ISSN No. 0976-5697

Volume 2, No. 6, Nov-Dec 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

System Testability Assessment and testing with Micro architectures

J.M.S.V.Ravi Kumar* M.Tech, (Ph.D)., Dr.M.Babu Reddy. Ph.D., I. Rajendra Kumar., P.ashok Reddy
Lakireddy Bali Reddy College of Engg.
Mylavaram, Andhra Pradesh, India.
*venkat7063@Gmail.Com, m_babureddy@yahoo.com, rajendralbrce@gmail.com, ashokreddimca@gmail.com

Abstract— Testing is necessary to ensure software reliability and fidelity. However, testing activities are very expensive and difficult. Micro architectures, such as design patterns and anti-patterns, widely exist in object oriented systems and are recognized as influencing many software quality attributes. Our goal is to identify their impact on system testability and evaluate how they can be leveraged to reduce the testing effort while increasing the system reliability and fidelity. The proposed research aims at contributing to reduce complexity and increase testing efficiency by using micro architectures. We will base our work on the rich existing tools of micro architectures detection and code reverse-engineering.

Keywords- Testing, testability, design patterns, anti-patterns test case generation.

I. INTRODUCTION

Software is now pervasive and essential in our daily life. Its reliability is, therefore, a major concern. Software failure may cause unacceptable financial losses or risk human lives in critical systems such as avionics and health systems. Testing is the most widely adopted practice to improve software quality and assure reliability and dependability. Though it does not guarantee the total reliability or absence of faults, testing allows to provide an increase in our confidence in software dependability, reliability and more generally user perceived quality. Testing activities are important and crucial throughout the entire software life cycle. They involve different abstractions levels.

Unfortunately, testing activities are very expensive, complex and time consuming. They represent up to 50% [1] of the effort and the cost of software development. Testing activities are very challenging. Many researchers have been trying to find ways to reduce their complexity increase their efficiency and reduce required effort. Several testing approaches and coverage criteria have been proposed with different results, almost always with the same goal: reduction of testing effort and cost by testing activity automation. In the literature, various levels of testing are reported such as unit testing, integration testing, system testing and regression testing. Black box approaches test the system functionalities and white box approaches assume that code is available. Other approaches aim at assessing systems testability. According to IEEE Standard Glossary of Software Engineering, IEEE 610.12, Testability is defined as the degree with which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met. Another definition presents the testability as a quality factor for programs, or program architectures that relates to the easiness for testing a piece of software [2].

The testability can be assessed at different stages of software development. Many researchers argue (e.g., [2], [3]) that this information is more useful early in the software life-cycle. Indeed, at early stages, testability information can be used to improve software artifacts such as the design, in order to facilitate testing with the ultimate goal to enhance software quality. The goal is therefore to integrate the means of reducing cost and complexity of the testing, early in the life-cycle: Other things being equal, a more testable system will reduce the time and cost needed to meet reliability and more generally, quality goals [4]. Object oriented software design often embeds micro architectures such as design patterns and antipatterns. These micro architectures convey intentions and often play important, yet specific roles. They influence significantly many quality attributes (eg.,[5], [6], [7], [8]). In this research, we are concerning with important research questions such as: What is the link between these micro architectures and the testability of systems? Can we exploit them in the testing perspective to increase the reliability and quality of a software system? These questions compel to explore the impact of these micro architectures on the testability and how micro architecture information can be exploited to improve the testing of the system and consequently its reliability and dependability. This exploration will extensively use or adapt existing tools to detect micro architectures and also rebuilt models from source codes. The remainder of the paper is organized as follows: In section 2, we give the motivation underlying this research work. Section 3 describes the research questions and section gives an overview of the methodology for answering them. We summarize our expected contributions in the section 5.

II. MOTIVATIONS

It is our belief that exploiting testability can significantly, improve testing while reducing its cost. Several previous works [2], [3], [4], [9] have addressed this and related research topics including approaches for testability assessment and testability improvement and also analysis of factors which impact systems testability. While analyzing the metrics for object-oriented systems testability, Bruntink et al. [9] discovered factors which influence the testability at unit testing level. Their goal was to identify and evaluate a set of metrics that could be used to assess the testability of Java Systems, with focus on unit testing. They performed five case studies on commercial and open source Java systems. They found that there is a significant correlation between metrics they studied and testing effort.

They suggested further studies to generalize these findings and also explore other testing levels. This is an example which shows that we need some evidence between testability and its influencing factors. Testability can be considered at different stages in the life-cycle of software. Many researchers [2], [3] argue that its assessment and its exploitation are more valuable in early stages of the life-cycle; for instance at the design stage. Indeed, testability is an important quality factor that risk being useless if it is not available early in the life-cycle [2]. This can explain why design for- testability is considered as a very important issue in software engineering. It becomes crucial in the case of OO designs, where control flows are generally not hierarchical, but are diffused and distributed over the whole architecture [10]. In this perspective, Baudry et al. [2] presented a study on the testability of design and how to integrate testability improvements into the usual design process.

