Ontology-based Development of Testing Related Tools

Ellen F. Barbosa, Elisa Y. Nakagawa, Ana C. Riekstin, José C. Maldonado
University of São Paulo – ICMC/USP
Av. do Trabalhador São-Carlense, 400
P.O. Box 668, 13560-970 – São Carlos (SP), Brazil
{*francine, elisa, claudiar, jcmaldon*}*@icmc.usp.br*

Abstract

Testing constitutes one of the most relevant software engineering activities in order to guarantee the quality of the software under development. Besides that, the automation of testing activity is an important issue to be addressed – the availability of testing tools makes the testing a more systematic activity and minimizes the errors caused by human intervention. In spite of their advantages, most of testing tools are still built in an ad hoc way and considerable rework is necessary if it is desired to configure or adapt them to different testing techniques and criteria. In this paper we discuss the using of an ontology of software testing (OntoTest) in the development of testing related tools. Two perspectives have been investigated: (i) the development of an OntoTest-based application to share and harmonize testing concepts; and (ii) the specification of a common set of functional modules that testing tools should provide, aiming at establishing an OntoTest-based testing architecture.

Keywords: *Ontology, Software Testing, Testing Tool.*

1. Introduction

Producing reliable, robust and high quality software systems is one of the most important software development concerns. Testing, which intends to reveal the existence of faults in the software, constitutes one of the most relevant software engineering activities, being crucial to guarantee the quality of the software under development. Besides that, the data collected during testing phases is also important for debugging, maintenance, and reliability assessment [11].

The software testing domain involves integration of three basic types of knowledge – theoretical, empirical and tool specific. In fact, a significant amount of information should be mastered to perform an effective testing activity (e.g., testing techniques and criteria, testing phases, testing steps, testing artifacts, testing tools). Such diversity of concepts and their inter-relations make the establishment of a consensual shared understanding on software testing a fundamental issue to be addressed.

Ontologies play an important role in this perspective. An ontology is a formal and declarative representation which includes [6, 13]: (i) the vocabulary required for referring to the concepts in the subject area; and (ii) the logical statements which describe what the concepts are and how they are related to each other. Hence, it provides a vocabulary for representing and communicating knowledge about some topic as well as a set of relationships which hold among the concepts in that vocabulary.

Currently we are establishing *OntoTest* [2], an ontology of software testing, which aims to support acquisition, organization, reuse and sharing of knowledge on the testing domain. Based on ISO/IEC 12207 [9] and on the Falbo and Bertollo's work [4], *OntoTest* intends to explore different perspectives involved in the testing activity, such as techniques and criteria, human and organizational resources, and automated tools.

Also regarding the quality and productivity of the software development process, the automation of testing activity is another important issue to be addressed. Indeed, applying a testing criterion without the support of a testing tool is an error-prone activity.

In order to automate software testing, commercial and academic testing tools have been developed in the last years. Despite their advantages, most of testing tools are still built in an ad hoc way. As a consequence, is difficult to identify a well-defined architecture for them. Besides, since a testing tool typically supports a unique testing criterion, considerable rework is necessary if it is desired to configure or adapt it to another criterion or technique.

In this paper we discuss the using of *OntoTest* in the development of testing related tools. Two scenarios are considered: (i) to developing an *OntoTest*-based application for organizing and searching testing concepts, aiming at disseminating testing information; and (ii) to defining a common set of functional modules that different testing tools

The authors would like to thank the Brazilian funding agencies (FAPESP, CAPES, CNPq) and to the QualiPSo Project for their support.

should provide, aiming at establishing an *OntoTest*-based architecture for testing tools.

The remainder of this paper is organized as follows. In Section 2 the related work is presented. Section 3 describes *OntoTest* and its general structure. In Section 4 we discuss how testing related tools can be built based on *OntoTest*. In Section 5 we summarize our contributions and discuss the perspectives for further work.

2. Related Work

An ontology is a formal explicit specification of a shared conceptualization [6]. That is, a simplified way of perceiving a piece of reality, often conceived as a set of relevant terms and their relationships, whose structure is constrained by some rules. Basically, it consists of concepts and relations, as well as their definitions, properties and constraints expressed by means of axioms [13].

