches, or business process modelling notation graphs [57], in the case of business process recovery.

### 2.1.6 Validation

It is always important to validate any new approach, either by using case studies, quasi- or controlled experiments or surveys, by comparing it with other tools or by evaluating it according to accepted measures, such as precision, recall and code coverage.

Case Study: A case study consists in identifying key factors that affect the outcome and document activity and in collecting data in work environment or real world situation. In general, real life applications are used.

Quasi-Experiment: A quasi-experiment is an experimental design performed after the data collection in which there is no random assignment of subjects to groups, independent variables can not be fully controlled and values of key values are predefined [58].

Controlled Experiment: A controlled experiment consists in identifying key factors and manipulating them to test their effects on the outcomes. The subjects are randomly assigned to groups and the data collected from subject performance [58].

Survey: The main purpose of a survey is to obtain feedback on the approach and its results. This is obtained by questioning groups of people who are usually experts but may also be end users, depending on the goal of the approach.

Evaluation: There are some validation metrics to sustain the quality of an approach, such as precision, recall, performance or code coverage. This classification was assigned whenever any of these metrics were used.

Comparison with other tools or approaches: In order to evaluate an approach it is also useful to compare the results with others obtained by existing approaches or tools. If the experiments are performed in conditions as similar as possible it is a good indicator of which approach is better.

None: Even though it is rare, there may be cases when researchers do not validate their approach in any way. This happens, for instance, when the sole purpose of a paper is to present an approach but no implementation has been developed yet.

## 2.2 Software Reverse Engineering Approaches

This Section analyses the approaches on software reverse engineering, providing their classification according to the ontology defined in Section 2.1.

Table 2.1 presents, for each goal, its subcategories, when they exist, their definition and the papers related to each of them. Each approach is classified in only one goal.

TABLE 2.1: Classification of the approaches according to their goal

| Goals | **Sub-goals** | **Definition** | **References** |
|---|---|---|---|
| Feature Location | general purpose | identify the source code fragments implementing a particular feature | [59–69] |
| | specific for characteristics of product lines | a feature is a characteristic of the product line of a system, *e.g.*, performance or CPU frequency | [70–72] |
| Pattern Identification | design patterns | detect design patterns in the system | [48, 73–77] |
| | communication patterns | identify communication patterns in the system | [78] |
| Model Recovery | Software Architecture | recover a model representing the architecture of a system | [36, 79–81] |
| | Business Process | recover a model representing the business process of a system | [82] |
| | Configuration | recover a model representing the configuration of a system | [83] |
| | Other | recover a model representing something that is not architecture, business processes nor configuration of the system | [33, 84–90] |
| V&V | Security | verify if there is no unwanted access to information | [91–95] |
| | Testing | ensure the correct behaviour of the application | [17, 32, 52, 96–103] |
| Maintenance | | extract information relevant to the maintenance of the system | [104, 105] |
| Migration | | migrate a system between platforms or languages | [106–110] |
| Program Comprehension | Normal | extract information relevant to the comprehension of a software system | [51, 111–121] |
| | Predictions | predict how a system behaves under certain conditions | [122, 123] |
| | Re-documentation | produce documentation of a system | [124] |

It is possible to state that in the set of papers aiming at pattern identification, only Kienle *et al.* [78] did not focus on design patterns but on communication ones instead. Moreover, no approach focused on behavioural patterns. The four goals in which the researchers mostly focus are: feature location, model recovery, verification and validation and program comprehension. It is interesting to note that even though maintenance was the initial goal of reverse engineering, nowadays not many approaches present it as their main purpose. Nevertheless, one must remember that in order to maintain a system it is necessary to first comprehend it. Thus, it is likely that the results of some of the approaches classified as program comprehension will be used in maintenance. Besides, the information obtained in model recovery and feature location approaches may also be useful for program comprehension.

Table 2.2 presents which approaches focus each type of target. Each approach has only one target, with the exception of Madsen *et al.*'s [119], which targets both Web and Desktop applications. However, they only target, in fact, JavaScript applications, which includes Windows 8 standalone applications.

TABLE 2.2: Classification of the approaches according to their target

| **Target** | **References** |
|---|---|
| Mobile | [32, 33, 89, 99, 101, 102] |
| Web | [17, 66, 68, 82, 84, 85, 87, 90–92, 94, 97, 98, 103, 112, 117, 119, 121] |
| Desktop | [59, 61, 119] |
| Other | [36, 48, 51, 52, 60, 62–65, 67, 69–81, 83, 86, 88, 93, 95, 96, 100, 104–111, 113–116, 118, 120, 122–124] |

There is a clear tendency of researchers towards web applications, which is easily explained taking into consideration the amount of information available on the web and the quantity of accesses there are daily. The migration of information from desktop application to the cloud (web) explains the low number of approaches targeting this type of applications. The mobile platform is still recent and, even though some of the approaches already focus iOS and Android application it is expectable to see a growth of this number in a near future.

Table 2.3 presents the classification of the approaches according to its method. An approach applies only one method.

Each method of reverse engineering, being it static, dynamic or hybrid, has its own advantages and disadvantages and enables the recovery of different types of information

TABLE 2.3: Classification of the approaches according to the method aspect

| **Method** | **References** |
|---|---|
| Static | [63, 65, 68, 70, 72, 74, 75, 78–81, 83, 88, 90, 94, 96, 100, 103, 108–113, 115, 116, 119, 120, 123] |
| Dynamic | [17, 32, 33, 48, 51, 52, 62, 66, 67, 82, 84, 85, 89, 95, 99, 102, 104, 105, 107, 114, 124] |
| Hybrid | [36, 59–61, 64, 69, 73, 76, 77, 87, 91–93, 97, 98, 101, 105, 106, 112, 117, 118, 121, 122] |

as stated in Section 2.1. Therefore, it is natural to verify that the numbers of approaches that followed each of these methods is similar. Nevertheless, the higher number of static approaches can be explained by the fact that it has existed for more time than the others.

It is also important to identify the techniques used by each approach. In reverse engineering a technique may be used in the context of static or dynamic analysis. Concolic execution is an exception as it is a hybrid technique itself. Table 2.4 defines the most common techniques and presents which approaches apply them. Each approach may use several techniques.

Regarding static techniques there is a tremendous preference for parsing, which is part of the foundations of static reverse engineering, as stated in Section 2.1. Instrumentation and trace analysis go together in several approaches as one of the main goals of the former is to provide execution traces for later analysis. The same goes for automatic concrete execution and event handling, as when automatically executing an application, it is usually necessary to detect when an event is fired.

It is worth mentioning that even though the approaches of Amalfirano *et al.* [84, 85] and Machiry *et al.* [102] are classified as using instrumentation in Table 2.4 they apply instrumentation to the environment (*i.e.*, the browser (Amalfitano *et al.* [84, 85]) and the Android SDK framework Machiry *et al.* [102]), and not to the actual system. Besides these, only Rohatgi *et al.* [61] uses instrumentation on the environment but it provides the option to also use instrumentation on the application itself.

As stated in Chapter 1, reverse engineering has two phases: the extraction of information and the abstraction of this information in other forms of representation. The techniques present in Table 2.4 are relevant for the first phase. Table 2.5 presents some techniques for the second phase and which approaches use them. In this table, pattern identification

TABLE 2.4: Classification of the technique according to the context in which it is applied and of the approaches according to the techniques they use

| Context | Technique | Definition | References |
|---|---|---|---|
| Static | parsing | formal analysis by computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other | [60, 61, 63, 65, 68, 69, 72–80, 87, 88, 90, 92, 94, 96, 97, 100, 105, 109, 110, 112, 113, 115, 118, 119, 121, 123] |
| | information retrieval | obtain information resources relevant to an information need from a collection of information resources by searching on meta-data or on full-text (or other content-based) indexing | [59, 60, 64, 70, 72, 81, 106, 120, 121] |
| | grammar transformation | define a set of rules that transforms certain symbols into others | [68, 92, 108, 111, 113] |
| | points-to analysis | establish which pointers or heap references can point to which variables or storage locations | [83, 116, 119] |
| | symbolic execution | analyse a program to determine what inputs cause each part of a program to execute | [91, 93, 103, 121] |
| Dynamic | instrumentation | ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information, implemented in the form of code instructions that monitor specific components in a system | [36, 48, 51, 52, 61, 62, 64, 66, 67, 69, 73, 76, 82, 84, 85, 89, 91–93, 95, 97, 102, 104–107, 112, 117, 118, 122, 124] |
| | automatic concrete execution | automatic execution of a program/application | [17, 32, 87, 99, 102] |
| | trace analysis | analyse the information contained in collected execution traces | [36, 48, 51, 59–62, 64, 66, 67, 73, 76, 77, 82, 84, 85, 89, 91, 93, 95, 104–106, 112, 114, 118, 124] |
| | event handling | handle the system's events as they appear | [17, 32, 33, 82, 84, 85, 89, 98, 99, 101, 102, 104, 117, 118, 121, 124] |
| | pattern mining (information retrieval) | find existing patterns in data. In this context patterns often mean association rules | [66, 67] |
| Hybrid | concolic execution | uses both symbolic and concrete execution to solve a constraint path | [91, 101] |

and feature location appear as techniques as some approaches use them as steps to achieve their final goal and not as a goal in itself. For instance, Ceccato [106] uses feature location in order to ease the migration task. Each approach may use several techniques.

TABLE 2.5: Classification of the approaches according to the techniques used in the second phase of the reverse engineering process

| **Techniques** | **Definition** | **References** |
|---|---|---|
| Clustering | group a set of objects so that objects in the same group (called cluster) are more similar to each other than to those in other clusters | [36, 52, 71, 80–82, 95, 111, 112, 114, 116, 118] |
| Mapping | associate attributes/characteristics within a system | [48, 51, 67, 75, 83, 85, 91, 93, 105, 110, 124] |
| Program proving | formally prove a program meets its specifications, *e.g.*, model checking and theorem proving | [76, 86] |
| Formal concept analysis | principled way of deriving a concept hierarchy or formal ontology from a collection of objects and their properties | [67, 106] |
| Pattern identification | use pattern identification as means to a goal and not as end goal | [63, 65–67, 118] |
| Feature location | use feature location as means to a goal and not as end goal | [36, 81, 104, 106] |
| Machine learning | branch of artificial intelligence which concerns the construction and study of systems that can learn from data | [81, 122] |

Table 2.6 presents the different types of outputs and the approaches using them. Each approach may have more than one type of output.

TABLE 2.6: Classification of the approaches according to the output produced

| **Output** | **References** |
|---|---|
| Sequence Diagram | [101, 102, 114, 118] |
| Event Flow Graph | [96, 97] |
| Finite State Machine | [17, 33, 84, 85, 89, 90] |
| Control Flow Graph | [123] |
| Other Graphs | [52, 63, 67, 69, 87, 88, 91, 113, 117, 119] |
| Test Suit | [32, 93, 98, 99] |
| Report | [32, 36, 59–62, 64–66, 68–71, 73–78, 81, 93, 94, 99, 100, 103, 104, 107, 111, 116] |
| Other | [32, 33, 48, 51, 72, 79, 80, 82, 83, 86, 92, 93, 98, 99, 105, 106, 108–110, 112, 115, 120–122, 124] |

After analysing Table 2.6, it is possible to conclude that the most used form of output is the report. This is to be expected, for instance, in feature location and pattern

identification approaches. Even though model recovery approaches are supposed to extract the intended model, one of these approaches, Patel *et al.*'s [36], outputs a report due to a clustering step. The output of some approaches is classified as *other* as they do not fall under any other category, such as source code, in the case of migration-oriented approaches like Trudel *et al.*'s [110], or business process modelling notation graph [57], in the case of business process recovery approaches like Di Francescomarion *et al.*'s [82].

