Towards a Reference Architecture for Software Testing Tools[*]

Elisa Yumi Nakagawa[†], Adenilso da Silva Simão, Fabiano Ferrari, and José Carlos Maldonado
Dept. of Computer Systems
USP - University of São Paulo
São Carlos/SP, Brazil
{elisa, adenilso, ferrari, jcmaldon}@icmc.usp.br

Abstract

Producing high quality software systems has been one of the most important software development concerns. Software testing is recognized as a fundamental activity for assuring software quality; however, it is an expensive, error-prone, and time consuming activity. For this reason, a diversity of testing tools has been developed, however, they have been almost always designed without an adequate attention to their evolution, maintenance, and reuse. In this paper, we propose an aspect-based software architecture, named RefTEST (Reference Architecture for Software Testing Tools), that comprises the knowledge to develop testing tools. This architecture is strongly based on separation of concerns and aspects, aiming at evolving, maintaining and reusing efforts to develop these tools. Our experimental results have pointed out that RefTEST can contribute to the development and reengineering of testing tools.

1. Introduction

Software engineering activities have been fundamental to achieve high quality systems development. In special, software testing is one of the most important activities to guaranteeing the quality and the reliability of the software under development [15]. In this context, the availability of tools makes the testing a more systematic activity and minimizes the cost, the time consumed, as well as the errors caused by human intervention. Testing automation is an important issue related to the quality and productivity of the software process. A diversity of testing tools — commercial, academic, and open source — automating software testing tasks can be found. However, these tools have almost always been implemented individually and independently, presenting its own architectures and internal structures. As a consequence, difficulty of integration, evolution, maintenance, and reuse of these tools are very common. These tools often focus on automating specific testing techniques and criteria; without considering the whole testing process. Besides, while software architecture has become an increasingly important research topic in recent years, contributing to software quality [19], the investigation and establishment of software architectures for testing tools have been largely ignored. As known, the establishment of architectures, specially reference architectures, for a given domain consolidates the knowledge about how to develop tools to that domain, contributing to the reuse of design expertise.

Recently, Aspect-Oriented Programming (AOP) has arisen as a new technology to support a better separation of concerns and to more adequately reflect the way developers reason about the system [11], to contribute to maintainability, reusability, and easiness to write software. Besides programming, aspects have also been explored in the early life cycle phases including the requirements analysis, domain analysis and architecture design phases.

In this paper, we investigate the use of aspect in architecture design phase and present an aspect-based reference architecture, named RefTEST (Reference Architecture for Software Testing Tools), that supports the development of software testing tools. Aspect-based architecture refers to architectures that use aspects — from AOP — as mechanism to establish the communication among the modules that compose this architecture. RefTEST is also strongly based on separation of concerns, aiming at providing reusability, maintainability, and capability of evolution to the testing tools built based on this architecture. For the purpose of communicating adequately the knowledge into RefTEST, architectural views were developed and are presented here.

The remainder of this paper is organized as follows. At first, the background about software architecture and related work are presented. Next, we present the proposed archi-

[*]This work is supported by Brazilian funding agencies CAPES, FAPESP and CNPq.

[†]She is also professor of the Dept. of Administrative Science and Technology, Uniara - Araraquara University Center, Araraquara/SP, Brazil.

tecture for testing domain, discussing its establishment and use. Finally, our conclusions and future directions are presented.

2. Background

Over the last decades software architecture has received increasing attention as an important subfield of Software Engineering [19]. According to Shaw and Clemments [19], in the near future, software architectures will attain the status of all truly successful technologies. As already highlight in [22], software architectures play a major role in determining system quality — performance and reliability, for instance —, since they form the backbone for any successful software-intensive system. Software architecture is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [2]. In this context, reference architectures have been investigated as a mechanism that promotes reuse of design expertise and achieves well-recognized understanding of a specific domain. Reference architectures for diverse domains — e-commerce, for instance — can be found. Although, there is a lack of adequate reference architectures for software engineering domain, including software testing.

In order to design correctly and document clearly software architectures, ADLs (Architecture Description Languages)[1], as well as architecture evaluation methods such as ATAM (Architecture Tradeoff Analysis Method)[2], have been proposed. Documentation based on architectural views [9] and UML (Unified Modeling Language)[3] have been investigated as a promising approach to document architectures.

