ABSTRACT: It has been recognized that mobile platforms continue to prevail in all aspects of human activity. However, mobile applications might suffer from improper behavior just like other software types. Consequently, there is a growing need for testing mobile applications thoroughly. Model-based testing is one of the popular testing approaches, which depends on having a functional model of system specification to generate and evaluate test cases. In most cases, the specification model is not available, suffers from an unsatisfactory quality, or does not represent all business rules and functional requirements. That would lead to increase in the difficulties in testing, modifying and maintaining mobile applications. Reverse engineering can be used to reveal application’s designs, architecture and knowledge and extract models to represent them.

This paper proposes a solution by developing a framework for testing mobile applications and automatically extracting a GUI (Graphical User Interface) model based on reverse engineering techniques. The extracted model is then extended by adding missing functional requirements and business rules and validated according to accuracy and quality measures. The framework is evaluated by testing two Android applications and comparing the testing outcomes with similar frameworks. The evaluation shows promising results in terms of coverage ratio and fault detection.

Subject Categories and Descriptors:
[D.2.7 Software Distribution, Maintenance, and Enhancement]: Reverse engineering, and reengineering [F.1.1 Models of Computation]: Automata; [C.1 Processor Architecture]

Cellular architecture

General Terms: Mobile Applications, Graphical User Interface

Keywords: Reverse Engineering, Model-based Testing, Business Rules, Extended Finite State Machine

Received: 17 October 2018, Revised 2 February 2019, Accepted 12 February 2019

Review Metrics: Review Scale- 0/6, Review Score-4.52, Inter-reviewer Consistency- 78%

DOI: 10.6025/jdim/2019/17/4/189-200

1. Introduction

Mobile platforms are expected to outperform desktop and web applications because of the current shift to mobile devices in many applications such as e-mail, social networking and some e-commerce applications [1]. This trend has led to exponential growth in the number and diversity of mobile applications. For instance, Google Play had more than 600,000 applications installed over 10 billion times in June 2012 [2]. Several mobile applications with similar functionality are now available in the market. Some of them have even critical behavior such as banking apps. As a result, it is essential to have high-quality applications in order to be secure and competitive.

Testing mobile applications tend to be essential and create challenges within software engineering companies. To explain, mobile applications are designed in many
frameworks, operating systems, and programming languages. In addition, mobile applications have many options and settings that affect application properties. Applications should be tested to prove that they are working correctly and expectedly in all settings by generating abstract test cases, executing them and then comparing actual outputs with expected ones.

Mostly, mobile applications are manually tested by extracting and executing selected test cases. However, manual testing is tedious, cumbersome and exhaustive because the testing process must cover a wide range of scenarios, properties and settings.

Automated testing by which test generation and execution are completed automatically is more preferable. One of the common testing approaches that support automatic testing is Model-Based Testing (MBT) [3]. It uses a model that represents the application properties under test, and responsible for generating test cases [8]. However; this model in most cases is might to be not available. Reverse engineering [4], an approach to extract behavioral model, can be seen a solution to the model-absence. Several MBT testing frameworks based on reverse engineering techniques are developed [5]. They automatically analyze the GUI of mobile applications under test; generate a model that represents application properties and generate tests and execute them. However, these approaches suffer from (1) generating incomplete, invalidated and unverifiable models, (2) losing business rules and some functional requirements and (3) generating a very large number of test cases.

This paper addresses these issues by developing a framework for testing Android applications by extracting specification model using reverse engineering techniques. Based on which, the extracted specification model is extended with missing business rules and functional requirements and then validated using measures of accuracy and quality. Furthermore, abstract test cases are automatically generated and executed on two Android applications. The proposed framework is evaluated by comparing the testing results with other similar frameworks.

This paper is structured as follows: Section 2 reviews the state of the art. The proposed framework is explained in Section 3. Two case studies for evaluating the overall framework and results discussion are presented in Section 4. Section 5 concludes the paper with a research summary and future trends.

2. Related Work

Several frameworks for testing mobile applications using MBT techniques were proposed. To start with, in 2016 L.Aceto et al. [10] introduced a methodology for testing mobile applications based on a manual abstraction of Extended Finite State Machine (EFSM) model. This methodology supports web and mobile applications. The drawback as it appeared from the methodologies evaluation is the large cost of manual effort for building the model [9].