Table 2.7 presents the classification of the several approaches according to the validation they use. Each approach may present more than one type of validation.

TABLE 2.7: Classification of the approaches according to how they were validated

| **Validation** | **References** |
|---|---|
| Case Study | all except [62, 83, 96, 108, 111, 117] |
| Quasi-Experiment | [116] |
| Controlled Experiment | [90, 97, 106, 116] |
| Survey | [117] |
| Evaluation | [17, 36, 51, 59, 60, 64, 65, 70, 72–76, 79, 81, 83, 87, 95, 98, 100, 102, 105, 107, 110, 111, 117, 119, 123, 124] |
| Comparison with other tools or approaches | [32, 68, 83, 87, 91, 93, 94, 102, 105, 123] |
| None | [62, 88, 96, 108, 121] |

Nearly all authors validate their approaches, even if just with a case study, which is, in fact, the most popular validation method. Even when presenting a quasi- or controlled experiment, a case study is usually also present. Moreover, almost half of the approaches present some sort of self-validation such as calculations on performance, precision or recall. However, only a minority compares their results with the ones obtained by other approaches.

## 2.3 Mobile Reverse Engineering Approaches

From the launching of iPhone and the first Android phone in 2007 and 2008, respectively [2, 3], there has not been many mobile reverse engineering approaches. In fact, the first one is only from November 2010.

Table 2.8 presents the classification of the approaches here presented according to the ontology defined in Section 2.1.

TABLE 2.8: Classification of the mobile RE approaches according to the ontology

| | | Papers | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Aspect | Classification | [125] | [32] | [20] | [126] | [101] | [33] | [34] | [127] | [31] | [102] |
| Goal | V&V | x | x | x | x | x | | | x | x | x |
| | Model Recovery | | | | | | x | x | | | |
| Target | Android | x | x | x | x | x | | x | x | x | x |
| | iOS | | | | | | x | | | | |
| Method | Static | | | | x | | | | | | |
| | Dynamic | x | x | | | | | | | x | x |
| | Hybrid | | | x | | x | x | x | x | | |
| Technique | Crawling | x | | x | | | x | x | | | x |
| | Ripping | | x | | | | | | | | |
| | Instrumentation | x | x | x | | | | | x | x | x |
| | Parsing | | | | | | | x | | | |
| | Trace analysis | | | | | | | | | x | |
| | Event Handling | | | | | x | | | | x | |
| | Event Simulation | x | x | x | | | x | x | | x | x |
| | Pattern Identification | | | | x | | | | | x | |
| | Clustering | x | | | | | | | | | |
| | Test Generation | x | x | x | | | | | | x | |
| | Code Injection | | | | | | x | | | | |
| | Code Replacement | | | x | | | x | | | | |
| | Reflection | | | x | | | x | | | | |
| | Comparison of Interface | x | | | | | x | | | | |
| | Data Mining | | | | x | | | | | | |
| | Concolic Execution | | | | | x | | | x | | |
| | Crash Detection | x | x | x | | | | | | | |
| Output | Test Suit | x | x | x | | | | | | | |
| | Sugestions | | | | x | | | | | | |
| | Call Graph | | | | | x | | | x | | |
| | FSM | | | | | | x | x | | | |
| | Report | | | | x | | | | | x | x |
| Information | Crash Detection | x | x | x | | | | | | | |
| | Bugs | | x | x | | | | | | x | |
| | Event Sequence | | | | | x | | | x | | |
| | Runtime Behaviour | | | | | | x | x | | | |
| | Inputs | | | | | | | | | | x |
| | Malicious Functionalities | | | | x | | | | | | |
| Validation | Case Study | x | x | x | x | x | x | x | x | x | x |
| | Comparison With Other Approaches/Tools | | x | x | | | | x | | x | x |
| | Evaluation | | x | x | | | | x | | | x |

In this set of approaches there is a clear inversion of the tendency depicted in software RE approaches regarding the *method* aspect as only Batyuk *et al.* [126] follow a static approach, targeting the problem of detecting and mitigating unwanted activities, *i.e.*, they try to identify possible security vulnerabilities, such as unwanted access of user data, and present a report with suggestions of improvement.

On the other hand, the tendency of using instrumentation is still present. All the dynamic approaches apply instrumentation, even if it is to the Android SDK framework [128], like Machiry *et al.* [102], who generate valid testing input values for the application, or to the virtual machine where the application is being run, like Hu *et al.* [31], who aim at identifying bugs. Unlike their work in [84], in which Amalfitano *et al.* targeted rich internet applications, in [32, 125] they apply instrumentation to the application under analysis itself. Even in following a hybrid approach to generate event sequences to test the application Anand *et al.* [127] apply instrumentation both to the application under analysis and to the Android SDK framework. It is also verifiable that most approaches, specially the most recent ones, opt for an hybrid approach in order to benefit from both worlds.

Amalfitano *et al.*'s work in 2011 [125] aimed at automatically generating test cases and detecting crashes by crawling the application. The instrumentation was used to obtain analysable logs in order to detect the origin of a crash. In 2012 [32], with the same idea in mind, they applied Memon's GUI Ripper [55, 129] to Android applications, which rips the application and then analyses each part separately simulating user events. The main difference in the results was their new capability of detecting some bugs besides crashes. In 2013 [20] they decided to add an analysis of the behaviour of the application in the presence of system events. To do so they opted again to crawl the application and replaced the Android Sensor framework with an ad hoc version to ease the task of injecting this new type of events.

Alike Anand *et al.* [127], Jensen *et al.* [101] also attempt to obtain event sequences to test the application with a hybrid approach. However, Anand *et al.* aim at obtaining a set of event sequences with the higher coverage percentage possible and Jensen *et al.*'s idea is to find a feasible path that enables the testing of a given code statement that has not yet been reached by other testing techniques, *i.e.*, Jensen *et al.*'s approach complements other approaches which do not present 100% coverage.

Joorabchi *et al.* [33] and Yang *et al.* [34] are the only ones who claim their goal is to obtain a model of the application's UI. Naturally other approaches also have this purpose, like Amalfitano *et al.*'s [32], but Joorabchi *et al.* and Yang *et al.* make no effort of trying to test the application or to generate event sequences. Their main goal is to obtain a finite state machine representing the behaviour of the application. According to Yang *et al.* [34] the main difference between their approach and Joorabchi *et al.*'s [33] is that the latter has no means of identifying which GUI elements are actionable and which events these elements support.

Joorabchi *et al.* [33] are the only ones who target iOS applications.

All approaches do case studies with at least one application, half of them compare their results with the ones from other approaches [34], existing tools [32, 102], with data obtained by manual analysis [31, 102] or even their own previous work [20] and almost half the approaches do some kind of evaluation, being it coverage percentage the most popular.

## 2.4 Conclusions

There are two reverse engineering approaches dealing with patterns: Amalfitano *et al.* [20] and Batyuk *et al.* [126]. Amalfitano *et al.* define a set of event sequences to test situations like an incoming call. The term pattern is used because these event sequences can be applied to any application. Batyuk *et al.* [126] apply pattern identification to detect malicious intents of the application. However, none of these approaches try to take advantage of the existence of behavioural patterns in the application to facilitate their task, which is one of the goals of the approach described in this document. In fact, none of the software reverse engineering approaches who aimed at pattern identification focus on behavioural patterns neither.

Moreover, only Amalfitano *et al.* [20] and Machiry *et al.* [102] consider the effect of system events. However, Amalfitano *et al.*'s goal is to generate a test suite and to detect crashes and Machiry *et al.* attempt at recovering input data to enable testing, while the approach in this document will output the behavioural model of the application as well as a testing report.

# Chapter 3

# Previous Work

There is already some work produced and some experiments conducted in reverse engineering and in pattern-based testing even though the platform in which these approaches were applied were Desktop and Web, respectively. Nevertheless, they provide some insight on the work to be produced. This Chapter describes these works.

## 3.1 GUI Reverse Engineering for Visual and Formal Models

Previous experiments with reverse engineering desktop applications have already provided some interesting results. In [135], a dynamic approach with a fully automatic exploration was followed. It consisted in trying to obtain a model as complete as possible of the GUI of a desktop application, using the Microsoft Notepad application as a case study. This outputted, apart from a tree representing the structure of the GUI, a window graph, a navigation graph and a dependency graph. The window graph presents the different windows opened during the exploration and from which windows it is possible to access other windows. An example of such graph obtained by an exploration of the Notepad application's GUI is depicted in Figure 3.1. The navigation graph comprehends the set of elements relevant to the navigation and the actions required to access the different elements. An example of this graph for the same exploration is depicted in Figure 3.2. The dependency graph relates the different elements according to dependencies among them, such as the change of the *enabled* status due to an interaction with a

different element. An example is in Figure 3.3. Apart from the structure tree, all these graphs are defined in GraphML [136].

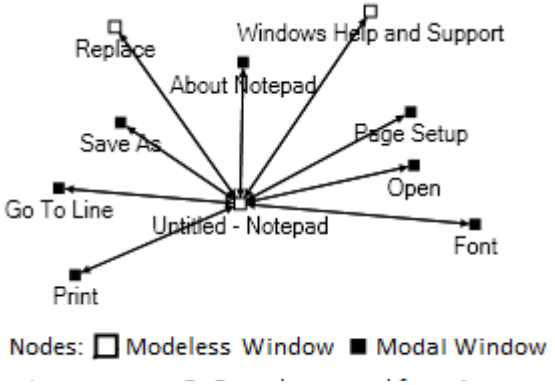


FIGURE 3.1: Visual representation of a window graph

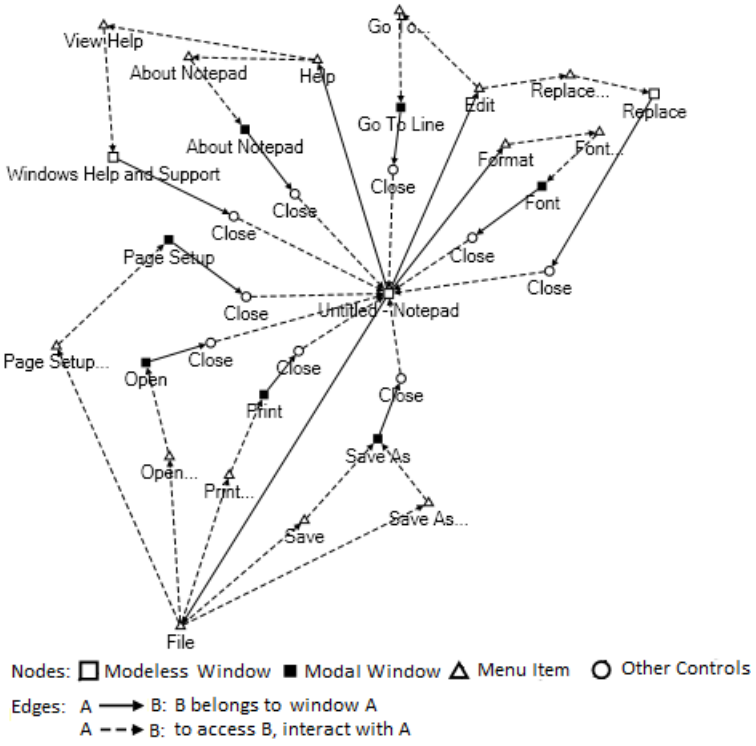


FIGURE 3.2: Visual representation of a navigation graph

Figure 3.4 depicts the overall approach used for the exploration and for the abstraction of the system under analysis.