Ontologies have been applied to describe a variety of domains, such as medicine, engineering and law. In software engineering, several ontologies have already been identified – software engineering ontology [15, 16], software quality ontology [1], software process ontology [4], enterprise ontology [14], software testing ontology [7, 17], software maintenance ontology [10], among others.

Falbo and Bertollo [4], for instance, developed an ontology of software process to support the acquisition, organization, reuse and sharing of software process knowledge. The ontology is designed to support software process definition and automatization in a meta-SEE (Software Engineering Environment), which is able to generate, by means of instantiation, SEEs adequate to the particularities of specific software processes, application domains and projects. Every knowledge based tool linked to the software process ontology shares a common vocabulary, facilitating the communication between developers and, also, allowing the sharing and reuse of knowledge bases in the meta-environment as much as in the instantiated SEEs.

Regarding software testing, Huo et al. [7, 17] investigated the development of an ontology of testing as a support for a multi-agent software environment for testing web-based applications. The idea is to use the ontology to enable flexible integration and to mediate communication between multiple agents. In short, a taxonomy of testing concepts (basic and compound concepts) is established. The ontology is represented in UML, at a high level of abstraction, and in XML to codify the knowledge of software testing for agents' processing of messages.

3. OntoTest: an Ontology of Software Testing

Most of the testing concepts considered in Huo's work [7, 17] are in agreement with the concepts established in *OntoTest* [2]. The software process ontology, defined by Falbo and Bertollo [4], and the ISO/IEC 12207 standard [9] were also considered, focusing on the idea of a testing process. Additionally, we have explored concepts from: (i) definition and evaluation of testing criteria; and (ii) knowledge and experience in testing tools development.

Due to the complexity of the testing domain, we have adopted a layered approach to the development of *OntoTest*. On the ontology level, the Main Software Testing Ontology addresses the main concepts and relations associated with testing. On the sub-ontology level, specific concepts from the Main Software Testing Ontology – testing process, testing artifacts, testing steps, testing strategies and procedures, and testing resources – are refined and treated into details. Figure 1 shows the graphical representation of the main ontology. We adopted UML notation for representing it.

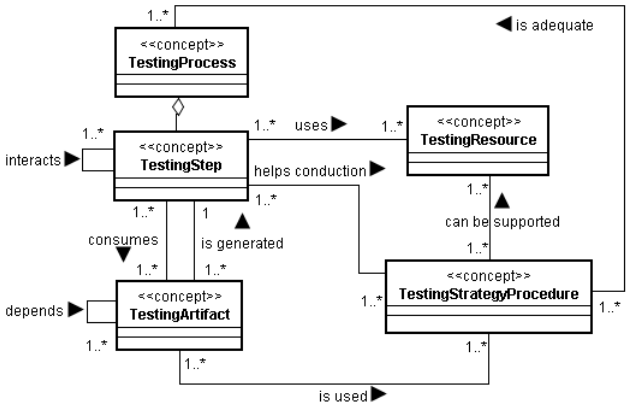

Figure 1. Main Software Testing Ontology

To develop the Main Software Testing Ontology, we established an analogy between software process and software testing process [9, 4]. Similar to the software process, which is a sequence of steps required to develop and maintain software, a testing process can be seen as a sequence of testing steps required to develop and maintain the testing activity. A testing step can consume and/or produce several testing artifacts* and can use different testing resources. Moreover, when defining a testing process, it is important to determine how the testing steps shall be performed. To do so, we must establish the testing strategies and procedures, adopted in the accomplishment of the testing steps.

For each basic concept represented in the Main Software Testing Ontology, there may be a number of subconcepts which are refined and treated in the sub-ontologies [2]:
Testing Process: Testing processes are defined based on the development paradigm as well as on the application do-

*A testing step can also consume some artifacts produced by other activities of the software development process (e.g., a requirement specification document, a quality plan, and so on).

main. Testing life cycle models are also used as a reference in the definition of a testing process, establishing its main testing steps and the dependency relations among them.

Testing Step: When performing a test, a set of essential steps should be considered [11]. Each testing step is composed of testing activities. Based on the ISO/IEC 12207 standard [9], we have classified a testing activity as primary, organizational or supporting one, depending on the role it plays in the testing process. As primary activities of software testing we have identified: (1) test case design; (2) artifact under testing handling; (3) test requirement establishment; (4) test execution; and (5) test analysis and measurement. Similarly, organizational and supporting testing activities have also been established.

