ON IMPROVING TEST AUTOMATION AND COVERAGE

Fatemah Aljamal
Yarmouk University
CIS department
IT faculty
Irbid, Jordan
Fatimaahmad2007@yahoo.com

Izzat alsmadi
Yarmouk University
CIS department
IT faculty
Irbid, Jordan
aisalsmadi@yu.edu.jo

ABSTRACT
Formal methods are used to provide formal and systematic models that can be used in software development to improve the correctness in those models. Formal methods are crucial, particularly, in critical systems where even minor mistakes can be pricey. In this paper, a formal method tool: ZEVS is used to describe requirements for two systems formally. Formal requirements are then used as an input for a tool used to automatically generate, execute and verify test cases. Path coverage algorithms are used to evaluate the effectiveness of the generated test cases and the ability of the formal model to describe all or most of the system functional requirements.

1. INTRODUCTION
Software testing is an expensive stage in software development. Reducing the cost of this stage has been the main subject for software testing researchers. The main approaches for such cost reduction includes: automation, and using formal methods to improve coverage and accuracy with a small number or subset of test cases.

Software testing process aims to ensure to both the customer and developer that the software meets the business and technical requirements and find defects or faults in the software that show deviation from users expectations, logical, semantic errors, etc. [23].

Figure 1 shows a general model for the testing process activities. The oval icon represents a testing sub stage or activity while the rectangle represents an activity output or deliverable. For example the output of the activity: designing test cases (or also called test case generation) is a set of test cases. Those are used as an input for the test execution process by defining input test cases along with expected results. Once test cases are executed and results are gathered actual outputs are compared with expected one for results verification.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Figure 1. General model for the software testing process activities, Sommerville (2007).

Exhaustive testing, where each component in the system is tested, is impossible. So, testing must be based on a subset of test cases. Coverage assessment is used to evaluate the effectiveness of generated test cases. There are many program perspectives that can be evaluated for coverage. For example, we can evaluate the amount of code the test cases covered relative to the whole code. Similarly, this can be applied to all paths, nodes, inputs, outputs, etc.

Formal methods are developed to handle the software quality or verification problems in the specification stage and before going to the design and implementation stages by providing mathematically-based techniques, so that the obstacles of the manual testing, such as: the high costs and time taken to test and fix bugs can be improved. Formal methods provide mathematically-based techniques, including formal specification, refinement, and verification [17].

In spite of the existence of some problems in formal methods in its obscure notation, the difficulty in using its tools, the high learning curve, the need to have some mathematical sophistications and theorem proving skills [20], it is still recommended and advised to use them to clarify the requirements of the systems so that their ambiguity can be reduced. Formal methods help may cost extra time and resources at the beginning of the development process with the expectation to pay that on later stages by providing well designed and described formal requirements and design [9]. For testing, formal methods have positive impact on testing as it simplifies and consolidates defining system inputs and expected outputs [1].

There are two main approaches in formal specification that are used in writing detailed specifications for software systems:

1. An algebraic approach, the system is described in terms of its operations and the relationships between them.
2. A model-based approach, the system is modeled using mathematical constructs such as: sequences and sets, and its operations are described by the way they modify the state of the system.

There are also several formal languages used. This includes: Z, Constructive Z, [10], VDM and Larch.

This paper uses a Z specification language based tool: Z/EVES as a tool to describe the two systems in the case study: an ATM and a University Registration (UR) system.

1.1 The Z specification language
The Z language is a formal specification language that enables the writing of mathematical description of system requirements. Such description consists of a mixture of formal and informal parts. It is considered as highly expressive, and support several formal specification styles and methods.

This language relies on the standard mathematical notation used in axiomatic set theory, predicate calculus, schema calculus, and first-order predicate logic. Z contains a standardized mathematical toolkit of popular mathematical functions and predicates [2].

Copyright ©2011 IJJ: The Research Bulletin of Jordan ACM - ISWSA; ISSN: 2078-7952 (print); 2078-7960 (online)
There are several Z editing tools, such as CADIZ [25]; ProofPower [18]; B-Tool, [4]; the Type Checker ZTC, and Z/EVES [22]. Each one of those tools has its own properties, and features despite the fact that all of them use Z on the background.

1.2 Z/EVES toolset
Z/EVES toolset is an interactive tool for composing, checking, and analyzing Z specifications. It is based on the EVES system, and uses its proof checker to carry out its proof steps. The language accepted by Z/EVES is a LATEX markup form [19]. This toolset helps in the analysis of Z specifications in several ways: (1) syntax and type checking, (2) schema expansion, (3) precondition calculation, (4) domain checking, and (5) general theorem proving [7].

The model checker of the Z/EVES is considered as user friendly and simple, especially when compared with other related tools such as Isabelle-HOL or ProofPower-Z. It could also prove its merit and popularity; due to its power in proving the specifications of critical systems written using the Z notation.