This experiment enabled obtaining some preliminary results which were useful for the comprehension of the system, for future automatic test generation and to provide some

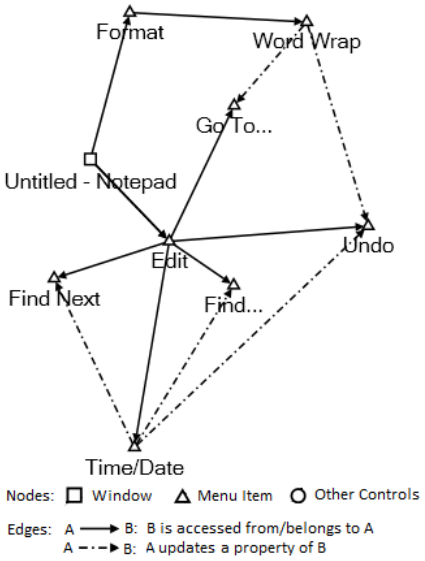


FIGURE 3.3: Visual representation of a dependency graph

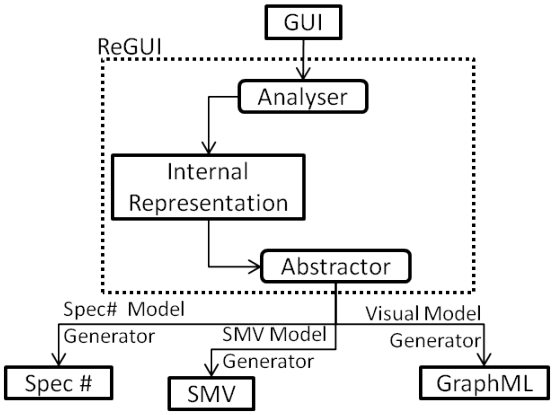


FIGURE 3.4: Architecture and outputs obtained in the RE process

insight in the difficulties and capabilities of reverse engineering approaches. Still in [135], a minimalist version of a Spec# model [137] built based on the extracted information and graphs was also outputted. An example of this is depicted in Figure 3.5.

This model can be used for MBGT after being manually verified to ensure it is consistent with the system's specification. Furthermore, in [14] a formal verification of some properties of the GUI of the Notepad application was performed using a symbolic model verification (SMV) [138] model automatically derived from the graphs previously extracted. In [14] the model checking technique used was symbolic model checking [139] with different properties expressed in Computation Tree Logic, a propositional temporal

```
namespace WindowUntitled___Notepad;
var windowUntitled___Notepad = 1;
var menu_itemFile = 1;
var menu_itemSave = 3;
[Action] void Menu_itemFile ()
      requires menu_itemFile == 1;{
            menu_itemSave = 1;
      };
[Action] void Menu_itemSave()
      requires menu_itemSave == 1;{
            menu_itemSave = 3;
            WindowSave_As.windowSave_As = 1;
      };

namespace WindowSave_As;
var windowSave_As4 = 3;
var buttonClose = 3;
[Action] ButtonClose ()
      requires buttonClose == 1;{
            buttonClose = 3;
            WindowUntitled___Notepad.
                  windowUntitled___Notepad = 1;
      };
```

FIGURE 3.5: Sample of the Spec# formal model generated

logic, and modelling the system as a FSM. Properties verification can be useful in several fields, such as usability analysis and improvement [140, 141]. In [14] the properties verified were deadlock freeness, *i.e.*, if regardless of the current state the system has always a way of leaving it, possibility of returning to the initial state, *i.e.*, if regardless of the current state it is always possible to return to the initial state, the main window, and if in $x$ steps from the main window it was possible to reach a window. Part of the SMV model obtained is depicted in Figure 3.6.

The works of Paiva *et al.* [12] and Grilo *et al.* [142] were the basis for the approach presented in this section.

## 3.2 Pattern-based GUI Reverse Engineering

In [135] and [14] the nodes of the different graphs obtained (window, navigation and dependency graphs) corresponded to GUI elements.

However, a different output could relate different states of the GUI with the events which provoke a modification in that state. In [143] an experiment was conducted in which a state represented a window and its enabled elements. This provided a different perspective than the one from [135] and [14]. However, this may arise a problem of ambiguity.

```
MODULE main
    VAR
        state: 1..20;
        follow: getNextState(state);
        moreInfo: getInfo(state);

       ASSIGN
        init(state) := 1;
        next(state):=
            case
                state = 1: {2, 4, 6, 7, 9, 11, 13, 15, 17, 19};
                --from state 1, it is possible to go to states 2 (Open...),
                --4 (Save), 6 (Save As...), 7 (Page Setup), 9 (Print...),
                --11 (Replace...), 13 (Go To...), 15 (Font...),
                --17 (View Help), 19 (About Notepad)
                state = 2: 3;  --goes to window Open
                state = 3: 1;  --goes to the main window
                state = 4: 5;  --goes to window Save As
                state = 5: 1;  --goes to the main window
                state = 6: 5;  --goes to window Save As
                state = 7: 8;  --goes to window Page Setup
                state = 8: 1;  --goes to the main window
                state = 9: 10;  --goes to window Print
                state = 10: 1;  --goes to the main window
                state = 11: 12;  --goes to window Replace
                state = 12: 1;  --goes to the main window
                state = 13: 14;  --goes to window Go To Line
                state = 14: 1;  --goes to the main window
                state = 15: 16;  --goes to window Font
                state = 16: 1;  --goes to the main window
                state = 17: 18;  --goes to window Windows Help and Support
                state = 18: 1;  --goes to the main window
                state = 19: 20;  --goes to window About Notepad
                state = 20: 1;  --goes to the main window
            esac;
```

FIGURE 3.6: State machine in SMV

For instance, if the result of a search differs mainly on the content of the text and of the search text even though both come from the same window, the Find window, then the same event (clicking on button Find, for instance) could provoke a transition to a different state even though the source state was apparently the same, *i.e.*, an ambiguity in the model. In [143], a machine learning technique called Inductive Logic Programming [144, 145] was used in order to solve these ambiguities. The main idea was to identify ambiguities in the model and match them to previously defined patterns which indicate how to solve that same ambiguity and would modify the model accordingly. Figure 3.7 depicts part of the FSM obtained from a sample application before (a) and after (b) the ambiguity problem was resolved.

In order to apply ILP, all states, transitions and patterns were expressed in a declarative language, Prolog [146]. Even though the prolog code representing the states and the transitions can be automatically derived from the model, the patterns, alike every pattern identification approach, have to be manually defined. Nevertheless, they are reusable. The approach followed is summarised in Figure 3.8.

This approach was further explored in [16] by Nabuco *et al.* using execution traces

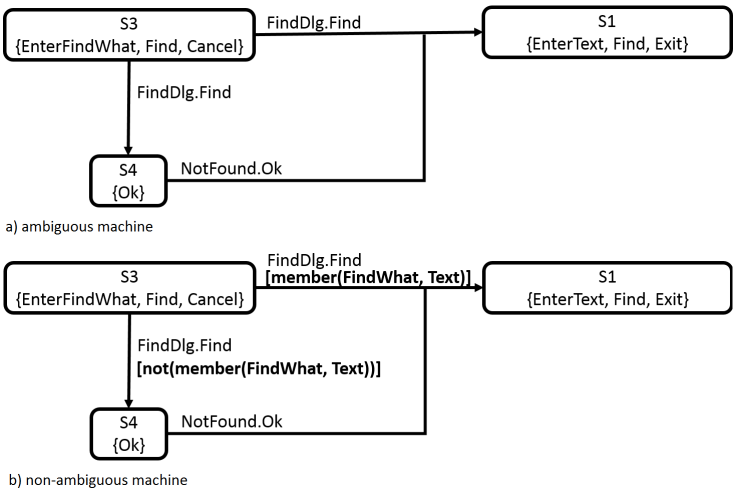


FIGURE 3.7: Sample of the state machine with (a) and without (b) ambiguity

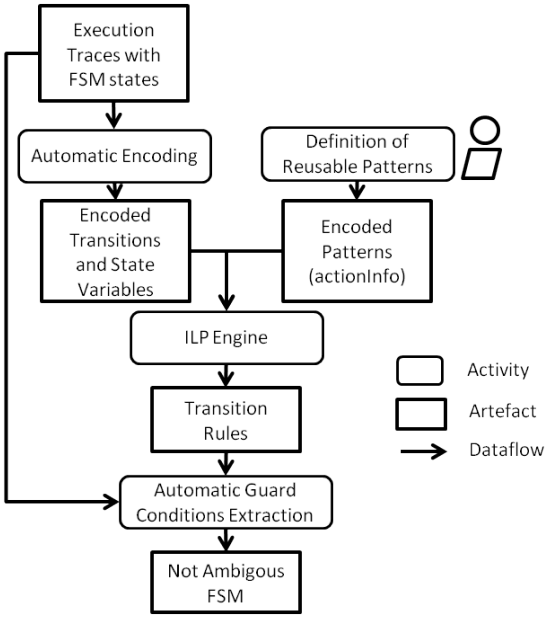


FIGURE 3.8: Architecture of the ILP approach

extracted from manually exploring the Amazon website[1] using the Selenium IDE[2] and thus validating the approach with a real case study.

[1] http://www.amazon.co.uk/
[2] http://www.seleniumhq.org/

## 3.3 Pattern-based GUI Testing

Regarding the test pattern definition, there is also some work already done. Moreira *et al.* [147] defined a new domain specific language, PARADIGM, to gather applicable domain abstractions (*e.g.*, test patterns), to allow specifying relations between them and also to provide a way to structure the models in different levels of abstraction to cope with complexity. Monteiro and Paiva [148] presented a modelling environment, PARADIGM-ME, to model UI patterns that can be found in web applications in the PARADIGM language. The idea is to enable a previous modelling of the patterns so that when identifying a certain pattern in the application under analysis, the corresponding modelling is already known. The final model is useful for MBGT. These works are also part of the PBGT project. A first experiment on behavioural patterns was proposed by Cunha *et al.* [149] previous to the starting of this project.

## 3.4 Conclusions

The work in reverse engineering, even though applied in desktop applications and not on mobile applications, enabled a first contact with reverse engineering techniques at the same time as providing some insights on what kind of information can be extracted and how reverse engineering can really be useful. Moreover, the experiments conducted raised some of the difficulties one needs to deal with when dynamically reverse engineering an application, namely the behaviour extracted depends on the order of the exploration and the difficulties of dynamically identifying an element.

The work on patterns already proves that identifying patterns in an application may be useful for understanding it. The exploration of ILP techniques proved its usefulness on disambiguating models obtained with the reverse engineering process and the current results on the PBGT project show the usefulness of the definition and application of test patterns to test web applications and thus, it is reasonable to assume they can also be useful in testing other types of applications such as mobile ones.

# Chapter 4

# Approach and Methodology

This Chapter presents the approach to be followed in the research work, the research hypothesis, the research methodology and the work plan. However, before describing the work at hands it is important to provide the definition of some concepts essential to its good understanding.

## 4.1 Definitions

This section defines some useful concepts.

Lifecycle The life cycle refers to the process-related states of an activity, such as running and paused, and the allowed transitions between these states [130]. An activity is "a single, focused thing that the user can do" [1] and that usually has a view associated so that the user can interact with it. It indicates the states of each activity and which methods are responsible for the transition between them. Figure 4.1 depicts the life cycle of an Android Activity.