Testing Artifact: Each testing step may consume and/or produce different testing artifacts (test documents, test cases and test requirements). In terms of test documentation, eight different documents, established on the basis of the IEEE 829 standard [8], are represented in the sub-ontology. Test cases consist in the input data against which the software is executed, in conjunction with the output data observed [11]. Test requirements correspond to the required elements to be exercised during the testing activity [11].

Testing Strategy Procedure: Different testing strategies and procedures can be established according to the artifact under testing (system requirement, specification, design, source code), and depending on how testing phases (unit, integration, system or regression testing), testing approaches (requirement-based, specification-based, design-based or implementation-based) and testing techniques (functional, structural, error-based, state-based, random or ad-hoc) are combined. Testing techniques are responsible for establishing the testing criteria to be adopted. Control-Flow (e.g., All-Nodes, All-Edges, All-Paths) [11], Data-Flow (e.g., All-Definitions, All-Uses) [12], and Mutation Analysis [3] are some of the representative criteria that can be found in the literature. Also, testing techniques support testing methods, which are based on testing guidances (standards and guidelines).

Testing Resource: Testing resources are required to the accomplishment of a testing step. The Testing Resource Sub-Ontology is presented in details in Section 4.

OntoTest has been described in OWL [†] (Web Ontology Language), using the Protégé [‡] ontology development tool.

4. OntoTest and Testing Related Tools

We have explored *OntoTest* in two scenarios: (i) to developing a tool to disseminate and harmonize testing concepts; and (ii) to structuring an ontology-based architecture for testing tools.

[†] http://www.w3.org/TR/owl-ref/

[‡] http://protege.stanford.edu

4.1. An OntoTest-Based Application

Figure 2 illustrates *OntoTestSearchingTool* – a web application, based on *OntoTest*. It allows searching for testing concepts, showing concept's details such as: (i) description, (ii) direct superclasses and subclasses (concept hierarchy); (iii) *has part* and *is part of* relations (concept composition); (iv) images; (v) facts (additional information related to the concept); (vi) references; and (vii) examples. Figure 2 shows the results of a query on the concept *TestingTool*.

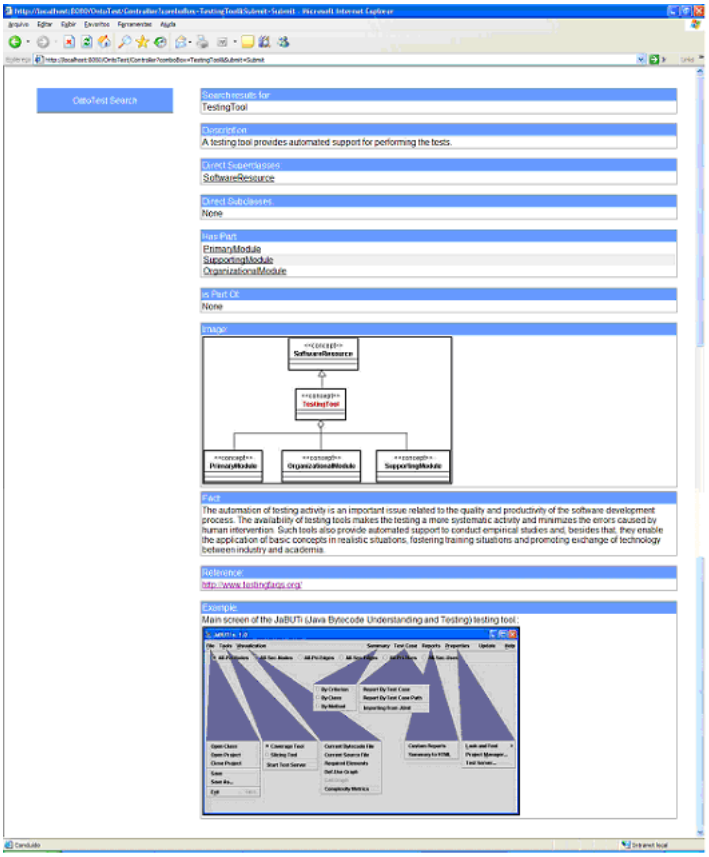

Figure 2. OntoTestSearchingTool

As a short-term goal, we intend to explore *OntoTestSearchingTool* for teaching/learning software testing. In this sense, the *OntoTest*-based application would be explored as part of a testing tool, in particular as an organizational module for training activities.