2.1 Related Works

Few works related to software architecture for testing domain have been conducted [5, 23]. Eickelmann [5] established the first reference architecture for this domain. This architecture establishes six functions of testing tools: test execution, test development, test failure analysis, test measurement, test management, and test planning. However, the relationship among these functions, as well as practical and detailed considerations were not established. Based on Eickelmann's work, Yang [23] established a more detailed architecture; however, the focus of this work is the testing of web applications. As a consequence, it is concentrated in how to structure the tools to test web applications. In spite of the Yang's work not proposing a reference architecture, it has contributed with the establishment of a set of concrete and abstract class, as well as operations of testing tools. Another work established an architecture for testing tools deployed in web platform [7]; although this work does not address the activities directly related to testing tools core.

[1]http://www.sei.cmu.edu/architecture/adl.html
[2]http://www.sei.cmu.edu/architecture/ata_method.html
[3]http://www.uml.org

Besides these works, we can find architectures of specific testing tools, such as [12, 14]. These architectures did not consider issues related to evolution, reusability, and maintainability. Also, these architectures do not support or foresee the support to all activities of the testing process, such as configuration management, planning, documentation, among others. When supported, the functionalities related to these activities are implemented in the scattered and tangled way in the core of the testing tools. Also, those works exclusively related to architecture of testing tools [5, 23] do not support all activities of the testing process.

Considering not only the relevance of software architectures, but also the lack of works related to architectures in the software testing domain, we have proposed a reference architecture for that domain, presented next.

3. Reference Architecture for Testing Domain

RefTEST is based on a more generic reference architecture, named RefASSET (Reference Architecture for Software Engineering Tools) [16] that supports the development of tools and SEEs (Software Engineering Environments). Thus, before to present RefTEST, we will briefly discuss its principles next.

3.1. Principles of RefTEST

In order to support adequately the development of tools and SEEs, it is first important to understand the relationship among the software engineering activities. We have adopted the concept of separation of concerns, one of the key principles in Software Engineering. According to Harrison [8], an adequate separation of concerns is pointed out as the main mechanism to build evolvable and reusable SEEs.

In the context of software development process, we have considered each software engineering activity as a concern [18]. In order to identify all concerns, we adopted the international standard ISO/IEC 12207 (Information Technology - Software Life Cycle Processes) [10] that provides a comprehensive set of life cycle processes, activities, and tasks for developing and maintaining software system. This standard establishes and groups the activities (concerns) into three categories — primary, organizational, and supporting — depending on the role they play in the software development. According to its characteristics, a concern can be either a primary concern — that refers to activities that perform the development, operation, and maintenance of software systems — or a crosscutting concern, term used by the AOSD (Aspect-Oriented Software Development) community and referred to concerns that are spread throughout or tangled with others concerns. It is important to highlight that an inherent characteristic of the crosscutting concerns (i.e. the organizational and supporting concerns) is their occurrence along all primary concerns, as illustrated in Figure 1. For instance, documentation is one of the concerns that must be considered from requirements specification to maintenance.

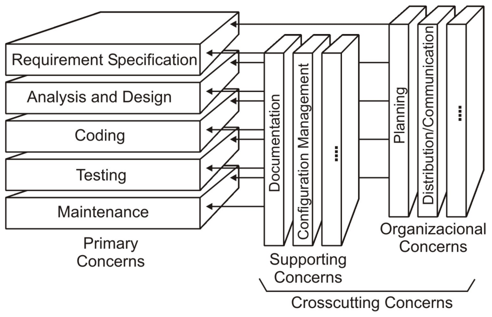

Figure 1. Relationship among Primary, Supporting and Organizational Concerns

The ideas discussed above were consolidated in RefASSET, illustrated in Figure 2. This architecture is not the focus of this work, but it is briefly described herein because it is the basis of RefTEST. In short, RefASSET is a reference architecture based on architectures already investigated, such as ECMA Reference Model [4] and architectures of interactive systems and Web systems (architectural pattern MVC (Model-View-Controller) and 3-tiers architecture[4]), considering the future perspective of Web as platform to provide diverse software systems, including software engineering tools.