Many of model-based testing approaches were based on the principle of reverse engineering. In 2011, Fasolino et al. [11] developed MBT approach based on dynamic analysis the technique to extract Graphical User Interface (GUI) tree model [12] from five mobile applications. Fasolino et al. defined the process of the crawler (Ripper) which extracts the model in a particular format generates and executes tests then record the results. The approach was evaluated by comparing the testing results with similar approaches and achieved better results. However, the approach was limited to the type of extracted model [6].

Tramontana et al. [6] proposed GUITAR methodology based on ripper algorithm where the user Event Flow Graph (EFG) is extracted. Abstract test cases are then built based on the EFG model and executed based on coverage criteria. The approach was implemented by a tool called Mobiguitar and evaluated by comparing its testing results with simple ripper results.

Sebastien Salva et al. [13] developed an automatic MBT approach based on a theme of extracting more than one model from the application using reverse engineering. The extracted model was a Finite State Machine (FSM). Several techniques were developed to facilitate the exploration and acceleration of the extraction of the model.

Wei Yang et al. [1] also developed the ORBIT tool to test mobile applications and manipulate the results. The Hybrid analysis was used to extract the model by combining static analysis [7], for extracting user events, and the dynamic analysis, for executing them on the application under test. WALA tool [14] was used to perform the static analysis. Monkey [15] was used for the dynamic exploration and Robotuim [16] was used for the implementation of the testing process. The approach was evaluated by comparing the test results with similar approaches and achieved better results. However, testing tools were not available to others.

Despite there were a number of approaches that use MBT techniques for testing mobile applications, there were some limitations on the type of extracted models, model validating, missing functional requirements and business rules. Reviewing previous studies on MBT using reverse engineering, we have not found any study that extracts EFSM [17]. In other words, all extracted models miss conditions of transitions. Consequently, this study bypasses this issue by extracting EFSM-based on reverse engineering. Before using the extended model for testing, the study validates and extends the model by adding functional requirements and business rules.

3. Proposed Approach

The current study develops an interactive framework for testing mobile applications. The framework consists of
three phases as it appears in Figure 1:

**Phase 1 (Model Extracting):** Using dynamic analysis reverse engineering technique to generate a GUI tree model.

**Phase 2 (Model Extension):** Generating EFSM model by transforming GUI tree to FSM and then extending the FSM model with missing business rules and functional requirements.

**Phase 3 (Model-based Testing):** Generating abstract test cases, converting them into executable test cases, and running the tests on the application under test.

### 3.1. Model Extraction

Figure 2 represents phase 2 of the proposed framework. Ripping algorithm [18] used to extract the GUI tree model is based on the dynamic analysis of the application by launching events and testing them within GUI as shown in Figure 3.

In GUI Tree, nodes express the GUI components of the application under tests and edges express the events triggered between these components. The crawler crosses the application GUI and generates GUI tree model using Depth First Search (DFS) algorithm. The Explorer launches events and collects information about the interfaces. Through this process, the navigator can test the first collision and detect runtime crashes by (java exceptions) and return the sequence of events causing them.

GUI Ripper tool [18] is used to run the ripper algorithm because it is an open-source and available for mobile applications. In addition, it has been used in similar studies on a range of mobile applications.

### 3.2. Model Extension

The second phase of our framework as it appears in Figure 4 transforms GUI tree to FSM model and then to EFSM by extending the former with business rules and missing functional requirements. EFSM is chosen as a
PROCEDURE RIP (SUT A)

\[ T = \text{get-top-level-windows}(A) \]

\[ GUI = T \]

\[ \text{FORALL} \ t \in T \ \text{DO} \]

\[ \text{RIP-RECURSIVE}(t) \]

PROCEDURE RIP-RECURSIVE (Window t)

\[ \Psi = \text{extract-widgets-and-properties}(t) \]

\[ \epsilon = \text{identity-executable-widgets}(\Psi) \]

\[ \text{FORALL} \ e \in \epsilon \ \text{DO} \]

\[ \text{execute}(e) \]

\[ C = \text{get-invoked-windows}(e) \]

\[ t = \text{extract-widgets-and-properties}(t) \]

\[ GUI = \text{merge}(GUI, C) \]

\[ \text{FORALL} \ c \in C \ \text{DO} \]

\[ \text{RIP-RECURSIVE}(c) \]

Figure 3. Ripping Algorithm

Figure 4. Phase 2 – Model Extension

1: digraph G {
2: main -> parse -> execute;
3: main -> init;
4: main -> cleanup;
5: execute -> make_string;
6: execute -> printf
7: init -> make_string;
8: main -> printf;
9: execute -> compare;
10: }

Figure 5. DOT file example from the Tomdroid application
final output model because of its understandability and power of expressing conditions and rules.