4.2. An OntoTest-Based Set of Testing Modules

According to the sub-ontologies of *OntoTest*, a testing step can be seen a transformational primitive where step inputs and outputs correspond to testing artifacts. Nevertheless, other elements are necessary to the accomplishment of a testing step. These elements are called testing resources

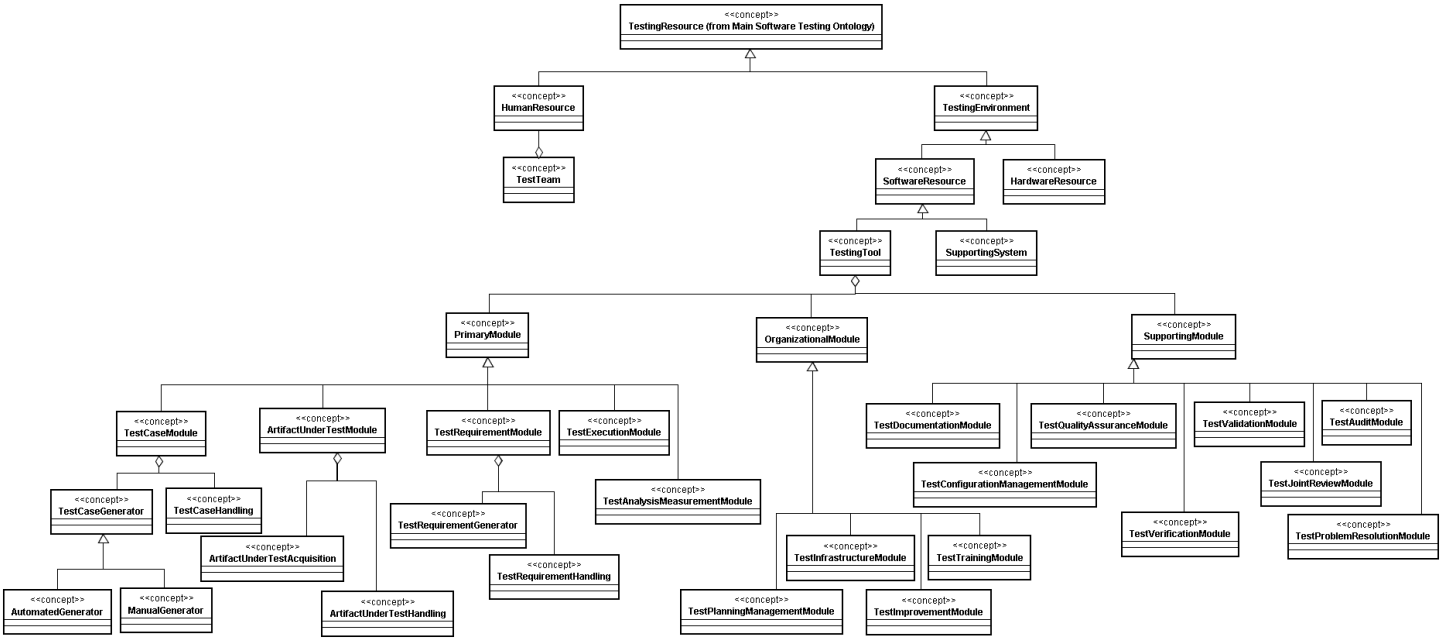

Figure 3. Testing Resource Sub-Ontology

and are considered in the Testing Resource Sub-Ontology (Figure 3).

A testing resource can be a human, a hardware or a software resource. Hardware and software resources are characterized in terms of a testing environment, which can be used to automate the testing strategies and procedures. Testing tools, which provide automated support for performing the tests, are a special kind of software resources. Considering the ISO/IEC 12207 standard, three types of functional modules can be considered as part of a testing tool: (i) primary, (ii) organizational, and (iii) supporting modules.