It should be highlighted that the `Application Layer` in Figure 2 contains all identified concerns (`primary`, `supporting`, and `organizational`). In particular, the `primary concerns` part contains the core of the tools that implement the primary concerns.

RefASSET has been explored in order to facilitate the use and integration of tools, processes, and artifacts in SEEs. By considering separation of concerns in this architecture, we are pursuing easier integration, maintainability, quality, and reusability of the environments built based on this architecture.

3.2. Establishing RefTEST

RefASSET was specialized to software testing domain, resulting in RefTEST; thus, RefTEST inherited all RefASSET's characteristics, such as the separation of concerns and the architectures of interactive systems. In particular, the specialization concentrated in identifying the core of testing tools, i.e. the core concepts into `primary concerns` part. Three steps were conducted:

Step 1: Investigation of the Software Testing Knowledge

This step was conducted to identify the core activities performed during the testing. Knowledge into software testing domain was arisen, considering diverse information sources. Software testing processes, such as [20, 21], and the activities contained in these processes were investigated and identified as potential core activities. Software architectures proposed in the testing literature [5, 23] were also investigated and the knowledge (structure, modules, and their relationships) was considered. Besides that, as known, an ontology is a formal explicit specification of a shared conceptualization, providing a vocabulary for representing and communicating knowledge about some topic and a set of relationships which hold among the concepts in that vocabulary as well; thus, an ontology for software testing domain, named OntoTest [1], was basis to identify the concepts and relationships into testing domain. Moreover, the investigation of testing tools was the most important contribution for this step. The most known eight academic testing tools were considered, such as MuJava [12] and Jazz [14], and two broadly known and used commercial testing tools: those of the Mercury[5] and Rational[6]. Our experience in developing testing tools — for instance, published in [3] and [13] — was also important. As a result, 32 activities were identified, illustrated partially in Table 1. The information sources of each activity are also indicated in that table. For instance, activity "Generate test requirement" was identified through investigation of software testing processes (P), testing tools (T) and testing ontology (O). It is noticed that these activities refer to the core activities of testing; rather, the activities related to testers management and test planning, for instance, were set aside, since these are addressed by planning_management module, related to a crosscutting concern.

Step 2: Identification of the Core Concepts

Based on the identified activities, the core concepts were established. Each activity was related to a software testing concept. For instance, the activity "Include test cases" was related to the concept "test case", the activity "Generate test requirements" was related to "test requirement", and so on. It is important to highlight that only four concepts were identified and that they seem to be sufficient to represent the core elements of testing tools: test case, test requirement, test artifact, and test criteria. Figure 3 presents the conceptual model of the core of testing tools. Besides this