The factor measured to assess the testability was called testability anti-pattern and corresponded to the undesirable configurations in the class diagram. To deal with the complexity of the overall design, they did a local analysis for the testability of the design by focusing on design patterns. They provided as result a testability evaluation grid quantifying the relation between each pattern and the severity of the testability anti-patterns. This evaluation grid considers most of design patterns and can be used to guide developers in improving design testability. Finally, they argued that testability can also be improved by adding specific stereotypes targeting testability and for example helping programmers to avoid useless object interactions.

These stereotypes could be automatically added for design patterns at the meta level, like testability constraints. They illustrated this approach with one design pattern, the Abstract Factory design pattern. However, this work did not explore the impact of the design pattern on the testability of the overall system and did not provide concrete experiment to support presented theory. More in general, design patterns and anti-patterns are design specifications which widely exist in object oriented systems. As the names suggest, the term design pattern indicates design solution believed to promote quality while anti-patterns are considered poor design solution, impairing quality. Design patterns have, since their introduction in the mid 90s, been the focus of various research trying to establish relations between design patterns and design quality. They are defined as reusable solutions to recurring design problems [5]. Many researchers claimed that design patterns impact positively quality attributes like maintainability, reusability and understandability, for instance [5], [6]. However, other studies, for instance [7], [8], rejected some of these claims stating that some design patterns can also degrade some quality attributes. These studies suggest that design patterns should be carefully used.

For example, Khomh et al [7] carried out an empirical study on the impact of design patterns on software quality attributes. They based their study on a questionnaire (for software engineers) about the usage of design patterns in software development or maintenance. Their analysis supported the conjecture that some quality attributes (understandability, reusability, expandability) may be negatively impacted by the use of some design patterns, in contrary to the common belief. In contrast to design patterns, the anti-patterns are defined as poor solutions to recurring design problems. In literatures, they are widely described as having a negative impact on the software quality. The numerous studies on these microarchitectures and their impact on software quality show their importance and influence they have on systems quality. The conclusions of these studies sometimes rejected popular beliefs and show thus the importance of such studies. They are the basis of taking rational decisions to improve the quality of systems. Our research stems from the observation that despite the vast amount of previous work, only a few contributions addressed, in general, the impact of these microcarchitectures on testability.

III. DISCUSSION

The class diagrams of Figures 1 and 2 represent a program’s abstraction in a system of compilation. System B uses the design pattern Visitor to implement operations performed by the objects of the data structure. The design pattern Visitor undoubtedly improves the maintainability, the extensibility, the understandability of the system relatively to the system A. In an other hand, it seems decrease the system testability. Indeed, the use of inheritance and the polymorphism in this pattern can make testing activities hard.

IV. RESEARCH QUESTIONS

Overall, our main research question is can micro architectures, and in particular design patterns and anti-patterns, exploit to improve systems testability and testing? At this early stage of the research, this question can be explored through the following research questions which are grouped in two main parts.

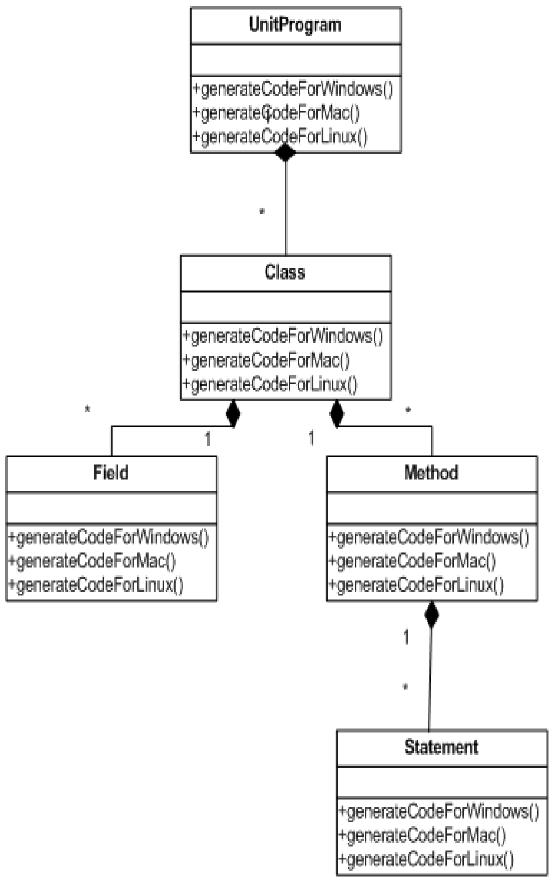

Figure 1. Program's abstraction in a system of compilation without design pattern system A)

A. ***Micro Architectures and Testability* :**

In this part, we plan to study the testability of design pattern and identify factors which explain it. We will also study the impact of micro architectures on the testability of the system.

a. Are design patterns testable?
b. What is the impact of micro architectures (design patterns and anti-patterns) on the testability?