Based on our experience in testing tools development and on the primary activities of software testing established in the Testing Step Sub-Ontology, we have identified and represented in *OntoTest* a set of primary functional modules that a testing tool should provide:

Test Case Module: Comprises the basic operations involving the test case set (create, view, update, store, remove, import/export, enable/disable and minimize). For instance, test cases can be manually or automatically created, or can be imported from text documents or other testing tools. They can be enabled/disabled in a test session without to be physically removed from the test case database. Also, the test case set can be minimized in order to keep only the most effective test cases.

Artifact Under Test Module: Comprises the basic operations involving the artifact under testing (acquire, analyze, instrument, and store). Depending on the testing approach, an artifact under testing can be a source code, a system requirement, a specification or a design document. In particular, instrumentation is a technique frequently used in software testing for different purposes, (e.g., program and/or specification execution trace, and testing criteria coverage analysis). Instrumenting the artifact to be tested can be divided into two main tasks: (i) deriving the artifact structure; and (ii) inserting statements for collecting runtime/simulation information.

Test Requirement Module: Comprises the basic operations involving the test requirements (establish/generate, execute, enable/disable, mark/unmark requirements as infeasible, and store). Test requirements are the required elements to be exercised during the tests to satisfy a given testing criterion. A requirement is said infeasible if it cannot be exercised by any input data and should be discarded.

Test Execution Module: Responsible for executing the test cases against the artifact under test.

Test Analysis and Measurement Module: Responsible for determining the percentage of satisfaction of the test requirements for a specific criterion by a test case set. It also generates statistical reports (e.g., number of test requirements exercised/not exercised, the most effective test cases, and so on) about the performed tests.

Supporting and organizational modules have been specified similarly. In short, the automated support they provide is in agreement with the processes established by ISO/IEC 12207, adapted to the scope of software testing. As supporting modules, a testing tool should provide mechanisms for automating test documentation and test configuration management, among others. In terms of organizational modules, it should include mechanisms for automating the test

training activities, for instance. It is important to highlight that ontology-based tools, such as *OntoTestSearchingTool*, can also be integrated to testing tools as organizational modules for training.

Aiming at a basis to develop testing tools as well as to systematize its construction in a testing architecture, we have refined the Testing Resource Sub-Ontology, discussing how testing tools to support data-flow [12] and mutation testing [3] can be built based on this sub-ontology. Moreover, this refinement helped us: (i) to validate the concepts and relationships of the Testing Resource Sub-Ontology; and (ii) to investigate how difficult is the establishment of more specific testing ontologies from *OntoTest*.

Basically, we noticed that *Test Case* and *Source Code Under Test* modules remain the same for all testing subdomains. The *Test Requirement Module* corresponds to the *Association Module* for data-flow, and to the *Mutant Module* for mutation:

Association Module: The data-flow test requirements correspond to the associations, i.e., interactions between each variable definition and its subsequent uses (or references) through the program. To derive the associations it is necessary to construct the def-use graph, which represents the definitions and uses of all variables of the program.

Mutant Module: The mutation test requirements corresponds to the mutants, i.e., different versions of the original program, each of which containing a simple syntactic change (fault). The simple faults are modeled by a set of mutant operators applied to a program under testing. A mutant can also be enabled/disabled through a given test session. So, this module is responsible for generating and handling the set of mutants.

Regarding the *Test Execution Module*, it corresponds to the *Instrumented Source Code Execution Module* for data-flow, and to the *Test Execution Module* for mutation:

Instrumented Source Code Execution Module: Executes the test cases against the instrumented program generated by the *Source Code Under Test Module*.

Test Execution Module: For mutation testing, it is necessary to execute the test cases not only against the original program but also against the set of generated mutants. The quality of the test set is measured by its ability to distinguish the behavior of the mutants from the behavior of the original program. The goal is to find a test case that causes a mutant to generate a different output from that of the original program. This kind of mutant is said to be dead. In this sense, the *Test Execution Module* is composed by two other sub-modules: (1) the *Mutant Execution*, responsible for executing the mutants; and (2) the *Source Code Under Test Execution*, responsible for executing the original program.

