PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 09/12

Modeling Relevant Test Concepts from UTPX and SecureUMLX

Andrew Diniz da Costa
Viviane Torres da Silva
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 09/12 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena May, 2012

Modeling Relevant Test Concepts from UTPX and SecureUMLX

Andrew Diniz da Costa, Viviane Torres da Silva[1],
Carlos José Pereira de Lucena
[1] Computer Science Department – Federal Fluminense University
acosta@inf.puc-rio.br, viviane.silva@ic.uff.br, lucena@inf.puc-rio.br

Abstract. Documenting software tests is almost as essential as documenting source code itself. As the recognition of systematic software testing increases, there is a need to conceive modeling techniques to explicitly document key concerns associated with test cases. Modeling techniques are known as a good way to perform such documentation, because they provide abstractions and a visual notation to represent testing-specific concerns facilitating the communication among the members of the project team. Based on this context, the goal of the paper is to present a test conceptual framework able to represent relevant test concepts that are not handled by famous test approaches, such as AGEDIS Modeling Language and UML Testing Profile. In order to represent the test concepts proposed by the conceptual framework, the UTP and SecureUML modeling languages were extended. Such extensions were evaluated from a controlled-experiment based on two industrial large-scale systems and that involved participants with different skills and experience on software testing and modeling.

Keywords: Unified Modeling Language, UML Testing Profile, SecureUML, Test of Software.

Resumo. Documentar testes de software é quase tão essencial como documentar o código fonte em si. Com o reconhecimento da crescente importância de realizar testes de software, há a necessidade de prover técnicas de modelagem para documentar explicitamente as principais preocupações associadas com testes. As técnicas de modelagem são conhecidas como uma boa maneira de realizar essa documentação, já que fornecem abstrações e uma notação visual para representar específicas preocupações de teste, facilitando assim a comunicação entre membros de equipes de teste. Com base neste contexto, este trabalho apresenta um framework conceitual de teste capaz de representar relevantes conceitos de teste que não são manipulados por conhecidas abordagens de teste, como AGEDIS Modeling Language e UML Testing Profile. Visando aplicar esses conceitos propostos no framework conceitual, a UTP e SecureUML foram estendidas. Essas extensões foram avaliadas a partir de um experimento controlado baseado em dois sistemas industriais de larga escala e que envolveram participantes com diferentes habilidades e experiência em modelagem e teste de software.

Palavras-chave: Teste de Software, Modelagem de Teste, Perfil UML.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

1 Introduction

The maintenance and updating of large-scale systems are usually supported by the development of test cases and their documentation. Thus, the documentation of the tests is essential and extremely important to track the tests and the changes performed in each system version. Without documenting the test cases it is difficult to plan the investments necessary to perform the tests and also to guarantee that the remaining faults were not caused due to the changes realized while maintaining or updating the software [Black, 2002][Kaner et al., 1999][Kaner et al., 2001]. One way to perform such documentation is by using test modeling languages that provide abstractions and a visual notation to represent testing-specific concerns.

A test modeling language able to document test cases should provide abstractions and a visual notation to represent testing-specific concerns that will facilitate the communication among the members of the project team about the tests to be executed. Although there are several modeling languages that support the modeling of test cases, such as [OMG, 2007][Trost and Cavarra, 2012] and [UTML, 2012], they are not expressive enough to document the test cases of different projects due to their inability to cover test concepts that are relevant in such kinds of systems [Harrold, 2000][Harrold, 2008]. In addition, there is no published documentation that demonstrates the systematic evaluation of the modeling languages. The available evaluations are basically performed by the development team and do not demonstrate the applicability and efficiency of the language to model important test concepts.

Based on this context, the study presented in [Costa et al., 2010] revealed relevant test concepts that are not often represented in test modeling languages proposed in the literature, such as, (i) identifying which test cases are mandatory or optional to the system under test (SUT), (ii) informing the test type (e.g. functional, unit, performance etc.) of each test case, (iii) relating the version of the SUT to the tests, (iv) representing if a test case is of regression, new or impacted, due new requirements defined to a system, (v) distinguishing components that contain implemented test cases of the components that represent test suites, i.e., entities that execute a set of spread test cases automatically in a specific order.

Another important concern identified in [Costa et al., 2010] was the need of a modeling technique able to associate the test cases with members of a test team that will manipulate such tests. The available modeling languages do not give support to define the ones able to execute, create, delete, update and read the test cases, i.e, they are not able to assign permissions to members of test teams.