1.3 Test coverage
A test Coverage measures the percentage of the program proportion exercised by a test suite. A test criterion is a rule or set of rules that put constraints or conditions on a test set. Coverage level is the ratio of the number of the test requirements imposed by a test set to the size of the whole set of the test requirements [3].

Coverage measure is important for two main reasons. First, it is sometimes costly to achieve a coverage criterion, so it is necessary to make a trade off by attempting to accomplish a specific coverage level. Second, there are some requirements that cannot be met or achieved. So, it will be important to remove such requirements or replace them with less rigorous ones. Such types of requirements that cannot be met are called infeasible requirements. It means that there is not any test case value found that meets the test requirements.

In this paper, a formal model is used to write the specification of two systems; an ATM system and a University Registration (UR) system using the Z language, checking them on the Z/EVES toolset, due to its popularity and its appropriateness to such sequential systems. After that, a tool was built in C# in order to generate test cases dynamically from the formal model. Finally, Path coverage is evaluated automatically through the tool for those produced or generated test cases.

The rest of the paper is organized as the following: Section two summarized of the previous work related to the paper subject. Section 3 shows the methodology adopted in this paper. Section 4 shows results and analysis while the last section shows conclusion.

2. LITERATURE REVIEW
This section introduces the studies and related work to using formal methods in testing.

[16] tried to ease the final testing activity of the ATM system by describing its conceptual and formal models. They specified the formal model using the Z specification language, and then the Z/EVES toolset was used in order to check the output formal model.

[25] tried to improve the safety operations and quality services of the ATM system by presenting formal design, specification, and modeling for this systems’ static and dynamic behaviors, using Real-Time Process Algebra (RTPA), which is a notational mathematics. They introduced the conceptual model as the initial requirements of the system, and then created the architectural model through RTPA architectural modeling methodologies.

[5] used formal methods in specifying the requirements of the French railway interlocking system through using the functional Petri Net. The tool is used to check if there is an unsafe scenario disturbing a requirement. In such case, the tool will generate the proper test, and display it. Then it comes the role of the engineer, who will identify and repair the unsafe logic and re-run the tool again. They showed a noticeable reduction of testing time and costs, and a rise of the test coverage rate. In addition, formal methods proved to the infrastructure engineers that more safety is not necessarily more expensive.

[6] exploited the benefit of using formal methods to analyze the standard cryptographic protocols used to provide security-critical services such as authentication and secret keys distribution in critical systems. They provided a formal verification of the 802.11 shared key authentication protocol by S/A (Spi calculus Symbolic Analyzer), which is an automatic software tool that is based on a formal approach.

[12] illustrated a method known as Practical Formal Specification (PFS) for aircraft engine control and integrated it in a UML context with different forms of safety analysis. This method was developed to enhance traditional approaches in the development of embedded software systems by adding an engineering value, and being adaptable when integrated with existing well-established frameworks.

[13] formalized the requirement specification of the Air Traffic Control System (ATC); one of the highest safety critical systems and most challenging, as directed graph in terms of Z language, and checked the provided specification using the Z/EVES toolset.

[24] proposed an approach in order to solve the quality problem of medical guidelines and protocols prematurely and prior to the implementation and actual testing stages, relying on the use of formal methods. They modeled those protocols using a consolidated formal language and came out with a formal approach of them. After that, they evaluated the feasibility of such approach with respect to the formalization and verification of real-life medical protocols and could determine the number of errors that cannot be covered using this way; such evaluation step was the main objective of their work.

On the same direction, a group of medical experts evaluated the findings of the previous work, and judged that the detected problems in the protocols by the help of formal methods were serious and should be avoided.

[15] suggested an effective technique in order to overcome the difficulties of complete system requirements analysis for safety critical systems such as Nuclear Power Plant (NPP) software systems. This technique is an Integrated Environment (IE) approach that eases the process of inspection by incorporating requirement traceability and formal method effective use. They also support the IE approach for requirements by introducing a computer-aided tool called the Nuclear Software Inspection Support and Requirements Traceability (NuSISRST). This tool combines software inspection, requirement traceability, and formal specification capabilities. Finally, the formal requirements specification and analysis tool for nuclear engineering (NuSRS) was used for the purpose of formal requirements specification of the target system. As a result of their work, they were able to reduce some of the obstacles caused by the difference in knowledge between the system analyzer and designer via this effective technique for the software requirements analysis. The tools of their approach also help in the verification and validation V&V in the software requirement-phase.

[21] used a methodology called The KAOS goal-oriented requirements engineering methodology and the KAOS CASE tool in order to achieve a high formal assurance for a mission critical system, which is the railways signaling system, at an early stage. The reason behind working on that approach is to produce high quality requirements, so that they can ensure at an early stage that their system is being built effectively and efficiently. This formal methodology was used for the critical parts of the system, while the non-critical ones are modeled using a semiformal model, which helps missing requirements, wrong goal refinements or ignored assumptions to be discovered.