Event An event in a mobile application can be of two types: an UI event, *i.e.*, an event provoked by an interaction of the user with the application, such as tapping a widget; or a system event, *i.e.*, an event provoked by something external to the application, such as the detection of a new available network, an incoming call or message and the modification of the orientation of the phone (from vertical to horizontal, for instance). Each of these events may or may not have an impact on the application behaviour. For

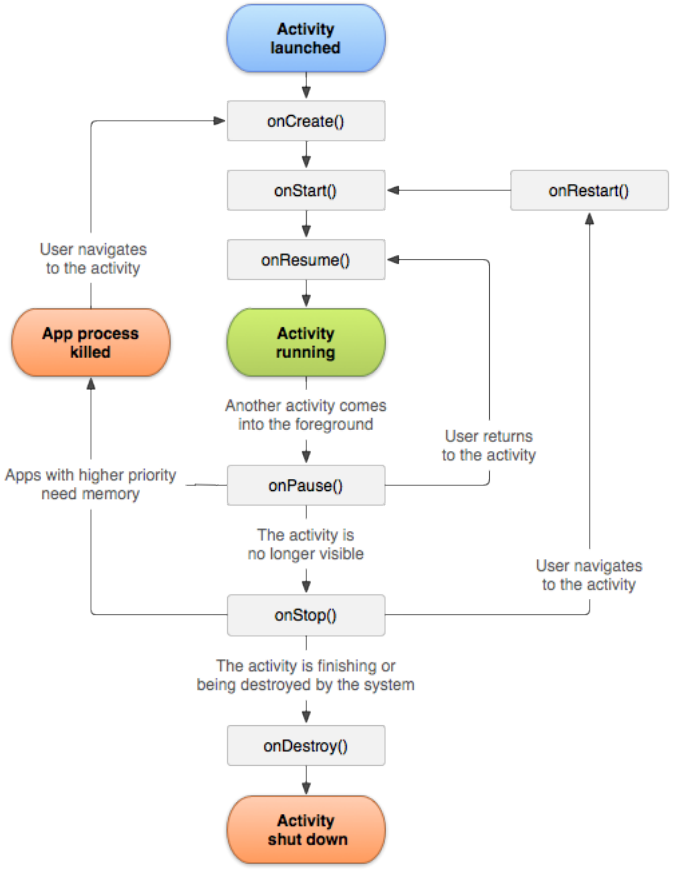


FIGURE 4.1: Activity lifecycle of an Android Activity [1]

instance, an incoming call changes the currently *running* activity to the *shut down* state and tapping a widget may modify the state of data of the currently *running* activity or even close it to open a new one.

Pattern As stated in section 2.1.1, a pattern is a recurring solution for a recurring problem in a certain context. A behavioural pattern reproduces a recurring behaviour situation, for instance, login or master/detail.

Examples of behavioural patterns in Android applications are the contextual options menu (which appears after long pressing an item or pressing the menu button), the Action bar, which enables access to the navigation on the application, or the starting of a new activity.

In 2009, Erik Nilsson [131] identified some recurring problems when developing an Android application and the UI design patterns that could help solve them. If these patterns

have an associated behaviour then it is possible to identify the pattern by the automatic detection of the behaviour.

In 2013, Sahami Shirazi *et al.* studied the layout of Android applications trying, among other goals, to verify if these layouts presented any patterns. They concluded that 75.8% of unique combinations of elements appeared only once in the application. Nevertheless, this study was conducted taking into consideration a static analysis of the layout and its elements while different combination of elements may represent the same behaviour and, thus, the same pattern.

## 4.2 Approach

The approach here presented has two main goals: 1) to extract a behavioural model of a mobile application through the application's automatic exploration and 2) identify and test behavioural patterns during the exploration.

This main steps of this approach consist in:

- defining a catalogue of mobile GUI patterns to identify and the corresponding test strategy;
- applying a hybrid reverse engineering approach to automatically explore mobile applications;
- identifying patterns on the fly and applying the predefined test;
- storing the information regarding all the explored behaviour;
- producing a behavioural model and a test report.

The reverse engineering process to be applied is based on Yang *et al.*'s approach [34], *i.e.*, static analysis will be used to identify which event handlers are associated with each widget in order to dynamically exercise them. A deep study on which frameworks or tools are best is still being undertaken. A possible static analysis tool is WALA [132], which is the one used by Yang *et al.* [34] and extracts a call graph of the application, and possible solutions for the dynamic exploration and exercising of widgets are Robotium [133] and Monkey Runner [134].

Finally, after each step of the exploration in which a pattern (from the previously defined catalogue) is identified, the corresponding test strategy is applied and a report is produced. This follows the same concept of MBT or, more precisely, MBGT.

### 4.2.1 Validation

Any approach attempting at extracting information from a system should have some way of evaluating how good the extraction was. In the presented approach this has two meanings: 1) exploration code coverage, *i.e.*, which percentage of code was explored and 2) test code coverage, *i.e.*, which percentage of the source code was tested.

Moreover, the intention is also to compare the results obtained with other approaches, such as the ones of Yang *et al.* [34] and Amalfitano *et al.* [20]. This comparison will mainly be on the reverse engineering results.

In order to do so, a case study will be conducted preferably on applications also used to validate other mobile reverse engineering approaches. Thus, the implementation of the approach will focus Android applications.

## 4.3 Research Hypothesis/Thesis Statement

Mobile applications have generic recurrent behaviour, independent of their specific domain, that may be tested automatically by combining reverse engineering with testing within an iterative process.

## 4.4 Research Methodology

Part of the research methodology is common to (almost) every other research work. The steps corresponding to this are as follows:

- Constant search for new approaches. This search can be conducted on top venues, like the research done for the state of the art on software reverse engineering or considering all venues;

- Analysis of the works, classifying them according to the ontology defined in Section (ontology) and identifying their main contributions;
- Perform a case study in order to study the feasibility of the approach and to compare the results with the ones obtained from other approaches;
- Validate the results (as explained in Section 4.2;
- Identification of the main contributions, limitations and future work.

Part of the methodology is specific to the work at hand:

- Formalisation of the concepts involved, such as the patterns or the model;
- Identification of the advantages and disadvantages of Android exploration tools and frameworks;
- Identification of the advantages and disadvantages of Android static analysis tools and frameworks;
- Research of Behaviour Patterns, defining the corresponding test strategy;
- Definition of the test patterns in an adequate language.

## 4.5 Work Plan

This section describes the plan to be followed during this PhD, which is depicted in Figure 4.2.

Initially, the focus will be on producing a catalogue of patterns, specially on the ones specific to mobile applications. This will take about five months. Simultaneously, the implementation of the whole process for one of the patterns will take place, for about six months. This includes the exploration part of the reverse engineering process. At the end, it will be possible to collect some results and to study the feasibility of the approach. In the following eight months, the identification of the other patterns and the application of the corresponding test strategy will be implemented.

In the end of this process it will be possible to do an assessment of the approach that will take about six months and that will be based on code coverage techniques and on

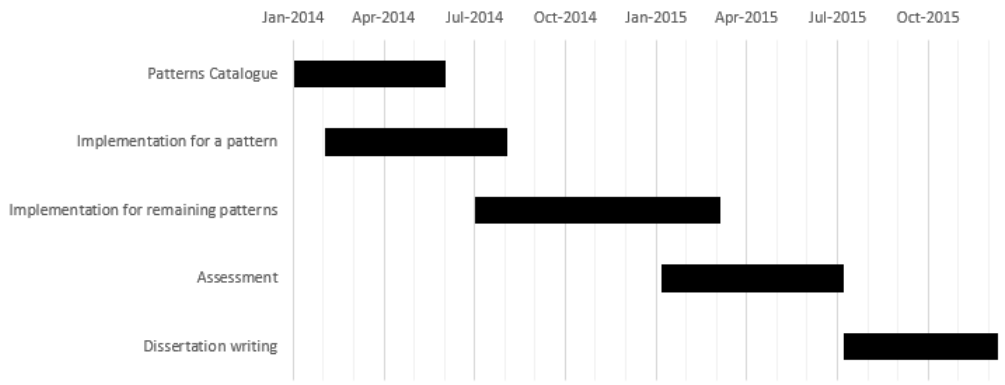


FIGURE 4.2: Gantt chart of the work plan

the comparison of the results with the ones obtained in other approaches. Finally, the last seven months are reserved for the writing of the dissertation.

It is important to state that all these steps will be accompanied by a publishable paper.

# Chapter 5

# Conclusions

This document presents the current state of the art on reverse engineering and, more specifically, on mobile reverse engineering. Furthermore, it describes the approach to be followed in this PhD work as well as the research methodology associated with it.

The final work intends to provide a behavioural model of the application by dynamically exploring and partially testing the application. the testing process will be based on the identification of behavioural patterns on its GUI.

In order to achieve this, a reverse engineering approach will be followed. The process will be based on an automatic and dynamic exploration of the application's GUI. However, in order to ease this exploration, a static analysis will identify the widgets that can be exercised and how they can be exercised, *i.e.*, it identifies the widgets which have event handlers associated. Furthermore, behavioural patterns are to be identified in order to enable the testing of the corresponding behaviour on the fly.

In summary, the contributions of the work will be: 1) a reverse engineering approach to extract a behavioural model of a mobile application, 2) the testing of part of the application according to behavioural patterns found on its GUI.

# Appendix A

# Final Paper Selection on Reverse Engineering

This Appendix presents the distribution of the selected papers amongst the different venues.

Tables A.1 and A.1 present the distribution of the analysed papers in software reverse engineering and in mobile reverse engineering, respectively.

TABLE A.1: Final papers selection on software RE and the venue where they were published

| **Venue** | **Selected papers** |
|---|---|
| European Conference on Software Maintenance and Reengineering (CSMR) | [36, 63, 64, 68, 69, 71, 73, 74, 76, 79, 80, 82, 90, 92, 96, 106, 113, 123, 124] |
| International Conference on Automated Software Engineering (ASE) | [32, 88, 108] |
| International Conference on Program Comprehension (IPCP) | [51, 60, 61, 67, 81, 112, 114, 115, 120] |
| International Conference on Software Engineering (ICSE) | [70, 77, 83, 100, 103] |
| International Conference on Software Maintenance (ICSM) | [59, 65, 66, 75, 85, 87, 99, 104, 150] |
| International Symposium on Software Testing and Analysis (ISSTA) | [91, 93, 94, 101] |
| Joint Meeting on Foundations of Software Engineering (ESEC/FSE) | [17, 72, 102, 119, 121] |
| Working Conference on Reverse Engineering (WCRE) | [33, 62, 84, 89, 105, 109, 110, 116–118] |
| IEEE Transactions on Software Engineering | [52, 86, 95, 122] |
| Journal of Software Maintenance and Evolution: Research and Practice | [48, 111] |
| Journal of Systems and Software | [97, 98, 107] |
| Software Quality Journal | [78] |

TABLE A.2: Final papers selection on mobile RE and the venue where they were published

| **Venue** | **Selected papers** |
|---|---|
| International Conference on Software Testing, Verification, and Validation (ICSTW) | [20, 125] |
| International Conference on Software Maintenance (ICSM) | [99] |
| IEEE/ACM International Conference on Automated Software Engineering (ASE) | [32] |
| International Conference on Malicious and Unwanted Software (MALWARE) | [126] |
| Working Conference on Reverse Engineering (WCRE) | [33, 89] |
| International Symposium on Software Testing and Analysis (ISSTA) | [101] |
| International Conference on Fundamental Approaches to Software Engineering (FASE) | [34] |
| Joint Meeting on Foundations of Software Engineering (ESEC/FSE) | [102, 127] |
| International Conference on Testing Software and Systems (ICTSS) | [31] |

# Appendix B

# Geographical Distribution of the Selected Papers on Reverse Engineering

This Appendix presents the geographical distribution of the selected papers.

Tables B.1 and B.2 present the geographical distribution of the analysed papers in software reverse engineering and in mobile reverse engineering, respectively.

