Exploiting Software Architecture to support Requirements satisfaction Testing

Paul Clements[1], María J. Escalona[2], Paola Inverardi[3], Ivano Malavolta[3], Eda Marchetti[4]
[1]Software Engineering Institute, Carnegie Mellon University, USA
clements@sei.cmu.edu
[2]Department of Languages and Computer Systems, University of Sevilla, Spain
mjescalona@us.es
[3]Department of Computer Science, University of Aquila, Italy
{paola.inverardi,ivano.malavolta}@univaq.it
[4]Software Engineering Laboratory, ISTI-CNR Pisa, Italy
eda.marchetti@isti.cnr.it

ABSTRACT

Currently, software testing is mainly carried on independently from software architecture-related information. Some approaches propose to perform integration and regression testing with respect to software architecture descriptions, but less attention has been paid to analysing software architecture in order to develop a less costly and time-consuming test plan that covers the requirements of the system of interest. If on one side, it is well known that providing an effective test plan is crucial to software quality, on the other side software testing is extremely difficult because it stems from the complexity of current software systems.

In this paper we (i) elaborate on how a new architectural analysis can help in producing better test plans and (ii) identify a set of corresponding research challenges. We believe that considering and analysing architectural information for requirements satisfaction testing purposes will provide substantial benefits in terms of test plan specification process, test plan effectiveness, and test cases understandability.

1.INTRODUCTION

Testing is one of the most labor-intensive activities in the software development life cycle, and consumes a substantial portion of total development costs. Estimates range from 30% to 50% [16], over 50% [3], 50% to 75% [12], even 80% or more [8]. Any technology, method, or approach that even slightly lowers the cost or increases the efficacy of testing can make a substantial dent in the tens of billions of dollars spent on software development annually.

Our work focuses on improving testing by applying knowledge about the architecture of the system being tested. So-called "architecture-based testing" is an idea with roots in the mid-1990s [1, 18]; work in architectural means to increase a system's testability going back even further [15, 10].

Much of that work can be characterized as opportunistic; that is, researchers' insights about architecture and testing were used to produce ideas about approaches with (in most cases) unknown and untried applicability. By contrast, this paper reports on the results of a piece of *directed* research in which architecture knowledge is brought to bear on testing in response to a specific practitioner problem.

1.1 Problem being solved

The problem being solved is one that came to us from the testing practitioner community. In February of 2011, the Software Engineering Institute in cooperation with the University of L'Aquila held a workshop on architecture-based testing attended by testing practitioners in the defense, aerospace, industrial systems, financial, control systems, and automotive industries. The purpose of the workshop was to produce a set of model problems, which are problems that (if solved) would provide a significant improvement in testing outcomes [2]. Workshop participants produced 29 such problems, and voted on the ones they would most like to see solved. Here is the problem that participants chose as being the most important, stated as a scenario:

> *At a time when the architecture is complete and system test has not yet begun, a tester needs to choose a test set to test the system for requirements satisfaction. The tester uses an architecture analysis tool that identifies the smallest number of tests to run to provide coverage of 98% of the requirements. Redundant tests are eliminated. Performing the analysis is much less costly and time-consuming to run than the tests it replaces.*

The idea is that practitioners wish to use the architecture to help them determine that (out of the enormous number of tests possible) a particular, small set of tests will cover 98% of the requirements.

1.2 Example scenario

In order to illustrate our idea, we introduce a pedagogical exam-

ple, the Arcade Game Maker (AGM) [1]. The AGM product line will produce a series of arcade games. Each game is a one-player game in which the player controls, to some degree, the moving objects. The objective is to score points by hitting stationary obstacles. The games range from low obstacle count to high and will be available on a variety of platforms.

Figure 1 presents the set of use cases for this system which covers three games: bowling, pong and brickles.

In the description of the AGM system, each use case is described with a detailed scenario. In order to test functionality of the final system, we should test each scenario and we should assure the right coverage of each execution path. Current research allows us to select which are the more critical execution paths or offers us heuristics criteria to select which are the redundant lines.