[4]http://www.sei.cmu.edu/str/descriptions/threetier.html

[5]http://www.mercury.com/us/products/quality-center/

[6]http://www-306.ibm.com/software/sw-bycategory/subcategory/SW730.html

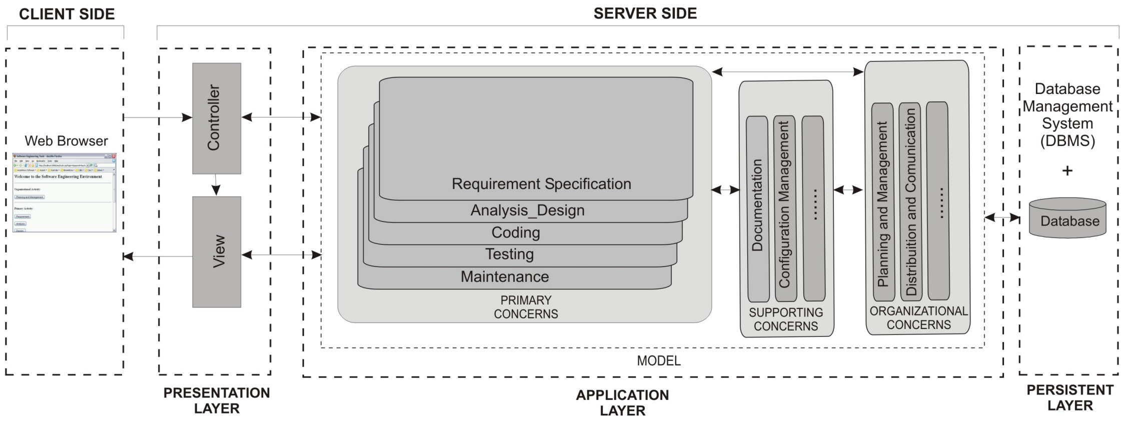

Figure 2. Reference Architecture of Software Engineering Environments

Table 1. Activities in the Testing Conduction

Core Activities	Source[a]
Deal with artifact to be tested	A, P, T, O
Execute artifact to be tested using test cases	A, P, T
Import test cases	P, T
Include test cases	P, T
Generate test requirements	P, T, O
Execute test requirements using test cases	P, T, O
Establish testing adequation criterion	A, O
Calculate test coverage	A, P, T
...	...

[a]P = testing processes; A = architectures; T = Testing Tools; O = OntoTest

model, definitions and related operations were established for each concept.

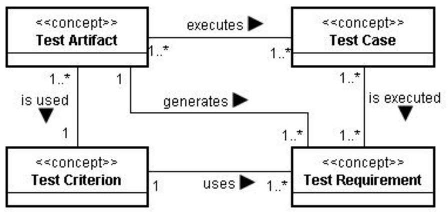
<<concept>> Test Artifact
<<concept>> Test Case
<<concept>> Test Criterion
<<concept>> Test Requirement
executes
generates
is used
is executed
uses
1..*
1

Figure 3. Core of Testing Tools

Step 3: Architectural Design

For the purpose of representing adequately the RefTEST, as recommended by [9], architectural views — module, runtime, deployment, among other views — were developed using UML 2.0. Also, extensions of the UML notation are required to support the aspects representation [17], since this architecture is based on aspects. For the sake of space, only the module and runtime views are presented herein.

The module view, in Figure 4, shows the structure of the software in terms of code units; packages and dependency relations are used to represent this view. While the core of testing tools (`testing_tool` package) can be implemented using classes, components, or subsystems, the modules that implement crosscutting concerns use aspects in their structures. For instance, the package `supporting_crosscutting_modules` contains modules that are implemented by classes and aspects. These aspects crosscut other modules and change the execution flow of these modules inserting functionalities related to a crosscutting concern. Thus, there are dependency relations, labelled with <<`crosscut`>>, among the package `supporting_crosscutting_modules` and other packages. It is important to highlight that we have explored the use of aspects to implement modules for SEE and used them as an integration and communication mechanism among modules in SEEs, including testing tools.

Differently from module view, the runtime view shows the structure of the system when it is executing. Thus, the interface among the modules is an important element that must be detached. In [17], the extensions required to UML to represent interfaces when aspects are used as a communication mechanism are discussed. In short, the Amodules (Aspect-based Modules) — i.e. modules that exclusively use aspects to communicate with other modules — have a filled circle attached by a solid line, representing their interface, named IMA (Interface Made by Aspects). This interface represents a declaration of a set of coherent features and obligations; in practice, it represents the characteristics — for example, objects and their operations — that modules must have to use the AModules. Also, it was proposed a half-square attached by a solid line to the module that is crosscuted or affected by aspects into AModules. Our experience points out that the establishment of the IMAs provides facilities to reuse AModules. Figure 5 presents the runtime view of RefTEST. In this figure, `Planning_management` and `Documentation` are AModules.