This phase comprises several subtasks:

3.2.1. Converting GUI tree to FSM
GUI tree model, the output of phase one is represented as a direct graph using DOT (Graph Description Language) [19], is transformed to FSM using visualization tool GRAPHVIZ\(^3\) in the following steps:

• Break any loop within the statement internally by reversing the inner direction of some edges.

• Determine a node for separate states. When crossing from top to bottom, events within the Y-axis are determined by edges that support more than one state divided by default nodes.

• Arrange the nodes based on the levels identified in the previous step in order to avoid the collision.

• Determine nodes values within the X coordinate axis to maintain short transitions and direct these transitions between nodes.

Figure 5 and 6 show a DOT file and its FSM conversion of Tomoroid\(^4\) application. Tomoroid is a note management application where any note can be created, saved and sent through other applications such as Facebook Messenger, Bluetooth, and Whatsapp.

3.2.2. Adding Business Rules and Missing Functional Requirements
The reversed model represents the GUI behavior of the application. Therefore, it can be used to generate tests to verify buttons, fields, text strings and input boxes. It can also be used to check user events that can be launched for the application for detecting any CRASH caused by the unexpected procedure. However, testing all functional requirements would not be possible. As a result, the proposed framework tends to add missing functional requirements and business rules before initiating the testing process. We use YED\(^5\) by Y-works to model and add rules and functional requirements in the model. YED is a pure Java Swing Application that works on Windows, Linux, Mac Operating Systems and any Java operating system. YED relies on the Y-files library to represent Java schemas that provide automated layout algorithms and analysis tools as well as a GUI that makes creating and editing charts easy. YED is able to use the user interface through the JIDE program but it receives the “graphml” files. DOT files are converted to “graphml” files within gv2ml command available in the Linux operating system, where an editable file is obtained by YED.

3.3. Model-Based Testing
The extracted and extended model is used for testing as it appears in Figure 7.

---

1 https://www.graphviz.org/
3 http://yed.yworks.com/
3.3.1. Test Generation
The model is traversed to obtain abstract test cases which will be a sequence of cases and transitions. The crossing starts with the starting node and stops when the required coverage criteria are met.

Graphwalker framework is used for crossing the model. Graphwalker is a JAVA library that generates test paths from FSM or EFSM model. Through the YED tool, the form is exported as a file with a *graphml* format that is suitable for importing by the Graphwalker. All nodes and transitions extracted from the model are extracted as software components and grouped into a table called a mapping table so that each abstract test case is connected to a piece of fragment code to be converted into an executable test case. Each test case is constructed by using an application emulator that extracts component properties from the GUI. This emulator is called the *UIAutomator* which is a framework used to test the functionality of the GUI for mobile applications installed on the device. The *UIAutomator* framework provides a set of programming interfaces (APIs) to build graphical test interfaces that interact with user applications and system applications. The interfaces allow you to perform a set of operations, such as browsing the settings menu or application launcher on any device. The *UIAutomator* framework is well suited to write tests based on the Black Box where the quiz does not support implementing details of the target application.

3.3.2. Test Execution
The abstract test cases are converted to the executable test cases within a mapping table by specifying the properties, as mentioned above, and writing the codes. Java is chosen to write the test code and use it as a translator and a test executer. The test application is selected on a Moto 7 Android device, and then the test is executed and reported by using *Appium*.

*Appium* is a testing system that supports an available source tool for mobile applications. It allows testing to run on physical devices and emulators; it can be used on Android and IOS applications. It is a available source tool, well documented, and has a forum and many users. It uses the WEB DRIVER JSON library, so it can be written in any language such as PHP, JAVASCRIPT, and C #, RUBY. We will use *Appiums JAVA* within the practical section of the current study.