[11] analyzed a configuration protocol for a Bluetooth Location Network (BLN) by applying formal testing and validation techniques and tools. As they argue, a critical system like BLN needs to be
understood very clearly and precisely in premature stages before going to design and implementation, so they can avoid any serious problems in those stages and limit the time needed for them. In their work, they used Spin/Promela based formal analysis to test and validate that system. As a conclusion of their work, they proved that formal methods can be adapted and exploited successfully for such critical system. In other words, formal methods provided better understanding and discovered un-forecasted errors that may arise in later stages, so they can save very high costs and time needed to debug them if occurred later.

[14] applied the concept of formal methods on train control systems in the domestic railway industry because of their key importance in keeping the safety operation of train. They could achieve that, by incorporating the Z approach as a formal based technique; so they were able to get the benefits from the formal methods, and used state chart for the graphical formal specification in order to overcome the obstacles and drawbacks of excessive formal method application, like: the complexity of formal specification languages. By mixing the two approaches, formal and graphical they exploited the advantages from them both and increased the railroad requirements set forth which are called: RAMS (Reliability, Availability, Maintainability and Safety) together.

3. THE METHODOLOGY
In this paper, a formal model or specification is used to define the specification of two critical systems; an ATM system and a UR system using the Z language, checking that on the ZEVES toolset, due to its popularity, effectiveness and appropriateness for such sequential systems. After that, a tool was built in order to generate test cases dynamically from the formal model. Finally, the coverage and effectiveness of these test cases were evaluated automatically through the tool.

3.1 Overview of the methodology phases
Our methodology consists of several tasks or phases, as shown in figure (3.1), beginning by "writing the informal specification" of the target systems and their conditions in order to understand the way they work in and the conditions that govern them. After that, the second phase comes; which is "converting the informal specifications into formal specification using the Z language" so that formal methods are utilized. Third, the formal checking and verification in ZEVES is utilized to check for some of the basic properties that they system should have. Later on, The output file from ZEVES is used as an input for the automatic test case generation process that was developed programmatically through a tool developed for this purpose. The tool parse through the ZEVES file looking for the different path, nodes, edges and generate test cases with the goal of achieving high path coverage. Finally, coverage is evaluated automatically in the tool using the ZEVES file and the generated test cases. All phases after generating ZEVES file are designed to be executed automatically without user intervention.

4. EXPERIMENTS AND EVALUATION
In this section, we introduce the results gained from the last phase; the evaluation process, discussing the evaluation measures, the preparation of data for evaluation, a comparison between the algorithms used in each case study in terms of coverage evaluation, and comparing the evaluation results of the two selected case studies.

4.1 Evaluation Measures
Path coverage, node coverage and edge coverage are the three metrics used to evaluate the effectiveness of the generated test cases. Path coverage metrics are calculated based in the following equatins.

Path Coverage = # Of executed paths / # Of extracted paths ....................................... (1)

Node Coverage = # Of executed nodes / # Of extracted nodes ........................................... (2)

Edge Coverage = # Of executed edges / # Of extracted edges ............................................ (3)

4.2 Comparisons and Evaluation Results
In this research, related data about the main functional requirements of the ATM and University Registration systems are gathered and analyzed. For both systems, we study the user authentication functional requirements. In addition, for the ATM, we study the money withdrawal, deposit, transfer and balance Inquiry functional requirements. For the UR, we study the marks view, schedule view, course dropping, and course registration functional requirements. Next, results are evaluated with respect to the systems under test.

4.2.1 ATM Test Case Evaluation
For the ATM system, we used 2 algorithms in order to generate random test cases, and these algorithms were evaluated according to three fixed numbers of test cases and then they were evaluated in order to see which one of these algorithms is the most effective. In this system, the real extracted number of paths was thirteen, so we evaluate the two algorithms in three numbers of test cases; 5, 10 and 13 test cases, and results were shown in Table 1.

<table>
<thead>
<tr>
<th>T.C # / Algo. #</th>
<th>Algo. #1</th>
<th>Algo. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Test Cases</td>
<td>15.30%</td>
<td>38.40%</td>
</tr>
<tr>
<td>10 Test Cases</td>
<td>23%</td>
<td>76.90%</td>
</tr>
<tr>
<td>13 Test Cases</td>
<td>23%</td>
<td>100%</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>20.43%</td>
<td>71.77%</td>
</tr>
</tbody>
</table>

The average value for each algorithm's path coverage is illustrated in Figure 2.