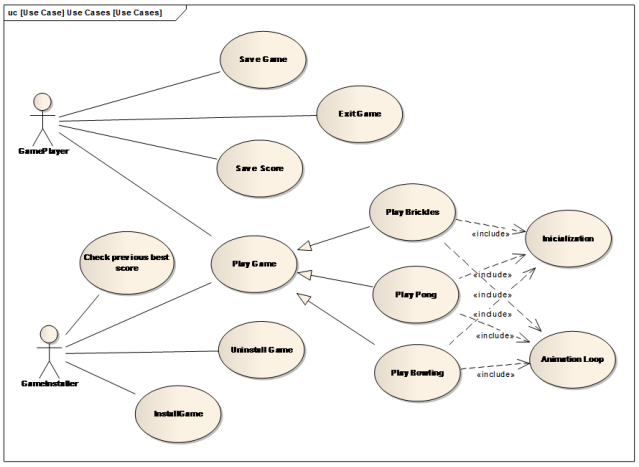

Figure 1: A set of use cases for the Arcade Game Maker

However, our research is oriented to analyse if we could reduce the set of test cases to check these requirements, depending of the architecture, and assuring the grade of coverage. For instance, depending of how we implement Play Game use case and use cases that extend it, we could reduce the number of test. If common functionality for the three use cases are implemented in a abstract and a general class, we could check it only once and drop repetitive execution paths or repetitive execution activities.

Extracting testing only from requirements, using common approaches like [9], we have to derive, at least, one test case for each use case: Play Game, Play Bowling, Play Brickles and Play Pong. If they have common functionality in the architecture, the direct derivation of test cases from requirements could not detect it. As it is presented in the next section, our approach is oriented to enrich the information of requirements with architectural details in order to improve the set of test cases that better exercise the system.

2. WHAT IS THE NEW IDEA?

In this section we propose a new perspective on how software architecture descriptions can be used. More specifically, we propose a new role for software architecture modeling, that is: architectural descriptions (especially together with their related architecture analysis capabilities) can be exploited to better test the system under development, both in terms of requirements coverage and testing efficacy. Such a new perspective comes from explicitly considering architectural analysis results when testing system requirements satisfaction.

Intuitively, if designers consider requirements only, they are basically doing black-box testing; if designers consider also the implementation code, they are doing white-box testing. Our main goal is to exploit software architecture-related information in order to perform grey-box testing. Indeed, our ambition is to be able to exploit architectural information in developing test plans so that it can be possible to positively assess the effectiveness of such test plans against the requirements.

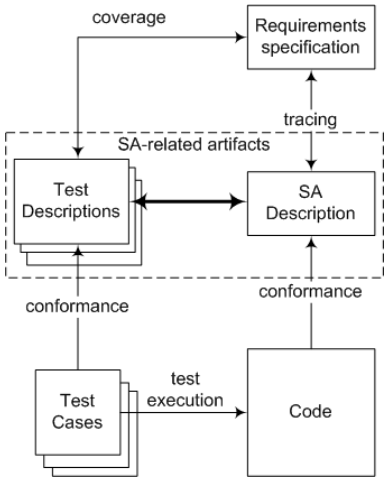

Figure 2: Overview of the new testing process

Figure 2 gives an overview of the testing process we envision. Central to our testing process is the description of software architecture (*SA Description* in figure). It may be a model conforming to a specific architecture description language (ADL), a UML model of the system, or any other representation of the architecture of the system. A set of *tracing* links relate architecture-related requirements in the *Requirements specification* with their corresponding architectural elements. Those links may be at different levels of granularity, i.e., they may relate the whole architectural model to a set of requirements, a single architectural element (e.g., a component, an architectural layer, etc.) to a requirement, and so on. It should be noted that our approach does not force engineers to use some specific means to define requirements: they can use plain text, UML models (for example, in Figure 1 we specified requirements as use case diagrams), or any other means. The *Code* box represents the implementation of the system. The code conforms to the software architecture of the system if it is synchronized with the intended meaning of the *SA Description*.