TABLE B.1: Geographical distribution of the research on software RE

| **Country** | **Institution** | **References** |
|---|---|---|
| Belgium | Ghent University, INTEC | [107] |
| | Vrije Universiteit Brussel, PROG | [107] |
| | University of Antwerp, LORE | [107] |
| | Faculty of Computer Science, University of Namur | [72] |
| Brazil | Federal University of São Carlos | [115] |
| | University of Salvador | [115] |
| | Computer Science Department, Federal University of Uberlândia | [62] |
| Canada | Department de Génie Informatique et Génie Locgical, École Polytechnique de Montréal | [48, 59, 60, 64] |
| | Yann-Gaël Guéhéneuc Département d'Informatique et Recherché Opérationnelle, Université de Montréal | [60] |
| | Department of Computer Science, University of Saskatchewan, SK | [104] |
| | Department of Electrical and Computer Engineering, Concordia University, Quebec | [36, 61] |
| | Department of Computer Science and Software Engineering, Concordia University, Quebec | [36, 61, 68] |
| | MCMaster University, Hamilton, ON | [66] |
| | School of Computing, Queen's University, Kingston, Ontario | [92, 111] |
| | Department of Computer Science, University of Alberta, Edmonton, AB | [68, 124] |
| | University of Waterloo | [70] |
| | Enterprise Performance Engineering, Research in Motion, Waterloo, Ontario | [111] |
| | University of Toronto | [86] |
| | University of Victoria | [78] |
| | University of British Columbia | [33] |
| China | School of Software, Shanghai Jiao Tong University | [121] |
| | Department of Computer Science & Engineering, Shanghai Jiao Tong University | [121] |
| Czech Republic | Distributed Systems Research Group, Charles University in Prague | [80] |
| Denmark | IT University of Copenhagen | [70] |
| | Aarhus University | [98, 101, 119] |
| France | LIP6, University of Pierre & Marie Curie, Paris | [71] |
| | University of Rennes 1 | [72] |
| | Triskell team, InriaIrisa | [72] |
| Germany | Embbedded Software Laboratory, Aachen | [89] |
| | Software Engineering, FZI Research Center, Karlsruhe | [80] |
| | IVU Traffic Technologies AG, Aachen | [89] |
| | Professional Services Organization, Netviewer AG | [79] |
| | Institute for Program Structures and Data Organisation, Universität of Karlsruhue | [79] |
| | University of Leipzig | [70] |

| | | |
|---|---|---|
| Germany | Software Engineering Group, Department of Computer Science, Heinz Nixford Institute University of Paderborn | [77] |
| | Software Design and Quality, Karlsruhe Institute of Technology | [80, 122] |
| | SAP AG | [17] |
| | Saarland University Saarbrücken | [17] |
| | Hasso Plattner Institute, University of Potsdam | [120] |
| Hungary | Research Group on Artificial Intelligence, Department of Software Engineering, University of Szeged | [109, 123] |
| | Szeged Software Zrt. | [109] |
| India | Tata Research Development and Design Centre, TCS innovation Labs - Software R & D, Pune | [105] |
| | VMware, Bangalore | [100] |
| Italy | Departimento di Informatica e Sistemistica, Università di Napoli Federico II | [32, 84, 85, 99] |
| | Consorzio Interuniversitario Nazionale per l'Informatica, Napoli | [84] |
| | Department of Engineering, University of Sannio | [59, 64, 75] |
| | Research Center on Software Technology | [96] |
| | Dipartimento di Matematica e Informatica, Università de Salermo, Fisciano | [73, 76] |
| | Fondazione Bruno Kessler - IRST, Trento | [82, 106] |
| | University of Milan Bicocca, Milan | [52] |
| Japan | NTT Software Innovation Center, Midori-cho, Tokyo | [67, 69] |
| | Tokyo Institute of Technology, Ookayama, Tokyo | [67, 69] |
| | Osaka University | [114] |
| | National Institute of Informatics, The University of Tokyo | [90] |
| | Waseda University, Tokyo | [90] |
| | Graduate School of Information Science, Nagoya University, Nagoya, Aichi | [118] |
| Romania | LOOSE Research Group, "Politehnica" University of Timisoara, Romania | [63, 65, 74] |
| Singapore | School of Electrical and Electronic Engineering, Nanyang Technological University | [97] |
| Spain | Indra Systems, Madrid | [71] |
| | University of Murcia | [108] |
| Sweden | Mälardalen University | [78] |
| Switzerland | Software Engineering, ETH Zurich | [110] |
| | Software Composition Group, University of Bern | [51] |
| The Netherlands | Delft University of Technology, SERG | [107, 117] |
| | Solid Source BV, Eidhoven | [113] |
| | Institute for Mathematics and Computer Science, University of Groningen | [113] |
| | Johann Bernoulli Institute, University of Groningen | [120] |

| | | |
|---|---|---|
| UK | Distributed Software Engineering Section, Department of Computing, Imperial College of London | [86] |
| | Department of Computer Science, University of York | [110] |
| USA | IBM Research | [94, 98] |
| | IBM Software Group | [94] |
| | Information Technology Laboratory, National Institute of Standards and Technology, Gaithersberg, MD | [87] |
| | Department of Information Systems, University of Maryland, Baltimore County | [87] |
| | Department of Computer Science, Columbia University, NY | [60] |
| | Department of Computer Science, Drexel University, Philadelphia, PA | [112] |
| | Department of Mathematics, George Mason University, Fairfax, VA | [87] |
| | DoCoMo USA Labs | [91] |
| | Department of Computer Science, University of Maryland | [32] |
| | Department of Computer Science and Engineering, University of Texas at Arlington | [87] |
| | Department of Computer Science, Rensselaer Polytechnic Institute, NY | [116] |
| | Department of Computer Science, University of Kentuchy | [59] |
| | University of California, Irvine | [81] |
| | Electrical Engineering and Computer Science Department, University of California, Berkeley | [83, 93] |
| | University of California, Davis | [91, 95] |
| | School of Computing, DePaul University | [72] |
| | Fujitsu Laboratories of America | [101] |
| | Georgia Institute of Technology | [102] |
| | Microsoft Research | [119] |
| | Microsoft Corporation | [119] |
| | Electrical and Computer Engineering Department, Iowa State University | [103] |
| | Department of Computer Science, Cornell University | [121] |

TABLE B.2: Geographical distribution of the research on mobile RE

| **Country** | **Institution** | **References** |
|---|---|---|
| Canada | University of British Columbia | [33] |
| Denmark | Aarhus University | [101] |
| Germany | Technische Universität Berlin - DAI-Labor | [126] |
| | Embedded Software Laboratory, Aachen | [89] |
| | IVU Traffic Technologies AG, Aachen | [89] |
| Italy | Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II, Napoli | [20, 32, 99, 125] |
| USA | Department of Computer Science, University of Maryland | [32] |
| | Fujitsu Laboratories of America, Sunnyvale, CA | [34, 101] |
| | Department of Computer Science, North Carolina State University | [34] |
| | Department of Computer Science and Engineering, University of California, CA | [31] |
| | Georgia Institute of Technology | [102, 127] |
| | University of Oxford | [127] |

# Bibliography


[1] Google, Activity (Jan. 2014).
URL http://developer.android.com/reference/android/app/Activity.html

[2] CrunchBase, iPhone (Jan. 2014).
URL http://www.crunchbase.com/product/iphone

[3] M. Wilson, T-Mobile G1: Full Details of the HTC Dream Android Phone (Jan. 2014).
URL http://goo.gl/6vqI4E

[4] J. Rivera, R. van der Meulen, Gartner Says Annual Smartphone Sales Surpassed Sales of Feature Phones for the First Time in 2013 (Feb. 2013).
URL http://www.gartner.com/newsroom/id/2665715

[5] V. H., Android's Google Play beats App Store with over 1 million apps, now officially largest (Jul. 2013).
URL http://goo.gl/8y3KPw

[6] N. Ingraham, Apple announces 1 million apps in the App Store, more than 1 billion songs played on iTunes radio (Dec. 2013).
URL http://goo.gl/z3RprB

[7] G. Android, UI/Application Exerciser Monkey (Jan. 2014).
URL http://developer.android.com/tools/help/monkey.html

[8] M. Utting, B. Legeard, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.
URL http://dl.acm.org/citation.cfm?id=1200168

[9] A. C. R. Paiva, J. a. C. P. Faria, R. F. A. M. Vidal, Specification-based testing of user interfaces, in: J. A. Jorge, N. J. Nunes, J. a. F. a. e. Cunha (Eds.), 10th International Workshop on Interactive Systems. Design, Specification, and Verification (DSV-IS '03), Funchal, Portugal, 2003, pp. 139—-153.

[10] A. Kervinen, M. Maunumaa, T. Pääkkönen, M. Katara, W. Grieskamp, C. Weise, Model-Based Testing Through a GUI, in: W. Grieskamp, C. Weise (Eds.), 5th International Workshop on Formal Approaches to Testing of Software (FATES 2005), Vol. 3997 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 16–31. doi:10.1007/11759744.
URL http://www.springerlink.com/content/a0r163864634k45j/

[11] A. M. Memon, An event-flow model of GUI-based applications for testing, in: Software Testing, Verification and Reliability, Vol. 17, 2007, pp. 137–157. doi:10.1002/stvr.364.
URL http://doi.wiley.com/10.1002/stvr.364

[12] A. C. R. Paiva, J. a. C. P. Faria, P. M. C. Mendes, Reverse engineered formal models for GUI testing, in: The 12th international conference on Formal methods for industrial critical systems, Springer-Verlag, Berlin, Germany, 2007, pp. 218–233. doi:10.1007/978-3-540-79707-4_16.
URL http://dl.acm.org/citation.cfm?id=1793603.1793621

[13] T. Pajunen, T. Takala, M. Katara, Model-Based Testing with a General Purpose Keyword-Driven Test Automation Framework, in: Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, IEEE, 2011, pp. 242–251. doi:10.1109/ICSTW.2011.39.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5954415

[14] I. Coimbra Morgado, A. C. R. Paiva, J. a. Pascoal Faria, Dynamic Reverse Engineering of Graphical User Interfaces, International Journal On Advances in Software 5 (3 and 4) (2012) 224–236.
URL http://www.thinkmind.org/index.php?view=article&articleid=soft_v5_n34_2012_7

[15] A. Mesbah, E. Bozdag, A. van Deursen, Crawling AJAX by Inferring User Interface State Changes, in: 2008 Eighth International Conference on Web Engineering,

IEEE, 2008, pp. 122–134. doi:10.1109/ICWE.2008.24.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4577876

[16] M. Nabuco, A. C. Paiva, R. Camacho, J. P. Faria, Inferring UI patterns with Inductive Logic Programming, in: 8th Iberian Conference on Information Systems and Technologies (CISTI '13), Lisbon, Portugal, 2013, pp. 1–5.

[17] M. Schur, A. Roth, A. Zeller, Mining behavior models from enterprise web applications, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, ACM Press, New York, New York, USA, 2013, p. 422.
URL http://dl.acm.org/citation.cfm?id=2491411.2491426

[18] S. Methong, Model-based Automated GUI Testing For Android Web Application Frameworks, in: 2nd International Conference on Biotechnology and Environment Management (ICBEM '12), IACSIT Press, Singapore, Phuket, Thailand, 2012, pp. 106–110. doi:10.7763/IPCBEE.2012.V42.20.