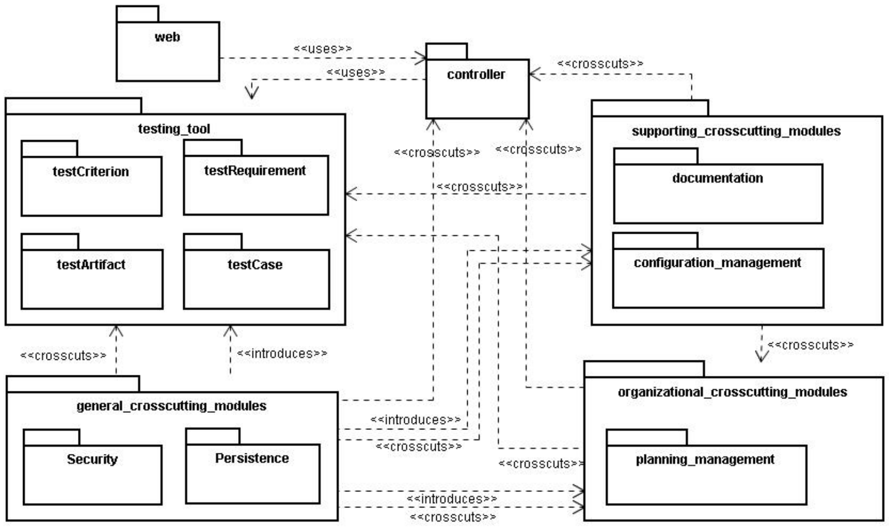

Figure 4. Module View of RefTEST

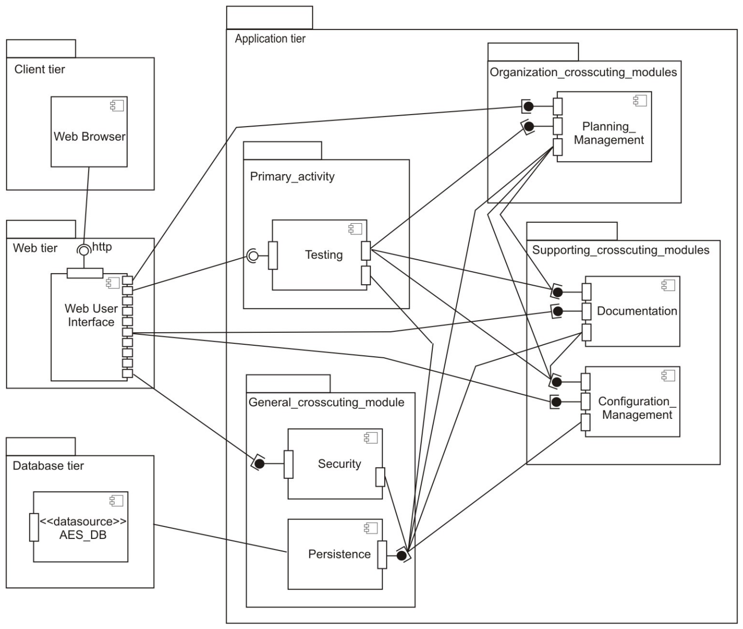

Figure 5. Runtime View of RefTEST

3.3. Use of RefTEST

We have used RefTEST to developing and reengineering of testing tools that are deployed in web platform [6]. Recent technologies to develop evolvable and reusable systems have been investigated and adopted; Java[7] and AspectJ[8] are in use. Also, we have developed modules that implement crosscutting concerns. For instance, a testing documentation module was implemented; the aspects collect information, such as the test cases and test requirement, during the tool runtime and format, update, and store these information. This module was designed and implemented in order to facilitate its evolution to other testing tools and tools of other domains, such as requirement specification and analysis/design, aiming at developing a documentation framework. In the end, we have pursuing a unique module that manages the documentation of all other software engineering activities that can be developed in an evolvable and

[7]http://java.sun.com/

[8]http://www.eclipse.org/aspectj/

incremental approach. Moreover, our experimental results have showed that the core of the tools can be independently developed of the modules that implement crosscutting concerns.

4. Conclusions

The systematization of the testing tools development is a real need of the testing research community. Considering the lack of recent works that establish reference architecture for testing domain, in this paper we proposed RefTEST, a reference architecture strongly based on separation of concerns, pursuing evolvability and reusability of the tools built based on this architecture. In special, this work contributed with the establishment of the testing tools core and also with a new point of view about how to address modules that implement crosscutting concerns. It is important to highlight that the use of aspects to develop and integrate these modules is another important contribution of this work. Our experience has pointed out that the development of testing tools using a well-established reference architecture is relevant. RefTEST have been an important guideline to minimize the efforts to the testing tools development.

References

[1] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado. Towards the establishment of an ontology of software testing. In *18th Int. Conf. on Soft. Engineering and Knowledge Engineering (SEKE'06)*, San Francisco Bay/USA, July 2006.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Publishing Company, 2003.

[3] M. E. Delamaro, J. C. Maldonado, and A. P. Mathur. Interface mutation: An approach for integration testing. *IEEE Trans. on Software Engineering*, 27(3):228–247, Mar. 2001.