Finally, the *Test Analysis and Measurement Module* corresponds to the *Data-Flow Analysis and Measurement Module* for data-flow, and to the *Mutation Analysis and Measurement Module* for mutation:

Data-Flow Analysis and Measurement Module: Responsible for providing mechanisms for: (1) determining the infeasible associations (associations which cannot be exercised by any test case); (2) coverage analysis (determining the percentage of satisfaction of the associations for a given data-flow criterion by the test case set); and (3) generating statistical reports about the performed tests.

Mutation Analysis and Measurement Module: Responsible for providing mechanisms for: (1) determining the equivalent mutants – mutants that always generate the same output from that of the original program; (2) calculating the mutation score – the ratio of the number of dead mutants to the number of non-equivalent mutants; and (3) generating statistical reports about the performed tests.

It is also important to notice that the training organizational modules to be considered in data-flow and mutation testing tools can be obtained by instantiating *OntoTest* for each specific knowledge domain. *OntoTest*-based applications, similar to *OntoTestSearchingTool*, can be generated and integrated as training modules into the testing tools.

For the sake of illustration, Figure 4 shows part of the Mutation Testing Resource Sub-Ontology.

Although supporting different testing techniques and criteria, both data-flow and mutation testing tools presented a very similar set of functionalities. This result reassures our expectation on using the Testing Resource Sub-Ontology as a supporting mechanism to define the basic functionalities a testing tool should provide. Actually, from the functional modules defined in the sub-ontology, specifying the core for a testing tool seems to be straightforward. The common set of functional modules for testing tools we have identified should be further investigated as a basis for the establishment of an ontology-based testing architecture.

5. Conclusions and Further Work

In this paper we discussed the using of an ontology of software testing in the development of testing related tools. Two perspectives were explored: (i) the development of an *OntoTest*-based application for sharing and harmonizing testing concepts; and (ii) the refinement of *OntoTest* aiming at investigating how testing tools can be built based on this ontology.

Currently, we are investigating the establishment of an ontology-based architecture to support the development of tools to automate the testing activities. The testing architecture is being built upon the *OntoTest* concepts and relationships and can be refined to different testing domains, supporting the development of specific testing tools. Also, we observed that current testing tools do not present considerable degree of integration, including data integration.

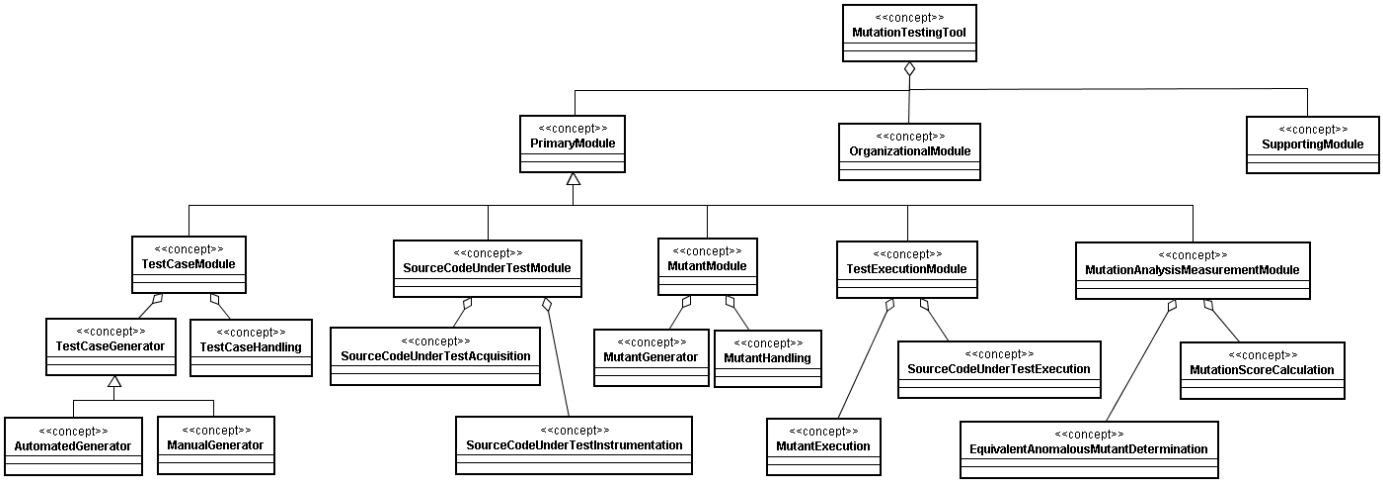

Figure 4. Part of the Mutation Testing Resource Sub-Ontology

On the other hand, intelligent agents have recently been investigated as a mechanism to test software. *OntoTest* should be further explored to enable integration of testing tools and of multiple agents by sharing knowledge, transferring information and negotiating the agent's actions.