[19] S. R. Garzon, D. Hritsevskyy, Model-based generation of scenario-specific event sequences for the simulation of recurrent user behavior within context-aware applications (WIP), in: 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium, Society for Computer Simulation International, 2012, p. 6.
URL http://dl.acm.org/citation.cfm?id=2346616.2346645

[20] D. Amalfitano, A. R. Fasolino, P. Tramontana, 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, IEEE, 2013, pp. 126–133. doi:10.1109/ICSTW.2013.22.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6571621

[21] M. Rekoff, On Reverse Engineering, IEEE Trans. Systems, Man, and Cybernetics (March-April) (1985) 244 – 252.
URL http://www.citeulike.org/group/1374/article/3944848

[22] E. Chikofsky, J. Cross, Reverse Engineering and Design Recovery: a Taxonomy, IEEE Software 7 (1) (1990) 13–17. doi:10.1109/52.43044.
URL http://dx.doi.org/10.1109/52.43044

[23] G. Canfora, M. Di Penta, New Frontiers of Reverse Engineering, in: Future of Software Engineering, Minneapolis, 2007, pp. 326 – 341. doi:10.1109/FOSE.2007.15.

[24] B. P. Lientz, E. B. Swanson, G. E. Tompkins, Characteristics of application software maintenance, Communications of the ACM 21 (6) (1978) 466–471. doi:10.1145/359511.359522.
URL http://dl.acm.org/citation.cfm?id=359511.359522

[25] I. Sommerville, Software Engineering, 5th Edition, Addison-Wesley, 1995.

[26] L. Erlikh, Leveraging legacy system dollars for e-business, IT Professional 2 (3) (2000) 17–23. doi:10.1109/6294.846201.
URL http://dl.acm.org/citation.cfm?id=612986.613032

[27] T. A. Standish, An Essay on Software Reuse, IEEE Transactions on Software Engineering SE-10 (5) (1984) 494–497. doi:10.1109/TSE.1984.5010272.
URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5010272

[28] T. A. Corbi, Program understanding: Challenge for the 1990s, IBM Systems Journal 28 (2) (1989) 294–306. doi:10.1147/sj.282.0294.
URL http://dl.acm.org/citation.cfm?id=97118.97124

[29] H. A. Muller, J. H. Jahnke, D. B. Smith, M.-A. Storey, Reverse engineering: a roadmap, in: Proceedings of the conference on The future of Software engineering - ICSE '00, ACM Press, New York, New York, USA, 2000, pp. 47–60. doi:10.1145/336512.336526.
URL http://dl.acm.org/citation.cfm?id=336512.336526

[30] E. Eilam, Reversing: Secrets of Reverse Engineering, 1st Edition, Wiley, Indianapolia, Indiana, 2005.

[31] C. Hu, I. Neamtiu, Automated GUI Testing on the Android Platform, in: The 22nd International Conference on Testing Software and Systems (ICTSS '10), ACM

Press, Natal, Brazil, 2010, pp. 67–72. doi:10.1145/1982595.1982612.
URL http://dl.acm.org/citation.cfm?id=1982595.1982612

[32] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, A. 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, ACM Press, New York, New York, USA, 2012, pp. 258–261. doi:10.1145/2351676.2351717.
URL http://dl.acm.org/citation.cfm?id=2351676.2351717

[33] M. E. Joorabchi, A. Mesbah, Reverse Engineering iOS Mobile Applications, in: 2012 19th Working Conference on Reverse Engineering, IEEE, 2012, pp. 177–186. doi:10.1109/WCRE.2012.27.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6385113

[34] W. Yang, M. R. Prasad, T. Xie, A Grey-Box Approach for Automated GUI-Model Generation of Mobile Applications, in: 16th International Conference on Fundamental Approaches to Software Engineering (FASE'13), Rome, Italy, 2013, pp. 250–265. doi:10.1007/978-3-642-37057-1_19.
URL http://link.springer.com/chapter/10.1007/978-3-642-37057-1_19

[35] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, R. Koschke, A Systematic Survey of Program Comprehension through Dynamic Analysis, IEEE Transactions on Software Engineering 35 (5) (2009) 684–702. doi:10.1109/TSE.2009.28.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4815280

[36] C. Patel, A. Hamou-Lhadj, J. Rilling, Software Clustering Using Dynamic Analysis and Static Dependencies, in: 2009 13th European Conference on Software Maintenance and Reengineering, IEEE, 2009, pp. 27–36. doi:10.1109/CSMR.2009.62.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4812736

[37] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language: Towns, Buildings, Construction, Oxford University Press, Berkeley, CA, USA, 1977. doi:10.1002/9780470478240.

[38] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
URL http://dl.acm.org/citation.cfm?id=186897

[39] IEEE Standard for System and Software Verification and Validation (May 2012). doi:10.1109/IEEESTD.2012.6204026.
URL http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6204026

[40] G. Engels, B. Opdyke, D. C. Schmidt, F. Weil, Model Driven Engineering Languages and Systems, Vol. 4735 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.
URL http://www.springerlink.com/index/10.1007/978-3-540-75209-7

[41] I. O. for Standardization, ISO/IEC 14764:2006 Software Life Cycle Processes - Maintenance (2006).
URL http://goo.gl/WBNgKt

[42] C. Arthur, Nokia revenues slide 24% but Lumia sales rise offers hope (Sep. 2013).
URL http://goo.gl/82PBlS

[43] D. Binkley, Source Code Analysis: A Road Map, in: Future of Software Engineering (FOSE '07), IEEE, 2007, pp. 104–119. doi:10.1109/FOSE.2007.27.
URL http://dl.acm.org/citation.cfm?id=1253532.1254713

[44] T. Bell, The concept of dynamic analysis, ACM SIGSOFT Software Engineering Notes 24 (6) (1999) 216–234. doi:10.1145/318774.318944.
URL http://dl.acm.org/citation.cfm?id=318774.318944

[45] H. Kagdi, M. L. Collard, J. I. Maletic, A survey and taxonomy of approaches for mining software repositories in the context of software evolution, Journal of Software Maintenance and Evolution: Research and Practice 19 (2) (2007) 77–131. doi:10.1002/smr.344.
URL http://dl.acm.org/citation.cfm?id=1345056.1345057

[46] IBM, UML basics: The sequence diagram (Oct. 2013).
URL http://www.ibm.com/developerworks/rational/library/3101.html

[47] C. J. van Rijsbergen, Information Retrieval, Newton MA, U.S.A., 1979.

[48] J. K.-Y. Ng, Y.-G. Guéhéneuc, G. Antoniol, Identification of behavioural and creational design motifs through dynamic analysis, Journal of Software Maintenance and Evolution: Research and Practice 22 (8) (2010) 597–627. doi:10.1002/smr.421.
URL http://doi.wiley.com/10.1002/smr.421

[49] C. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of Software Engineering, 1st Edition, Prentice-Hall, Inc, Upper Saddle River NJ, USA, 1991.

[50] L. Briand, The Experimental Paradigm in Reverse Engineering: Role, Challenges, and Limitations, in: 2006 13th Working Conference on Reverse Engineering, IEEE, 2006, pp. 3–8. doi:10.1109/WCRE.2006.53.
URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4023971

[51] D. Rothlisberger, O. Greevy, O. Nierstrasz, Exploiting Runtime Information in the IDE, in: 2008 16th IEEE International Conference on Program Comprehension, IEEE, 2008, pp. 63–72. doi:10.1109/ICPC.2008.32.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4556118

[52] L. Mariani, F. Pastore, M. Pezze, Dynamic Analysis for Diagnosing Integration Faults, IEEE Transactions on Software Engineering 37 (4) (2011) 486–508. doi:10.1109/TSE.2010.93.
URL http://goo.gl/0b9a1L

[53] C. Hu, I. Neamtiu, Automating gui testing for android applications, in: The 6th International Workshop on Automation of software test (AST '11), ACM Press, New York, New York, USA, 2011, p. 7. doi:10.1145/1982595.1982612.
URL http://dl.acm.org/citation.cfm?id=1982595.1982612

[54] Microsoft, Introduction to Instrumentation and Tracing (Oct. 2013).
URL http://msdn.microsoft.com/en-us/library/aa983649(VS.71).aspx

[55] A. M. Memon, I. Banerjee, A. Nagarajan, GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing, in: The 10th Working Conference on Reverse Engineering (WCRE '03), 2003.
URL http://www.cs.umd.edu/~atif/papers/MemonWCRE2003.pdf

[56] F. E. Allen, Control flow analysis, ACM SIGPLAN Notices 5 (7) (1970) 1–19. doi:10.1145/390013.808479.
URL http://dl.acm.org/citation.cfm?id=390013.808479

[57] O. M. Group, Business Process Model and Notation (Aug. 2013).
URL http://www.bpmn.org/

[58] F. Brito e Abreu, Empirical Software Engineering: a short course, Personal Communication.

[59] G. Antoniol, J. H. Hayes, Y.-G. Gueheneuc, M. di Penta, Reuse or rewrite: Combining textual, static, and dynamic analyses to assess the cost of keeping a system up-to-date, in: 2008 IEEE International Conference on Software Maintenance, IEEE, 2008, pp. 147–156. doi:10.1109/ICSM.2008.4658063.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4658063

[60] M. Eaddy, A. Aho, G. Antoniol, Y.-G. Gueheneuc, CERBERUS: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis and Program Analysis, in: 2008 16th IEEE International Conference on Program Comprehension, IEEE, 2008, pp. 53–62. doi:10.1109/ICPC.2008.39.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4556117

[61] A. Rohatgi, A. Hamou-Lhadj, J. Rilling, An Approach for Mapping Features to Code Based on Static and Dynamic Analysis, in: 2008 16th IEEE International Conference on Program Comprehension, IEEE, 2008, pp. 236–241. doi:10.1109/ICPC.2008.35.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4556137

[62] V. Sobreira, M. d. A. Maia, A Visual Trace Analysis Tool for Understanding Feature Scattering, in: 2008 15th Working Conference on Reverse Engineering, IEEE, 2008, pp. 337–338. doi:10.1109/WCRE.2008.40.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4656432

[63] D. C. Cosma, R. Marinescu, Understanding the Impact of Distribution in Object-Oriented Distributed Systems Using Structural Program Dependencies, in: 2008 12th European Conference on Software Maintenance and Reengineering, IEEE,

2008, pp. 103–112. doi:10.1109/CSMR.2008.4493305.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4493305

[64] F. Asadi, M. Di Penta, G. Antoniol, Y.-G. Gueheneuc, A Heuristic-Based Approach to Identify Concepts in Execution Traces, in: 2010 14th European Conference on Software Maintenance and Reengineering, IEEE, 2010, pp. 31–40. doi:10.1109/CSMR.2010.17.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5714415

[65] D. C. Cosma, Reverse engineering object-oriented distributed systems, in: 2010 IEEE International Conference on Software Maintenance, IEEE, 2010, pp. 1–6. doi:10.1109/ICSM.2010.5609716.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5609716

[66] A. Yousefi, K. Sartipi, Identifying distributed features in SOA by mining dynamic call trees, in: 2011 27th IEEE International Conference on Software Maintenance (ICSM), IEEE, 2011, pp. 73–82. doi:10.1109/ICSM.2011.6080774.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6080774

[67] H. Kazato, S. Hayashi, S. Okada, S. Miyata, T. Hoshino, M. Saeki, Toward structured location of features, in: 2012 20th IEEE International Conference on Program Comprehension (ICPC), IEEE, 2012, pp. 255–256. doi:10.1109/ICPC.2012.6240497.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6240497

[68] N. Negara, N. Tsantalis, E. Stroulia, Feature Detection in Ajax-Enabled Web Applications, in: 2013 17th European Conference on Software Maintenance and Reengineering, IEEE, 2013, pp. 154–163. doi:10.1109/CSMR.2013.25.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6498464