Thus, the goal of this paper is to present a test *conceptual framework* that represents these identified test concepts and also their relationships in order to provide the basis to create test-related technologies. Following such approach, the paper proposes the UML Testing Profile [OMG, 2007] eXtended (UTPX) and SecureUML [Lodderstedt et al., 2002] eXtended (SecureUMLX) based on the test concepts proposed in the conceptual framework. Test concepts, often handled by test modeling-approaches, are not mentioned in the *conceptual framework*, such as, data pools used for test cases, execution time of a test, oracle, test's verdict etc.

The paper is organized as follows. Section 2 presents the conceptual framework that represents the test concepts identified in [Costa et al., 2010]. Section 3 presents the use of the UTPX and SecureUMLX in test cases created to a large-scale petroleum control

system tested by a test team of the Software Engineering Lab. Aiming to evaluate the this new modeling approach, Section 4 describes the applied controlled-experiment related to the creation and maintenance of models. Section 5 discusses the test concepts that are used by the most famous test modeling languages and management tools to support the testing activities. And finally, in Section 6, the final considerations are presented.

2 Modeling Test Concepts

This section explains the test conceptual framework (see Figure 1) that represents test concepts revealed in [Costa et al., 2010] and that are not handled by famous test approaches, such as, TTCN-3 [TTCN-3, 2012], UML Testing Profile (UTP) [OMG, 2007] and Agedis [Trost and Cavarra, 2012]. This conceptual framework is presented in two parts: (i) modeling and explanation of the main concepts identified in [Costa et al., 2010], and (ii) modeling of access control policies from the use and extension of SecureUML.

2.1 Modeling Main Revealed Test Concepts

The most important meta-classes of the conceptual framework illustrated in Figure 1 are TestContext and TestCase. TestContext represents a set of tests (one or more test cases, represented by TestCase meta-class) that are related to a specific context. A context is any concept that analysts considered important to conceptually group one or more test cases.

Both test context and test case are concepts often considered by approaches proposed in the literature (e.g, TTCN-3 [TTCN-3, 2012] and UTP [OMG, 2012]). Thus the conceptual framework illustrated in Figure 1 also represents such concepts and the links among them and other related test concepts.

According to [Costa et al., 2010], it is important to specify the system version related to each test case. This information becomes crucial because, depending on the system version, a specific test case can or cannot be executed. Thus, the meta-class SUT is used to represent the current version of a system and the version related to the test context being documented (i.e., the desired version). Note that the test can refer to an antique version of the system.

One of the difficulties met in the projects tested by our teams is related to the identification of the mandatory and optional test case to be executed in each software version. The mandatory tests are those that must be executed in a specific system version, while the optional consists of tests that do not have a very high priority and can be executed if the test team has enough time to do so. This identification helps on planning the test activity. Aiming to represent such idea the enumeration ObligatorinessType was defined.

Another important concept that should also be modeled is the classification of test cases. Such concept is useful for helping in planning the tests and for identifying their importance in particular software releases. Different classifications can be used in a project, such as, regression, new and updated. The first classifies the tests executed every time a new version of the system is created. These tests do not depend on the new or updated requirements that have influenced the creation of the new version. The second classification is composed of new tests created due to the definition of new system

requirements or due to changes to the system functionalities. And the third classification is composed of tests that were modified due to the definition of a new requirement or the modification of a given requirement. These three classifications are examples adopted in different projects tested by the quality group of the Software Engineering Laboratory. Thus, to represent such idea the meta-class TestClassification was defined.