4. Framework Evaluation
The evaluation process comprises the following:

4.1. Extended Model Evaluation
The model is validated and evaluated by the following measures:

- **Consistency**: The number of absolute events corresponding to the number of nodes and edges.
- **Accuracy Measure**: The Accuracy of the model is the ratio of paths in the model corresponding to the real paths in the application. We use the *FSMDiff* [20] algorithm to calculate the validity of the model and measure the similarity between two models; the first FSM and the Extended FSM after adding business rules and functional requirements. According to that, there are a set of measures to be defined:

  - **True Positive (TP)**: The number of correct paths in the model that is correct in the system.
  - **True Negative (TN)**: The number of invalid paths in the model rejected by the system (that is, they are not valid in the system as well).
  - **False Positive (FP)**: The number of correct paths in the model but rejected by the system.
  - **False Negative (FN)**: The number of rejected paths in the model but not rejected by the system.

\[
\text{Recall} = \frac{TP}{TP + FP},
\]

\[
\text{Precision} = \frac{TN}{TN + FP}.
\]

- **BCR** (Binary Classification Rate): (Recall + Precision) / 2.
- **Time**: Time consumed to extract the model.

---

4 http://graphwalker.github.io/
5 https://developer.android.com/training/testing/ui-automator
6 http://appium.io/
4.2. Test Evaluation
The test will be analyzed based on a set of criteria:

• **Defect Detection Effectiveness**: it includes the following measures:
  
  The number of crashes extracted during the testing process.
  
  The number of bugs: sometimes the evaluation is based on the name of the bug and its own knowledge.

**LOCS Coverage**: The ratio of the number of lines executed during the test to the total number of lines in the code.

Number of LOCS covered / Total number of applicable LOCs.

**Class coverage**: The ratio of the rows that were called during the testing process.

**Method coverage**: The ratio of the known functions that are called during the test to the total functions defined within the code.

• **Efficiency**: It includes the following measures:
  
  The number of test cases extracted.
  
  The number of test traces crossed during the test.
  
  • **Testing time**.

4.3. Overall Evaluation
We will determine the effectiveness of the proposed framework by answering the following questions through conducting two case studies:

Q1- Are the extraction and modification algorithm of the extracted model in the proposed framework more effective than similar approaches?

Q2- Is the coverage of the GUI components during the EFSM model-based testing effective and cost-effective compared to model-based testing similar approaches?

Q3- Whether the model extracted through the methodology has higher quality than the models extracted by similar methodologies?

Q4- Is the expanded extended model applicable to other types of testing techniques used on traditional models extracted through similar methodologies?

5. Case Studies
Two Android applications (Tomdroid and Arddict) were chosen to run the evaluation process since they are used in similar studies. That enables the comparison between this study and them. Both the case studies are implemented within Android systems. However, the study impact and results are not expected to differ from IOS systems or other environments. The main principles of the proposed framework are the same, but the implementing tools may vary.

5.1. Case Study 1 (Tomdroid)
Tomdroid is a note management application where any note can be created, saved and sent through other applications such as Facebook Messenger, Bluetooth and Whatsapp.

**Rules proposed in the Tomdroid Application**:

• Prevent sending a note from the settings page.

  • Prevent searching for values that express numbers only by adding the following condition to the transition: E16_Search [android: id / search_src_text.contains (“a-z”)].

  • Add the status of sending the note through the Whatsapp application (a functional requirement but not modeled or tested within the simple Ripper methodology).

Table 1 shows a comparison of the extracted model (by ripping algorithm) with the model extracted from the proposed framework for the Tomdroid application:

<table>
<thead>
<tr>
<th>The model</th>
<th>Consistency</th>
<th>Recall</th>
<th>Precision</th>
<th>BCR</th>
<th>Time</th>
<th>Number of events</th>
<th>Number of transitions</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI tree by Ripper</td>
<td>0.32</td>
<td>0.976</td>
<td>0.99</td>
<td>0.983</td>
<td>2:45 hour</td>
<td>16</td>
<td>42</td>
<td>47</td>
</tr>
<tr>
<td>EFSM by the Proposed Approach</td>
<td>0.82</td>
<td>0.985</td>
<td>0.99</td>
<td>0.987</td>
<td>3:15 hour</td>
<td>20</td>
<td>82</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 1. Comparison of the extracted model (ripping) with the proposed EFSM model (Tomdroid)
functional requirements. We notice that the time of producing the model is higher than the others since it comprises the time needed to extract in addition to the time needed to expand it with the missing functional requirements and business rules.