[69] H. Kazato, S. Hayashi, T. Kobayashi, T. Oshima, S. Okada, S. Miyata, T. Hoshino, M. Saekii, Incremental Feature Location and Identification in Source Code, in: 2013 17th European Conference on Software Maintenance and Reengineering, IEEE, 2013, pp. 371–374. doi:10.1109/CSMR.2013.52.
URL http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6498491

[70] S. She, R. Lotufo, T. Berger, A. Wasowski, K. Czarnecki, Reverse engineering feature models, in: Proceeding of the 33rd international conference on Software engineering - ICSE '11, ACM Press, New York, New York, USA, 2011, p. 461. doi:10.1145/1985793.1985856.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6032485&contentType=Conference+Publications&sortType=asc_p_Sequence&filter=AND(p_IS_Number:6032438)&pageNumber=2

[71] T. Ziadi, L. Frias, M. A. A. da Silva, M. Ziane, Feature Identification from the Source Code of Product Variants, in: 2012 16th European Conference on Software Maintenance and Reengineering, IEEE, 2012, pp. 417–422. doi:10.1109/CSMR.2012.52.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6178889

[72] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, P. Heymans, Feature model extraction from large collections of informal product descriptions, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, ACM Press, New York, New York, USA, 2013, pp. 290–300. doi:10.1145/2491411.2491455.
URL http://dl.acm.org/citation.cfm?id=2491411.2491455

[73] A. De Lucia, V. Deufemia, C. Gravino, M. Risi, Behavioral Pattern Identification through Visual Language Parsing and Code Instrumentation, in: 2009 13th European Conference on Software Maintenance and Reengineering, IEEE, 2009, pp. 99–108. doi:10.1109/CSMR.2009.29.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4812743

[74] P. F. Mihancea, R. Marinescu, Discovering Comprehension Pitfalls in Class Hierarchies, in: 2009 13th European Conference on Software Maintenance and Reengineering, IEEE, 2009, pp. 7–16. doi:10.1109/CSMR.2009.31.
URL http://ieeexplore.ieee.org/xpl/login.jsp?arnumber=4812734

[75] M. L. Bernardi, G. A. Di Lucca, Model-driven detection of Design Patterns, in: 2010 IEEE International Conference on Software Maintenance, IEEE, 2010, pp. 1–5. doi:10.1109/ICSM.2010.5609740.

URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5609740

[76] A. De Lucia, V. Deufemia, C. Gravino, M. Risi, Improving Behavioral Design Pattern Detection through Model Checking, in: 2010 14th European Conference on Software Maintenance and Reengineering, IEEE, 2010, pp. 176–185. doi:10.1109/CSMR.2010.16.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5714432

[77] M. von Detten, M. Meyer, D. Travkin, Reverse engineering with the reclipse tool suite, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10, Vol. 2, ACM Press, New York, New York, USA, 2010, p. 299. doi:10.1145/1810295.1810360.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6062184&contentType=Conference+Publications&sortType=asc_p_Sequence&filter=AND(p_IS_Number:6062119)&pageNumber=3

[78] H. M. Kienle, J. Kraft, T. Nolte, System-specific static code analyses: a case study in the complex embedded systems domain, Software Quality Journal 20 (2) (2011) 337–367. doi:10.1007/s11219-011-9138-7.
URL http://www.springerlink.com/index/10.1007/s11219-011-9138-7

[79] L. Chouambe, B. Klatt, K. Krogmann, Reverse Engineering Software-Models of Component-Based Systems, in: 2008 12th European Conference on Software Maintenance and Reengineering, IEEE, 2008, pp. 93–102. doi:10.1109/CSMR.2008.4493304.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4493304

[80] S. Becker, M. Hauck, M. Trifu, K. Krogmann, J. Kofron, Reverse Engineering Component Models for Quality Predictions, in: 2010 14th European Conference on Software Maintenance and Reengineering, IEEE, 2010, pp. 194–197. doi:10.1109/CSMR.2010.34.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5714435

[81] H. Sajnani, Automatic software architecture recovery: A machine learning approach, in: 2012 20th IEEE International Conference on Program Comprehension

(ICPC), IEEE, 2012, pp. 265–268.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6240501&contentType=Conference+Publications&queryText=icpc+2012

[82] C. Di Francescomarino, A. Marchetto, P. Tonella, Reverse Engineering of Business Processes exposed as Web Applications, in: 2009 13th European Conference on Software Maintenance and Reengineering, IEEE, 2009, pp. 139–148. doi:10.1109/CSMR.2009.26.
URL http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=4812747

[83] A. Rabkin, R. Katz, Static extraction of program configuration options, in: Proceeding of the 33rd international conference on Software engineering - ICSE '11, ACM Press, New York, New York, USA, 2011, p. 131. doi:10.1145/1985793.1985812.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6032452&contentType=Conference+Publications&sortType=asc_p_Sequence&filter=AND(p_IS_Number:6032438)

[84] D. Amalfitano, A. R. Fasolino, P. Tramontana, Reverse Engineering Finite State Machines from Rich Internet Applications, in: The 15th Working Conference on Reverse Engineering (WCRE '08), IEEE, 2008, pp. 69–73. doi:10.1109/WCRE.2008.17.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4656395

[85] D. Amalfitano, A. R. Fasolino, P. Tramontana, Experimenting a reverse engineering technique for modelling the behaviour of rich internet applications, in: 2009 IEEE International Conference on Software Maintenance, IEEE, 2009, pp. 571–574. doi:10.1109/ICSM.2009.5306391.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5306391

[86] S. Uchitel, G. Brunet, M. Chechik, Synthesis of Partial Behavior Models from Properties and Scenarios, IEEE Transactions on Software Engineering 35 (3) (2009) 384–406.
URL http://www.computer.org/csdl/trans/ts/2009/03/tts2009030384-abs.html

[87] W. Wang, Y. Lei, S. Sampath, R. Kacker, R. Kuhn, J. Lawrence, A combinatorial approach to building navigation graphs for dynamic web applications, in: 2009 IEEE International Conference on Software Maintenance, IEEE, 2009, pp. 211–220. doi:10.1109/ICSM.2009.5306321.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5306321

[88] H. Bruneliere, J. Cabot, F. Jouault, F. Madiot, MoDisco: a generic and extensible framework for model driven reverse engineering, in: Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10, ACM Press, New York, New York, USA, 2010, pp. 173–174. doi:10.1145/1858996.1859032.
URL http://dl.acm.org/citation.cfm?id=1858996.1859032

[89] D. Franke, C. Elsemann, S. Kowalewski, C. Weise, Reverse Engineering of Mobile Application Lifecycles, in: 18th Working Conference on Reverse Engineering (WCRE '11), IEEE, 2011, pp. 283–292. doi:10.1109/WCRE.2011.42.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6079853

[90] Y. Maezawa, H. Washizaki, S. Honiden, Extracting Interaction-Based Stateful Behavior in Rich Internet Applications, in: 2012 16th European Conference on Software Maintenance and Reengineering, IEEE, 2012, pp. 423–428. doi:10.1109/CSMR.2012.53.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6178915

[91] G. Wassermann, D. Yu, A. Chander, D. Dhurjati, H. Inamura, Z. Su, Dynamic test input generation for web applications, in: Proceedings of the 2008 international symposium on Software testing and analysis - ISSTA '08, ACM Press, New York, New York, USA, 2008, pp. 249–259. doi:10.1145/1390630.1390661.
URL http://dl.acm.org/citation.cfm?id=1390661

[92] M. H. Alalfi, J. R. Cordy, T. R. Dean, Automating Coverage Metrics for Dynamic Web Applications, in: 2010 14th European Conference on Software Maintenance and Reengineering, IEEE, 2010, pp. 51–60.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=

5714417&contentType=Conference+Publications&sortType=asc_p_Sequence&filter=AND(p_IS_Number:5714411)

[93] D. Babić, L. Martignoni, S. McCamant, D. Song, Statically-directed dynamic automated test generation, in: Proceedings of the 2011 International Symposium on Software Testing and Analysis - ISSTA '11, ACM Press, New York, New York, USA, 2011, pp. 12–22. doi:10.1145/2001420.2001423.
URL http://dl.acm.org/citation.cfm?id=2001420.2001423

[94] S. Guarnieri, M. Pistoia, O. Tripp, J. Dolby, S. Teilhet, R. Berg, Saving the world wide web from vulnerable JavaScript, in: Proceedings of the 2011 International Symposium on Software Testing and Analysis - ISSTA '11, ACM Press, New York, New York, USA, 2011, pp. 177–187. doi:10.1145/2001420.2001442.
URL http://dl.acm.org/citation.cfm?id=2001420.2001442

[95] T. Kwon, Z. Su, Automatic Detection of Unsafe Dynamic Component Loadings, IEEE Transactions on Software Engineering 38 (2) (2012) 293–313. doi:10.1109/TSE.2011.108.
URL http://goo.gl/PJmNOj

[96] M. L. Bernardi, Reverse Engineering of Aspect Oriented Systems to Support their Comprehension, Evolution, Testing and Assessment, in: 2008 12th European Conference on Software Maintenance and Reengineering, IEEE, 2008, pp. 290–293. doi:10.1109/CSMR.2008.4493329.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=4493329

[97] H. Liu, H. B. Kuan Tan, Testing input validation in Web applications through automated model recovery, Journal of Systems and Software 81 (2) (2008) 222–233. doi:10.1016/j.jss.2007.05.007.
URL http://dx.doi.org/10.1016/j.jss.2007.05.007

[98] S. Artzi, J. Dolby, S. H. Jensen, A. Mø ller, F. Tip, A framework for automated testing of javascript web applications, in: Proceeding of the 33rd international conference on Software engineering - ICSE '11, ACM Press, New York, New York, USA, 2011, p. 571.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=

6032496&contentType=Conference+Publications&sortType=asc_p_Sequence&filter=AND(p_IS_Number:6032438)&pageNumber=3

[99] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, G. Imparato, A toolset for GUI testing of Android applications, in: 2012 28th IEEE International Conference on Software Maintenance (ICSM), IEEE, 2012, pp. 650–653. doi:10.1109/ICSM.2012.6405345.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6405345

[100] V. Balachandran, Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 931–940. doi:10.1109/ICSE.2013.6606642.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6606642

[101] C. S. Jensen, M. R. Prasad, A. Mø ller, Automated testing with targeted event sequence generation, in: Proceedings of the 2013 International Symposium on Software Testing and Analysis - ISSTA 2013, ACM Press, New York, New York, USA, 2013, pp. 67–77.
URL http://dl.acm.org/citation.cfm?id=2483760.2483777

[102] A. Machiry, R. Tahiliani, M. 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, ACM Press, New York, New York, USA, 2013, p. 224. doi:10.1145/2491411.2491450.
URL http://dl.acm.org/citation.cfm?id=2491411.2491450

[103] H. V. Nguyen, H. A. Nguyen, T. T. Nguyen, T. N. Nguyen, DRC: A detection tool for dangling references in PHP-based web applications, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 1299–1302. doi:10.1109/ICSE.2013.6606702.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6606702

[104] A. Sutherland, K. Schneider, UI traces: Supporting the maintenance of interactive software, in: 2009 IEEE International Conference on Software Maintenance, IEEE, 2009, pp. 563–566. doi:10.1109/ICSM.2009.5306389.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5306389

[105] A. Jana, R. Naik, Precise Detection of Uninitialized Variables Using Dynamic Analysis - Extending to Aggregate and Vector Types, in: 2012 19th Working Conference on Reverse Engineering, IEEE, 2012, pp. 197–201. doi:10.1109/WCRE.2012.29.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6385115

[106] M. Ceccato, Automatic Support for the Migration Towards Aspects, in: 2008 12th European Conference on Software Maintenance and Reengineering, IEEE, 2008, pp. 298–301. doi:10.1109/CSMR.2008.4493331.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4493331