[4] ECMA and NIST. Reference model for frameworks of software engineering environments, Dec. 1991. Special Publication Report No. ECMA TR/55, 2nd Ed.

[5] N. S. Eickelmann and D. J. Richardson. An evaluation of software test environment architectures. In *18th Int. Conf. on Software Engineering (ICSE'96)*, pages 353–364, Berlin, Germany, 1996.

[6] F. C. Ferrari. *Supporting Structural and Mutation Testing of Object-Oriented and Aspect-Oriented Software*, 2006. Doctoral Work Plan, University of São Paulo, São Carlos, SP, Brazil. (in Portuguese).

[7] J. Gao, C. Chen, Y. Toyoshima, and D. Leung. Development an integrated testing environment using the World Wide Web technology. In *21st Inter. Computer Software and Applications Conference (COMPSAC'97)*, pages 594–601, Washington, USA, Aug. 1997.

[8] W. Harrison, H. Ossher, and P. Tarr. Software engineering tools and environments: a roadmap. In *13th Conf. on The Future of Software Engineering (ICSE'00)*, pages 261–277, New York, NY, USA, 2000.

[9] IEEE. 1471-2000 - Recommended practice for architectural description of software-intensive systems, Sept. 2000.

[10] ISO. ISO/IEC 12207. Information technology – software life-cycle processes, 1995.

[11] G. Kiczales, J. Irwin, J. Lamping, J. Loingtier, C. Lopes, C. Maeda, and A. Menhdhekar. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proc. of the European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, 1997.

[12] Y.-S. Ma, J. Offutt, and Y.-R. Kwon. Mujava: a mutation system for Java. In *ICSE'06 - 28th Int. Conf. on Software Engineering*, pages 827–830, New York, NY, 2006.

[13] J. C. Maldonado, M. E. Delamaro, S. C. P. F. Fabbri, A. S. Simão, T. Sugeta, A. M. R. Vincenzi, and P. C. Masiero. Proteum: A family of tools to support specification and program testing based on mutation. In *Mutation 2000 Symposium – Tool Session*, pages 113–116, San Jose, CA, Oct. 2000.

[14] J. Misurda, J. Clause, J. L. Reed, B. R. Childers, and M. L. Soffa. Demand-driven structural testing with dynamic instrumentation. In *27th Int. Conf. on Software Engineering (ICSE'05)*, St. Louis, Missouri, USA, May 2005.

[15] G. J. Myers, C. Sandler, T. Badgett, and T. M. Thomas. *The Art of Software Testing*. John Wiley & Sons, 2004.

[16] E. Y. Nakagawa. *A Contribution to the Architectural Design of Software Engineering Environments*. Phd thesis, University of São Paulo, São Carlos, SP, Brazil, Aug. 2006. (in Portuguese).

[17] E. Y. Nakagawa and J. C. Maldonado. Representing aspect-based architecture of software engineering environments. In *1st Workshop on Aspects in Architectural Description, 6th Int. Conf. on AOSD*, Vancouver, Canada, Mar. 2007.

[18] E. Y. Nakagawa, A. S. Simão, and J. C. Maldonado. Addressing separation of concerns in software engineering environments. In *IASTED Int. Conf. on Software Engineering*, Innsbruck, Austria, Feb. 2007.

[19] M. Shaw and P. Clements. The golden age of software architecture. *IEEE Software*, 23(2):31–39, Mar/Apr 2006.

[20] I. Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, 6 edition, 2001.

[21] A. M. R. Vincenzi, M. E. Delamaro, A. S. Simão, and J. C. Maldonado. Muta-pro: Towards the definition of a mutation testing process. In *Proc. of the 6th IEEE Latin American Test Workshop*, Salvador, Bahia, Mar 2005.

[22] A. I. Wasserman. Toward a discipline of software engineering. *IEEE Software*, 13(6):23–31, Nov. 1996.

[23] J. Yang, J. Huang, F. Wang, and W. C. Chu. An object-oriented architecture supporting web application testing. In *23rd Annual Int. Computer Software and Applications Conference*, pages 122–127, Phoenix, Arizona, USA, Oct. 1999.