The test is executed using 100% nodes coverage and 100 transition coverage criteria. Figure 8 represents abstract test cases extracted from the enhanced model through the proposed framework. The abstract test cases are then converted to operational test cases using the UIAutomator tool. The tests are executed and testing results are recorded using Appium tool9.

Table 2 shows a comparison between the testing results of the proposed framework with those of similar approaches for Tomdroid application. In addition to detecting same errors of similar approaches, our framework detects new errors occurred while sending a note via the application (Whatsapp) and a set of errors appeared during the search where random values are entered.

5.2. Case Study 2 (Arddict):
Arddict is an application running on the Android operating system. This application has two main activities (Lookup and Article view) and two sub activities (Dictionaries and Dictionary Info). The application

---

9 http://appium.io/
starts from the search activity (lookup). The existing dictionaries on the device are pre-used by the user. However, if they are not found and have not been used before, there are options to download dictionaries from the Internet.

**Rules Proposed in the Arddict Application:**

- Search for values in English only.
- Add the status of searching for a dictionary within the device by pressing the scan device button.
- Add status Go to download a dictionary link by clicking on the link available on the start page.

Table 3 shows a comparison of the extracted model (by ripping algorithm) with the model extracted from the proposed framework for the application.

The test is executed using 100% nodes coverage and 100% transition coverage criteria. Figure 9 represents abstract test cases extracted from the enhanced model through the proposed framework.

Table 4 shows a comparison between the testing results of the proposed framework with those of similar approaches for Arddict application.

The Derived results concluded that the proposed framework extracted the same bugs and crashes extracted by the similar approaches, as well as, in addition to the crashes due to the business rules and missing functional requirements, and in closed time with greater lines of code coverage.

6. Framework Evaluation

The proposed framework is a new way of testing mobile applications based on previous reverse engineering and model-based testing approaches, with consideration to business rules and functional requirements which have not been considered previously on GUI testing. The framework is evaluated by answering the previously asked questions:

**Q1- Are the extraction and modification algorithm of the extracted model in the proposed framework more effective than similar approaches?**

**Answer:** Yes, they are more effective because the model extracted on the Tomdroid application is more accurate and more concise. As well, it maintains the same precision. Most similar methodologies have used reverse engineering techniques (static analysis, dynamic analysis, and hybrid analysis) to extract the model. By comparing the previous methodologies, the hybrid analysis could effectively be selected in order to extract the model. All system test cases can be taken as well as validation of the model in the proposed framework. However, hybrid analysis is not used in this research because of the tools’ availability issue. This study proves that proposing a model expansion by adding functional and business re-

<table>
<thead>
<tr>
<th>Approach</th>
<th>Model</th>
<th>Events</th>
<th>Time</th>
<th>Bugs</th>
<th>Crashes</th>
<th>LOC Coverage %</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed approach</td>
<td>EFSM</td>
<td>125</td>
<td>4.45 h</td>
<td>1</td>
<td>25</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>Ripper</td>
<td>GUI Tree</td>
<td>86</td>
<td>3 h</td>
<td>1</td>
<td>13</td>
<td>40</td>
<td>[11]</td>
</tr>
<tr>
<td>MobiGUITAR</td>
<td>State Machine</td>
<td>86</td>
<td>-</td>
<td>1</td>
<td>13</td>
<td>-</td>
<td>[6]</td>
</tr>
<tr>
<td>Monkey</td>
<td>Random</td>
<td>1500</td>
<td>12 h</td>
<td>0</td>
<td>0</td>
<td>46</td>
<td>[15]</td>
</tr>
<tr>
<td>Orbit</td>
<td>State Machine</td>
<td>101</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>70</td>
<td>[1]</td>
</tr>
</tbody>
</table>