The implementation of the system is exercised by defining and executing a set of *Test Cases*. They are the typical test cases that help testers in determining whether the realized system is working correctly or not; for example, they may be JUnit [2] tests. *Test Descriptions* represent the same information of implementation-level test cases at the architectural level of abstraction. In the same way as the implementation of the system must conform to its architecture description, implementation-level test cases must conform to their corresponding test descriptions. Such a *conformance* relationship ensures that the results of the architectural analysis identifying the smallest number of tests to run to provide a certain coverage of the requirements are correct with respect to the implemented system (since the analysis is performed on consistent artifacts). The

[1] `http://www.sei.cmu.edu/productlines/ppl/requirements_model.html`

[2] `http://www.junit.org/`

coverage relationship between a test description and a requirement in the requirements specification means that the test description tests the system with respect to the requirement it is linked to. Obviously, the coverage relationship is a many-to-many relationship, i.e., many requirements may be covered by many test descriptions.

It is important to note that this new perspective helps in solving a well-known issue in model-based testing. Indeed, though model-based testing has been successfully used in many projects, current model-based testing techniques do not focus enough on testing non-functional requirements (like performance, security, reliability, etc.) [5]. As a matter of fact, one of the main strong point of software architecture is that it supports the analysis of non-functional properties of the system, thus exploiting architectural information during the testing activities should help in the test plans definition.

Furthermore, software architecture descriptions are multi-view and recent research in the software architecture area goes towards the automatic support of multiple architectural views and their interrelationships [4].

One crucial point in our approach is that we are not assuming any specific development process. We are just assuming that, given a system, an architectural description of the system exists and that such description conforms to the system under test (SUT). Thus, the architectural description can also be retrieved with reverse engineering technologies if not available from the design phase.

Considering architecture descriptions, requirements and testing artifacts in combination will help in identifying those test cases that may play an important role while evaluating the various tradeoffs during the architecting phase. By executing those test cases, designers are exercising the system (and its architecture) right in its tradeoff points, which are typically the most crucial and sensitive parts of the whole project.

Our new perspective on the relationship between software architecture, requirements, and testing artifacts triggers many interesting research questions; next section will present them.

2.1 Research questions

The main research question we identify in this work can be formulated as follows:

> *How software architecture descriptions can be exploited to enhance the effectiveness of a requirements satisfaction test plan?*

In other words, we are looking for a solution in which software architecture-related information is exploited in order to better support requirements satisfaction testing. Considering software architecture descriptions as first-class artifacts for requirements satisfaction testing is not straightforward. Indeed, there are many points that still need both theoretical and technological investigation. More specifically, we refined the main research question of this work into a set of secondary research questions:

RQ1. Find the right level of abstraction for describing test descriptions at the architectural level, i.e., we must specify what information should be part of an architectural test description. This aspect has many implications; for example, it will trigger the possibility to automatically generate (or at least to trace towards) concrete test cases defined at the implementation level. Moreover, it may allow test designers to automatically generate test descriptions from the combined information coming from requirements and the architecture description.

RQ2. Find a systematic or semi-automated way to derive or support mappings

a) from requirements to software architecture description,

b) from requirements to test cases (by passing through architectural test description).

RQ3. Define metrics for testing coverage of architecture in combination with requirements. It is important to note that coverage criteria are based on architecture analysis; this is not a direct requirement of the model problem, but a derived requirement: knowing the architecture (and the tests descriptions that most effectively exercise it) will help designers pick the tests that are most effective in prosecuting the model problem. For example, to carry out Req1 and Req2 involves the same paths through the architecture, so perhaps a test designer can fully test Req1 and Req2 to a lesser degree, as opposed to fully test both.