B. Exploiting Micro Architectures to Improve Testing:

We want, in this second part, explore ways to exploit design patterns to reduce testing cost and increase its efficiency.

a. Can test of design patterns enhance software reliability?

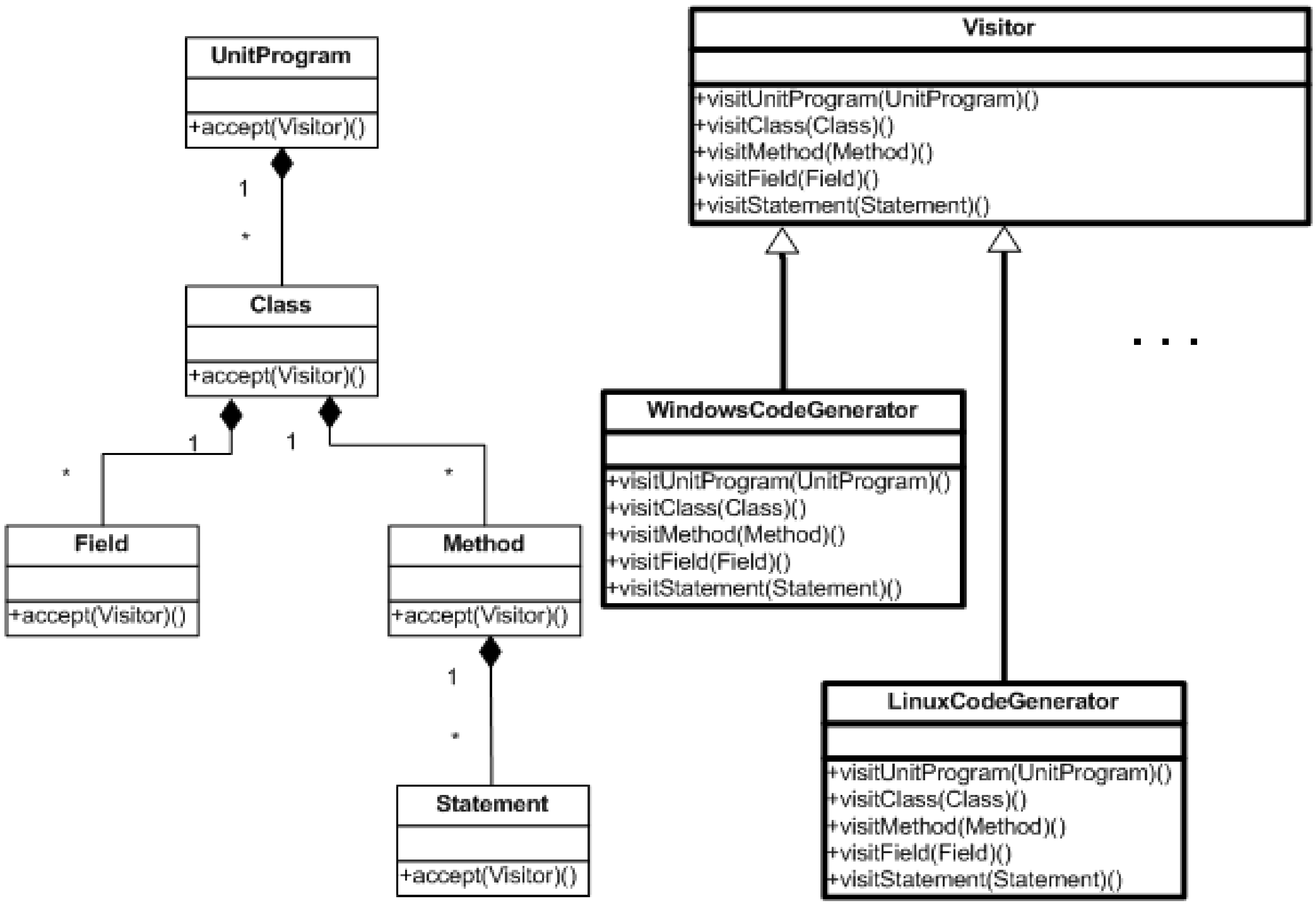

Figure 2. Program's abstraction in a system of compilation with design pattern (System B)

b. Can test of design patterns help to reduce testing cost?

c. Can test case generation for design patterns be generic enough to be reused across systems?

d. Can patterns be used to help test cases generation?