**Table 2. Comparison between test results of Tomdroid application**

<table>
<thead>
<tr>
<th>GUI tree by Ripper</th>
<th>Consistency</th>
<th>Recall</th>
<th>Precision</th>
<th>BCR</th>
<th>Time</th>
<th>Number of events</th>
<th>Number of transitions</th>
<th>Number of nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI tree by Ripper</td>
<td>0.42</td>
<td>0.95</td>
<td>0.99</td>
<td>0.97</td>
<td>3:15 h</td>
<td>20</td>
<td>48</td>
<td>50</td>
</tr>
<tr>
<td>EFSM by the Proposed approach</td>
<td>0.85</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>4 hour</td>
<td>25</td>
<td>50</td>
<td>12</td>
</tr>
</tbody>
</table>

**Table 3. Comparison of the extracted model (ripping) with the proposed EFSM model (Arddict)**
Figure 9. Abstract test cases for Arddict application

<table>
<thead>
<tr>
<th>Approach</th>
<th>Model</th>
<th>Events</th>
<th>Time</th>
<th>Bugs</th>
<th>Crashes</th>
<th>Loc Coverage %</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed Approach</td>
<td>EFSM</td>
<td>185</td>
<td>6.25 h</td>
<td>3</td>
<td>25</td>
<td>87</td>
<td></td>
</tr>
<tr>
<td>Ripper</td>
<td>GUI Tree</td>
<td>125</td>
<td>4h</td>
<td>1</td>
<td>15</td>
<td>27</td>
<td>[11]</td>
</tr>
<tr>
<td>Mobiguitar</td>
<td>State Machine</td>
<td>105</td>
<td>-</td>
<td>1</td>
<td>18</td>
<td>-</td>
<td>[6]</td>
</tr>
<tr>
<td>Monkey</td>
<td>Random</td>
<td>1300</td>
<td>14h</td>
<td>0</td>
<td>0</td>
<td>51</td>
<td>[15]</td>
</tr>
<tr>
<td>Orbit</td>
<td>State Machine</td>
<td>122</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>65</td>
<td>[1]</td>
</tr>
</tbody>
</table>

Table 4. Comparison between test results of Arddict application and similar approaches

requirements counters better results derived from methodologies relied on hybrid analysis in terms of testing outputs and time consumed.

Q2- Is the coverage of the GUI components during the EFSM model-based testing effective and cost-effective compared to model-based testing similar approaches?
Answer: The main objective of the proposed framework is to test functions that are not directly extracted from the reverse transverse but this does not affect the GUI test itself. On the contrary, there is also a possibility of expanding the access GUI components by adding the missing functional requirements and test them.

Q3- Is the model extracted through the methodology higher quality than the models extracted by similar methodologies?
Answer: According to the results obtained above based on comparison criteria, they are concluded that the extracted model has higher quality than the rest of the models due to its comprehensiveness on all test cases, as
well as the possibility of adding guards on the transitions.

Q4- Is the expanded extended model applicable to other types of testing techniques used on traditional models extracted through similar methodologies? Answer: The EFSM extracted model can be used for mutation testing, whereas this is impossible to apply to the GUI Tree or Event Flow Diagram extracted from other methodologies.

Based on this analysis, we find that the proposed framework achieved similar results to the other reverse engineering and model-based testing approaches in terms of GUI testing. In addition, it proposed testing missing functional requirements and business rules based on EFSM model which has not been previously proposed; that lead to more effective and accurate results.

7. Conclusions and Recommendations

In this paper, we present a framework were MBT was applied to test mobile applications through their GUI. The main objective of the research is to verify whether MBT could be used in an effective way to test Android systems using the previous reverse engineering methodologies to extract the model then by extending it by missing functional requirements and business rules. The study shows that EFSM models can be used to test mobile systems and develop various methods for testing. The results are studied then compared with similar approaches. Finally, there are evaluative questions for the proposed framework which are also answered according to the results of the two applied case studies.

• The proposed framework is an integrative approach to existing methodologies. We didn’t invent or construct tools to achieve the idea. However, existing methodologies are integrated, solutions proposed and results analyzed by this research. The following are some of the prospects for this research:

• Extracting the business rules and functional requirements by reverse engineering.

• Studying the effectiveness of the framework for IOS applications.

• Applying the framework to larger and more complex applications.

References