RQ4. Develop a notion of "tradeoff test cases" that shall represent the implementation counterpart of the prioritization among requirements, especially non functional ones. These test cases can be derived from the trade off analysis conducted during the architecture design if it exists, or can be the result of the analysis of the interdependencies among different architectural views.

3. WHY IS IT NEW?

Architecture-based testing is a common term adopted for describing the activity to increase a system's testability [15, 10, 7]. However the use of architectural information to support requirements satisfaction is still a research challenge and few proposals are currently available. Indeed a large part of the literature has been dedicated to requirements satisfaction exploiting functional models, as reported in one of the author's recent work [9]. The practical attitude of software-development organizations is to consider requirements and architectural specification as two different faces of the same problem. So far, the effort of researchers has been focused to join together these two aspects either presenting innovative development processes as in [13] or applications and methods that map (specific) requirements into the architecture, as surveyed in [17]. Our problem is not about formally relating requirements and architectural specification, rather we assume conformance and the existence of a mapping in the same way practitioners do between requirements and code when designing their test plans. Our approach is to exploit the architectural analysis for deriving testing plans that can be directly related to the requirements specification.

The analysis of related works evidences that the chain (requirement specification, software architecture definition and code verification) is far to be completely automated. Current proposals partially cover pieces of this path and usually they face the problem from a research point of view. New development processes [19, 6], additional formalisms or facilities [14] are the common suggestions [11]. However there is still a lack of practical solutions close to the best practices adopted by software-development organizations to automate as much as possible the code verification based on requirements and architectural specification. Therefore the approach we undertake is to not rely on any specific process development assumption and restrict ourselves to consider only the existence of a set of requirements, a set of software architecture artifacts and a conformance mapping.

4. EXPECTED FEEDBACK

In section 2.1 we described the main research question motivating our work, and we further refined it into a set of secondary research questions. Those research questions are interesting from a pragmatic point of view: systematic automation is part of the value proposition, and metrics would provide evidence of efficacy. We believe that both software architecture and testing research communities will benefit from being exposed to the expectation of a representative class of industrial representatives. Indeed as reported in the introduction, the main research question of this paper came

directly from industrial needs and shed a different light on the way we have been traditionally thought the role of software architecture in testing. It is therefore interesting to confront the approach we propose with the Software Engineering community at large since many different techniques and analysis can potentially play a role in the approach we have devised and other industrially relevant case studies can be offered to validate the approach.

5. FUTURE WORK

In this paper we elaborated a new perspective on how to exploit architecture-related information and analysis with the aim to achieve a better coverage of system requirements, and to improve the efficacy of test cases.

Our near term future work will be to investigate and elaborate solutions to the specific research questions outlined in Section 2.1. Although all research questions have technical aspects, they all will be usefully answered only in the context of industrial practice. Finding the "right" abstractions for describing test cases architecturally, finding metrics for architectural coverage in testing, and finding how to usefully express "tradeoff cases" depends on trying out proposed solutions through experimentation and demonstration in an industrial setting. We will start our search for (semi-)automation of requirements-architecture-test mappings by building upon current approaches to this problem that are used in everyday development.

In all cases, we envision practitioner workshops to vet our ideas, experiments in which we apply our ideas to real-world projects and take qualitative and quantitative improvement measures, and building method-supporting artifacts such as templates, tools, and training for those solutions that have proven their value in practice.

As a start in this direction, we are currently considering an industrial project that has been developed at the lab of one of the authors. We have a complete set of developed artifacts, including test plans and test suites. The objective of our study is to address the described research questions in this experimental setting, redesigning the test plans based on the analysis of the architecture artifacts with respect to the system requirements. Then a comparison between the test cases generated in the project and the test cases generated following our analysis will be performed.

In the long term we plan to arrange the solutions of the various research questions into an integrated testing framework. By doing this, the resulting integrated framework may help in promoting architecture-centric development: software architecture should be used as the main driver of the development of the system, and should be integrated into development processes that consider also requirements, implementation, testing, and so on [4].