V. METHODOLOGY

In this section, we give an overview of the methodology we will follow to achieve our goal and answer the research questions.

A. Micro Architectures and Testability:

For addressing the first research question of this part, we plan to analyze various software applications and existing testability measurement approaches to identify suitable metrics, in order to quantify the testability of design patterns. These metrics would then be used to assess both the testability of design patterns and the way in which it can be improved. In order to study the impact of micro architectures on system testability, so answer the second research question of this part, we plan to perform a survey targeting testing engineers. The questions would focus on quantifying the impact of micro architectures on testing effort. For example, we would present a list of micro architectures and ask testers to identify from this list, the easiest and the worst to test. Such a survey would permit to collect opinions on the testability of systems containing micro architectures, such as design patterns and anti-patterns. Based on questionnaires analysis, we will classify the micro architectures according to their impact on system testability. We will perform some statistical analyses on survey data in order to identify trends, if any. For the same purpose, we plan to identify or develop systems with similar size and objectives, with and without micro architectures and assess their testability with existing methods. This would lead to draw some conclusions about the impact of these microarchitectures on the system testability, confirming or confuting questionnaires results. The results would provide a basis for refactoring techniques to improve the testability of systems built with microarchitectures (ie., design patter and anti-pattern). And more importantly, they would gain insight helping to further refine design methodologies.

B. Exploiting Micro Architectures to get Better Testing :

For the two first questions of this part, we plan to identify systems which contain design patterns and whose testing data are available. We will explicitly test some design patterns in order to verify whether this action enhances the overall software reliability or educe the number of needed test cases. For that, we would compare the results of testing systems containing design patterns (which are not explicitly tested) and of testing explicitly the design patterns of these systems. This study would help us to know if the test of design patterns actually enhances software reliability.

For the third question, we will attempt to define a etalevel devoted to testing, characterizing design patterns, with the goal to ease the reuse or ease automatic test input data generation for design patterns. And finally, to address the last question, we will study the various ways to use the knowledge of the existing patterns in the system to check if it is possible to reduce the number of test cases while keeping the same level of quality.

VI. PROBABLE RESEARCH CONTRIBUTIONS

Our main expected contribution is to reduce testing effort and increase its efficiency by using design patterns and anti-patterns. This include identification and quantification of the impact of these micro architectures on system testability and testing. Thus, based on this knowledge, we would suggest refactoring techniques and guidelines for making design decisions. This proposed research would also guide us to improve system testability at an early stage of the software life-cycle. Further, it aims at providing ways to exploit design patterns to reduce test cases generation and testing effort, and also increase the reliability of systems. The expected results will, firstly, be useful for designers. Designers will have a base to make rational choice about the use of design patterns in their systems and how they should use them in order to facilitate testing activities. The results will be also useful for testers. They will permit them to organize testing efforts but also easily generate test cases. Finally, developers can use the expected results to justify refactoring of systems.

VII. REFERENCES

[1]. M. J. Harrold, "Testing: a roadmap," in ICSE '00: Proceedings of the Conference on The Future of Software Engineering. New York, NY, USA: ACM, 2000, pp. 61–72.

[2]. B. Baudry, Y. Le Traon, G. Suny´e, and J.-M. J´ez´equel, "Measuring and improving design patterns testability," in METRICS '03: Proceedings of the 9th International Symposium on Software Metrics. Washington, DC, USA: IEEE Computer Society, 2003,

[3]. [3] S. Mouchawrab, L. C. Briand, and Y. Labiche, "A measurement framework for object-oriented software testability," Inf. Softw. Technol., vol. 47, no. 15, pp. 979–997, 2005.

[4]. R. V. Binder, "Design for testability in object-oriented systems," Commun. ACM, vol. 37, no. 9, pp. 87–101, 1994.

[5]. R. J. Erich Gamma, Richard Helm and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison- Wesley Professional, 1995.

[6]. B. Venners, "How to use design patterns A conversation with Erich Gamma, part I, year = 2005, publisher=http://www.artima.com/lejava/articles/gammadp.htm."

[7]. F. Khomh and Y.-G. Gueheneuc, "An empirical study of design patterns and software quality," pp. pages 1–19, 2008.

[8]. L. Aversano, G. Canfora, L. Cerulo, C. Del Grosso, and M. Di Penta, "An empirical study on the evolution of design patterns," in ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. New York, NY, USA: ACM, 2007, pp. 385–394.

[9]. M. Bruntink and A. van Deursen, “An empirical study into class testability,” J. Syst. Softw., vol. 79, no. 9, pp. 1219–1232, 2006.

[10]. B. Baudry and Y. L. Traon, “Measuring design testability of a uml class diagram,” Inf. Softw. Technol., vol. 47, no. 13, pp. 859– 879, 2005.