![Figure 2. ATM Path Coverage Algorithms Average Results](image-url)

Following is a discussion of these two algorithms and their results in detail:

**Algorithm 1: The Random Algorithm**
In this algorithm, value of each part of the test case is given in a random way without any restrictions or limits. Thus, the coverage was very low. Table 2, and shows the results of this algorithm in terms of: path, node and edge coverage related to three numbers of test cases.

<table>
<thead>
<tr>
<th>T.C # / Algo. #</th>
<th>Node Cov.</th>
<th>Edge Cov.</th>
<th>Path Cov.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Test Cases</td>
<td>20.43%</td>
<td>71.77%</td>
<td>20.43%</td>
</tr>
<tr>
<td>10 Test Cases</td>
<td>23%</td>
<td>76.90%</td>
<td>23%</td>
</tr>
<tr>
<td>13 Test Cases</td>
<td>23%</td>
<td>100%</td>
<td>23%</td>
</tr>
</tbody>
</table>
Algorithm 2: Unique Paths Algorithm

In this algorithm, value to each part of the test case is given in a random way with restrictions based on finding unique paths in every cycle. Thus, the coverage was higher than the previous random algorithm. Table 3. shows the results of this algorithm in terms of: path, node and edge coverage related to three numbers of test cases.

<table>
<thead>
<tr>
<th>T.C #</th>
<th>Node Cov.</th>
<th>Edge Cov.</th>
<th>Path Cov.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Test Cases</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>10 Test Cases</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>13 Test Cases</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

4.2.2 UR Test Case Evaluation

For the UR system, we also used 2 algorithms in order to generate random test cases, and these algorithms were evaluated according to four fixed numbers of test cases and then they were evaluated in order to see which one of these three algorithms is the most effective.

In this system, the real extracted number of paths was forty two, so we evaluate the three algorithms in four numbers of test cases; 10, 21, 32 and 42 test cases. Table 4 shows the results from this algorithm.

<table>
<thead>
<tr>
<th>T.C #</th>
<th>Node Cov.</th>
<th>Edge Cov.</th>
<th>Path Cov.</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Test Cases</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>21 Test Cases</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>32 Test Cases</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>42 Test Cases</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>100.0%</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

This paper introduces the programming and implementation side in this research, and illustrates how the system behaves in each action, and the output of each step in it. The implementation is done using C# language.

4.3 The developed tool

As described earlier, a tool is developed in C# to perform all test activities automatically. This include: test case generation ( from for formal model), test case algorithm development, test case automatic execution, and test case path coverage evaluation. Figure 4. shows the main interface of the system. Figure 5. shows a sample of results from one of the algorithms and case studies described earlier.

<table>
<thead>
<tr>
<th>T.C #</th>
<th>Algo. #1</th>
<th>Algo. #2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Test Cases</td>
<td>11.90%</td>
<td>23.80%</td>
</tr>
<tr>
<td>21 Test Cases</td>
<td>21.40%</td>
<td>50.00%</td>
</tr>
<tr>
<td>32 Test Cases</td>
<td>21.40%</td>
<td>76.10%</td>
</tr>
<tr>
<td>42 Test Cases</td>
<td>23.80%</td>
<td>100.00%</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>19.63%</td>
<td>62.48%</td>
</tr>
</tbody>
</table>
In this paper, we have presented the generation of test cases from formal models for two systems, the ATM and the University Registration Systems using a Z based tool. We tried to show the usefulness of using formal methods while at the same time reducing one some of the main problems related to formal methods. This is as formal methods are usually seen as hard to learn and implement. By providing a tool that can use formal methods in the back ground while at the same time performing all or most activities programmatically, it is hoped that this can facilitate the spread of usage of formal methods while at the same time reducing some of the main problems related to formal methods.

Formal methods can be also used to demonstrate the testability of other systems. Nonetheless, formal methods are usually seen as hard to learn and implement. By providing a tool that can use formal methods in the background while at the same time performing all or most activities programmatically, it is hoped that this can facilitate the spread of usage of formal methods while at the same time reducing some of the main problems related to formal methods.

5. CONCLUSION

In this paper, we have presented the generation of test cases from formal models for two systems, the ATM and the University Registration Systems using a Z based tool. We tried to show the usefulness of using formal methods while at the same time reducing some of the main problems related to formal methods. This is as formal methods are usually seen as hard to learn and implement. By providing a tool that can use formal methods in the back ground while at the same time performing all or most activities programmatically, it is hoped that this can facilitate the spread of usage of formal methods while at the same time reducing some of the main problems related to formal methods.

Formal methods can be also used to demonstrate the testability of other systems. Nonetheless, formal methods are usually seen as hard to learn and implement. By providing a tool that can use formal methods in the background while at the same time performing all or most activities programmatically, it is hoped that this can facilitate the spread of usage of formal methods while at the same time reducing some of the main problems related to formal methods.

6. REFERENCES