Most importantly, we will continue our interaction with testing practitioners. This will provide us with a continuously updated understanding of what are the concrete needs of practitioners, and assess our solutions for practicality and usefulness.

Acknowledgments

This work is partially supported by the EU project CONNECT (http://connect-forever.eu/) No 231167 of the FET - FP7 program, the EU project CHOReOS (http://www.choreos.eu) No 257178 of the FET - FP7 program, and the Tempros project of the Ministry of Education and Science (TIN2010-20057-C03-02), Spain.

6. REFERENCES

[1] A. Bertolino and P. Inverardi. Architecture-based software testing. In *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops*, ISAW, pages 62–64. ACM, 1996.

[2] P. Clements, J. Hudak, P. Place, A. Bertolino, H. Muccini, and P. Inverardi. Architecture Support for Testing: Connecting Research to Practice. Technical Report , SEI report, in progress, 2011.

[3] G. T. Daich, G. Price, B. Ragland, and M. Dawood. Software Test Technologies Report. Technical Report p. 11, Software Technology Support Center, August 1994.

[4] D. Di Ruscio, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio. Developing next generation ADLs through MDE techniques. In *Int. Conf. on Software Engineering, ICSE 2010*, 2010.

[5] A. C. Dias-Neto and G. H. Travassos. A picture from the model-based testing area: Concepts, techniques, and challenges. volume 80 of *Advances in Computers*, pages 45 – 120. Elsevier, 2010.

[6] A. Douglas. Linking requirements and implementation. *Views on Evolvability of Embedded Systems*, pages 137–151, 2011.

[7] R. Dssouli and R. Fournier. Communication Software Testability. In *IFIP Transactions, Protocol Testing Systems III (the Proceedings of IFIP TC6 Third International Workshop on Protocol Test Systems)*, pages 45–55, 1991.

[8] R. Dunn. *Software defect removal*. McGraw-Hill, Inc., New York, NY, USA, 1984.

[9] M. Escalona, J. Gutierrez, M. Mejías, G. Aragón, I. Ramos, J. Torres, and F. Domínguez. An overview on test generation from functional requirements. *Journal of Systems and Software*, 84-6:1379–1393, 2011.

[10] R. Freedman. Testability of software components. *IEEE Transactions on Software Engineering*, 17:553–564, 1991.

[11] A. Goknil, I. Kurtev, K. van den Berg, and J. Veldhuis. Semantics of trace relations in requirements models for consistency checking and inferencing. *Software and Systems Modeling*, pages 1–24, 2009.

[12] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Syst. J.*, 41:4–12, January 2002.

[13] J. Hall, M. Jackson, R. Laney, B. Nuseibeh, and L. Rapanotti. Relating software requirements and architectures using problem frames. In *Proceedings of the IEEE Joint Int. Conf. on Requirements Engineering, 2002.*, pages 137–144. IEEE, 2002.

[14] IBM. IBM Rational Unified Process (RUP).

[15] V. Jeffrey M. Factors that affect software testability. Technical report, NASA, 1991.

[16] Q. Li, M. Li, Y. Yang, Q. Wang, T. Tan, B. Boehm, and C. Hu. Bridge the gap between software test process and business value: A case study. In *Proceedings of the International Conference on Software Process: Trustworthy Software Development Processes*, ICSP, 2009.

[17] S. Pandey and K. Mustafa. Recent advances in sre research. *Recent Advances in SRE Research*, 2(04):1079–1085, 2010.

[18] D. J. Richardson and A. L. Wolf. Software testing at the architectural level. In *Joint proc. of the second int. software architecture workshop (ISAW-2) and int. workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops*, ISAW '96, pages 68–71.

[19] E. Woods and N. Rozanski. Unifying software architecture with its implementation. In *Proceedings of the Fourth European Conference on Software Architecture*, pages 55–58, New York, NY, USA, 2010. ACM.