Finally, since the body of knowledge on testing keeps evolving, mainly due to the experimental studies conducted, we are also developing an ontology for experiments on software engineering ($_{EXPER}Ontology$ [5]), which should be integrated to *OntoTest*. Both ontologies should be explored in the context of testing reference architectures. At the very end, the idea is to compose, in an incremental and evolutionary way, a software testing environment, integrating tools, processes, artifacts and experimental studies, being useful in promoting reuse of testing expertise and in achieving well-recognized understanding in testing areas. Our experience on using ontologies in the establishment of an architecture-based software testing environment should be presented in a forthcoming paper.

References

[1] T. H. Al Balushi, P. R. F. Sampaio, D. Dabhi, and P. Loucopoulos. Performing requirements elicitation activities supported by quality ontologies. In *18th Int. Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pages 343–348, San Francisco, CA, July 2006.

[2] E.F. Barbosa, E. Y. Nakagawa, and J.C. Maldonado. Towards the establishment of an ontology of software testing. In *18th Int. Conference on Software Engineering and Knowledge Engineering (SEKE 2006)*, pages 522–525, San Francisco, CA, July 2006. Short Paper.

[3] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *IEEE Computer*, 11(4):34–43, April 1978.

[4] R. A. Falbo and G. Bertollo. Establishing a common vocabulary for software organizations understand software processes. In *EDOC Int. Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE 2005)*, Enschede, The Netherlands, September 2005.

[5] R. E. Garcia, E. N. Höhn, E.F. Barbosa, and J.C. Maldonado. An ontology for experiments on software engineering. In *20th Int. Conference on Software Engineering and Knowledge Engineering (SEKE 2008)*, San Francisco, CA, July 2008. To appear.

[6] T. R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. *Int. Journal Human-Computer Studies*, 43(5/6), 1995.

[7] Q. Huo, H. Zhu, and S. Greenwood. A multi-agent software environment for testing web-based applications. In *27th Annual Int. Computer Software and Applications Conference (COMPSAC03)*, 2003.

[8] IEEE Software Engineering Technical Committee. Standard for Software Test Documentation, September 1998.

[9] International Organization for Standardization. ISO/IEC 12207. Information technology – software life-cycle processes, 1995.

[10] B. Kitchenham, G. H. Travassos, and A. Maryhauser. Towards an ontology of software maintenance. *Journal of Sofware Maintenance: Research and Practice*, 11(6):365–389, 1999.

[11] G. J. Myers, Corey Sandler, Tom Badgett, and Todd M. Thomas. *The Art of Software Testing*. John Wiley & Sons, 2nd. edition, 2004.

[12] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Transactions on Software Engineering*, SE-11(4):367–375, April 1985.

[13] M. Uschold and M. Grüninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), June 1996.

[14] K. Villela, G. Santos, L. Schnaider, A. R. Rocha, and G. H. Travassos. The use of an enterprise ontology to support knowledge management in software development environments. *Journal of Brazilian Computer Society*, 11(2):45–59, November 2005. Special Issue on Ontologies Issues and Applications.

[15] P. Wongthongtham, E. Chang, T. S. Dillon, and I. Sommerville. Software engineering ontology – the instance knowledge (part i). *International Journal of Computer Science and Network Security*, 7(2):16–26, February 2007.

[16] P. Wongthongtham, E. Chang, T. S. Dillon, and I. Sommerville. Software engineering ontology – the instance knowledge (part ii). *International Journal of Computer Science and Network Security*, 7(2):27–36, February 2007.

[17] H. Zhu and Q. Huo. Developing a software testing ontology in UML for a software growth environment of web-based applications. In *Software Evolution with UML and XML*. Idea Group, 2004. Hongji Yang (eds.).