[107] B. Adams, K. De Schutter, A. Zaidman, S. Demeyer, H. Tromp, W. De Meuter, Using aspect orientation in legacy environments for reverse engineering using dynamic analysis—An industrial experience report, Journal of Systems and Software 82 (4) (2009) 668–684. doi:10.1016/j.jss.2008.09.031.
URL http://dx.doi.org/10.1016/j.jss.2008.09.031

[108] O. Sánchez Ramón, J. Sánchez Cuadrado, J. García Molina, Model-driven reverse engineering of legacy graphical user interfaces, in: Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10, ACM Press, New York, New York, USA, 2010, pp. 147–150. doi:10.1145/1858996.1859023.
URL http://dl.acm.org/citation.cfm?id=1858996.1859023

[109] C. Nagy, L. Vidacs, R. Ferenc, T. Gyimothy, F. Kocsis, I. Kovacs, Solutions for Reverse Engineering 4GL Applications, Recovering the Design of a Logistical Wholesale System, in: 2011 15th European Conference on Software Maintenance and Reengineering, IEEE, 2011, pp. 343–346.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5741335

[110] M. Trudel, C. A. Furia, M. Nordio, B. Meyer, M. Oriol, C to O-O Translation: Beyond the Easy Stuff, in: 2012 19th Working Conference on Reverse Engineering,

IEEE, 2012, pp. 19–28. doi:10.1109/WCRE.2012.12.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6385098

[111] Z. M. Jiang, A. E. Hassan, G. Hamann, P. Flora, An automated approach for abstracting execution logs to execution events, Journal of Software Maintenance and Evolution: Research and Practice 20 (4) (2008) 249–267. doi:10.1002/smr.374.
URL http://doi.wiley.com/10.1002/smr.374

[112] W. Mongan, M. Shevertalov, S. Mancoridis, Re-Engineering a Reverse Engineering Portal to a Distributed SOA, in: 2008 16th IEEE International Conference on Program Comprehension, IEEE, 2008, pp. 218–223. doi:10.1109/ICPC.2008.17.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4556134

[113] A. Telea, L. Voinea, SOLIDFX: An Integrated Reverse Engineering Environment for C++, in: 2008 12th European Conference on Software Maintenance and Reengineering, IEEE, 2008, pp. 320–322. doi:10.1109/CSMR.2008.4493339.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4493339

[114] S. Munakata, T. Ishio, K. Inoue, OGAN: Visualizing object interaction scenarios based on dynamic interaction context, in: 2009 IEEE 17th International Conference on Program Comprehension, IEEE, 2009, pp. 283–284. doi:10.1109/ICPC.2009.5090059.
URL http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5090059

[115] D. Porto, M. Mendonca, S. Fabbri, CRISTA: A tool to support code comprehension based on visualization and reading technique, in: 2009 IEEE 17th International Conference on Program Comprehension, IEEE, 2009, pp. 285–286. doi:10.1109/ICPC.2009.5090060.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5090060

[116] A. Milanova, Y. Liu, Static Analysis for Understanding Shared Objects in Open Concurrent Java Programs, in: 2010 17th Working Conference on Reverse Engineering, IEEE, 2010, pp. 45–54. doi:10.1109/WCRE.2010.14.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5645483

[117] T. Espinha, A. Zaidman, H.-G. Gross, Understanding the Runtime Topology of Service-Oriented Systems, in: 2012 19th Working Conference on Reverse Engineering, IEEE, 2012, pp. 187–196. doi:10.1109/WCRE.2012.28.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6385114

[118] K. Noda, T. Kobayashi, K. Agusa, Execution Trace Abstraction Based on Meta Patterns Usage, in: 2012 19th Working Conference on Reverse Engineering, IEEE, 2012, pp. 167–176. doi:10.1109/WCRE.2012.26.
URL http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6385112

[119] M. Madsen, B. Livshits, M. Fanning, Practical static analysis of JavaScript applications in the presence of frameworks and libraries, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, ACM Press, New York, New York, USA, 2013, p. 499.
URL http://dl.acm.org/citation.cfm?id=2491411.2491417

[120] J. Trumper, J. Dollner, A. Telea, Multiscale visual comparison of execution traces, in: 2013 21st International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 53–62. doi:10.1109/ICPC.2013.6613833.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6613833

[121] Q. Wang, J. Zhou, Y. Chen, Y. Zhang, J. Zhao, Extracting URLs from JavaScript via program analysis, in: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013, ACM Press, New York, New York, USA, 2013, p. 627.
URL http://dl.acm.org/citation.cfm?id=2491411.2494583

[122] K. Krogmann, M. Kuperberg, R. Reussner, Using Genetic Search for Reverse Engineering of Parametric Behavior Models for Performance Prediction, IEEE Transactions on Software Engineering 36 (6) (2010) 865–877. doi:10.1109/TSE.2010.69.
URL http://goo.gl/hnR0lr

[123] J. Jász, L. Schrettner, A. Beszédes, C. Osztrogonác, T. Gyimóthy, Impact Analysis Using Static Execute After in WebKit, in: 2012 16th European Conference on Software Maintenance and Reengineering, IEEE, 2012, pp. 95–104.

doi:10.1109/CSMR.2012.20.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6178857

[124] M. Smit, E. Stroulia, K. Wong, Use Case Redocumentation from GUI Event Traces, in: 2008 12th European Conference on Software Maintenance and Reengineering, IEEE, 2008, pp. 263–268. doi:10.1109/CSMR.2008.4493323.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4493323

[125] D. Amalfitano, A. Fasolino, P. Tramontana, A GUI Crawling-Based Technique for Android Mobile Application Testing, in: Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, IEEE, 2011, pp. 252–261. doi:10.1109/ICSTW.2011.77.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5954416

[126] L. Batyuk, M. Herpich, S. A. Camtepe, K. Raddatz, A.-D. Schmidt, S. Albayrak, Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications, in: 2011 6th International Conference on Malicious and Unwanted Software, IEEE, 2011, pp. 66–72. doi:10.1109/MALWARE.2011.6112328.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6112328

[127] S. Anand, M. Naik, M. J. Harrold, H. Yang, Automated concolic testing of smartphone apps, in: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12, ACM Press, New York, New York, USA, 2012, p. 1. doi:10.1145/2393596.2393666.
URL http://dl.acm.org/citation.cfm?id=2393596.2393666

[128] G. Android, Get the Android SDK (2014).
URL http://developer.android.com/sdk/index.html

[129] D. R. Hackner, A. M. Memon, Test case generator for GUITAR, in: Companion of the 13th international conference on Software engineering (ICSE Companion '08), ICSE Companion '08, ACM Press, New York, New York, USA, 2008, p. 959. doi:10.1145/1370175.1370207.

URL http://dl.acm.org/citation.cfm?id=1370175.1370207http://doi.acm.org/10.1145/1370175.1370207

[130] D. Franke, C. Elsemann, S. Kowalewski, Reverse Engineering and Testing Service Life Cycles of Mobile Platforms, in: 2012 23rd International Workshop on Database and Expert Systems Applications, IEEE, 2012, pp. 16–20. doi:10.1109/DEXA.2012.40.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6327397

[131] E. G. Nilsson, Design patterns for user interface for mobile applications, Advances in Engineering Software 40 (12) (2009) 1318–1328. doi:10.1016/j.advengsoft.2009.01.017.
URL http://www.sciencedirect.com/science/article/pii/S0965997809000428

[132] T. J. Watsoon, Wala (Jan. 2014).
URL http://wala.sourceforge.net/wiki/index.php/Main_Page

[133] Google, robotium (Jan. 2014).
URL https://code.google.com/p/robotium/

[134] G. Android, Monkey Runner (Jan. 2014).
URL http://developer.android.com/tools/help/monkeyrunner_concepts.html

[135] I. Coimbra Morgado, A. C. R. Paiva, J. a. Pascoal Faria, Reverse Engineering of Graphical User Interfaces, in: The Sixth International Conference on Software Engineering Advances (ICSEA '11), no. c, Barcelona, 2011, pp. 293–298.

[136] U. Brandes, M. Eiglsperger, J. Lerner, GraphML Primer (2007).

[137] M. Barnett, K. R. M. Leino, W. Schulte, The Spec\# Programming System: An Overview, in: International Conference in Construction and Analysis of Safe, Secure and Interoperable Smart Devices (CASSIS '04), Springer, Marseille, France, 2004, pp. 49–69. doi:10.1.1.11.2133.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.2133

[138] K. L. McMillan, Getting Started with SMV, Cadence Berkley Labs, 2001 Addison St., Berkley, CA, USA, 1999.
URL http://www.cs.indiana.edu/classes/p415/readings/smv/McMillan-tutorial.pdf

[139] E. Clarke, K. McMillan, S. Campos, V. Hartonas-Garmhausen, Symbolic model checking, in: R. Alur, T. Henzinger (Eds.), Computer Aided Verification, Vol. 1102 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1996, pp. 419–422.
URL http://dx.doi.org/10.1007/3-540-61474-5_93

[140] F. Paternò, C. Santoro, Integrating Model Checking and HCI Tools to Help Designers Verify User Interface Properties, in: 7th International Workshop on Interactive Systems Design, Specification and Verification, Limmerick, Ireland, 2001.
URL http://www.springerlink.com/content/jvv88l4wfp912ja8/

[141] N. Kamel, S. A. Selouani, H. Hamam, A Model-Checking Approach for the Verification of CARE Usability Properties for Multimodal User Interfaces, International Review on Computers & Software 4 (1) (2009) 152—-160.
URL http://goo.gl/fk8v88

[142] A. Grilo, A. Paiva, J. Faria, Reverse engineering of GUI models for testing, in: The 5th Iberian Conference on Information Systems and Technologies (CISTI '10), no. July, IEEE, 2010, pp. 1–6.
URL http://goo.gl/bXcIy

[143] I. Coimbra Morgado, A. C. R. Paiva, J. Pascoal Faria, R. Camacho, GUI Reverse Engineering with Machine Learning, in: Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE'12), Zurich, Switzerland, 2012, pp. 27–31.

[144] S. Muggleton, Inductive logic programming, in: Proceedings of the 1st Conference on Algorithmic Learning Theory, 1990, pp. 43–62.

[145] S. H. Muggleton, L. D. Raedt, Inductive Logic Programming: Theory and Methods, Journal of Logic Programming 19,20 (1994) 629–679.

[146] W. F. Clocksin, C. S. Mellish, Programming in Prolog, 4th Edition, Springer-Verlag New York Berlin Heidelberg, Berlin, Germany, 2003.

[147] R. M. L. M. Moreira, A. C. R. Paiva, A. Memon, A pattern-based approach for GUI modeling and testing, in: 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2013, pp. 288–297. doi:10.1109/ISSRE.2013.6698881.
URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6698881

[148] T. Monteiro, A. C. Paiva, Pattern Based GUI Testing Modeling Environment, in: 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops, IEEE, 2013, pp. 140–143. doi:10.1109/ICSTW.2013.24.
URL http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=6571623

[149] M. Cunha, A. C. R. Paiva, H. S. Ferreira, R. Abreu, PETTool: A pattern-based GUI testing tool, in: Software Technology and Engineering (ICSTE), 2010 2nd International Conference on, Vol. 1, IEEE, San Juan, PR, 2010, pp. V1–202 – VI–206. doi:10.1109/ICSTE.2010.5608882.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5608882

[150] B. Burgstaller, A. Egyed, Understanding where requirements are implemented, in: 2010 IEEE International Conference on Software Maintenance, IEEE, 2010, pp. 1–5. doi:10.1109/ICSM.2010.5609699.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5609699