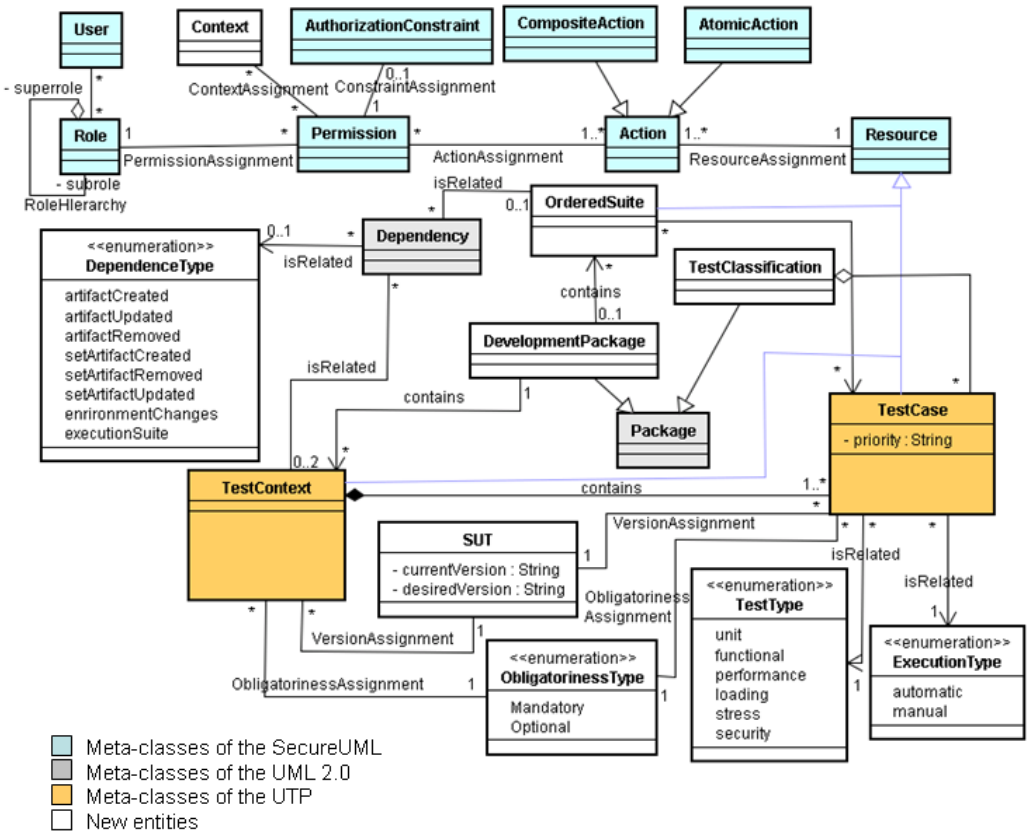

Figure 1. Modeling conceptual framework of test.

The meta-class DevelopmentPackage was represented in order to inform the real package used to develop a given test (especially for test' scripts). It is also useful to distinguish the classifications that group tests conceptually.

The enumeration TestType was defined to help in the identification of test types (e.g. functional, unit, etc.). This identification helps to understand the purpose of each test and is useful on the delegation of tasks to members with expertise in running different test types.

Another test concept represented in Figure 1 is the test cases executed automatically or manually (by using the enumeration ExecutionType). The automatic tests are executed by using a program, while manual tests require the tester to perform a set of manual actions. When a large amount of tests are created, the identification of automated and manual tests is usually time-consuming. Such identification helps the manager on scheduling the testes and guides the tester to know which tool he/she should use to execute the test.

The new attribute *priority* of the meta-class TestCase is used to identify the priority of test execution in more detail. This concept is particularly useful when the available time to test a given project is critical and when the set of mandatory tests is large and

the time to execute them is short. Thus, a subset of mandatory tests having higher priority is executed before the other also mandatory tests.

The meta-class OrderedSuite represents a test suite, which is a code component that executes a set of test cases automatically in a specific order. In the literature different tools and APIs can be used to create suites and execute them, such as Rational Quality Manager [RQM, 2012], Rational Test Manager [RTM, 2012], and DBUnit [DBUnit, 2012]. Thus, to document that a component is a suite and which tests are executed for each suite becomes important in the identification of the artifacts involved in that task. Realize that TestContext aggregates one or more test cases without caring about the execution order.

Another important concept that should be documented is the dependence between tests. This information is considered important in order to define the correct test execution order. To represent it, the Dependency meta-class of the UML 2.0 [Booch et al., 2005] is reused and it is related to enumeration DependenceType that describes the types of dependences between tests. ArtifactCreated, for example, is used when a test case depends on the creation of some artifact (e.g. file, component, entity, etc.) performed by another test case. While artifactRemoved indicates that the test case depends on the exclusion of another artifact of a system. Details of these types are presented in section 3.

2.2 Modeling Access Control Policies

To specify access control policies for test resources, we had decided to use SecureUML [Lodderstedt et al., 2002]. Essentially, it provides a language (see the blue meta-classes in Figure 1) for modeling Roles, Permissions, Actions, Resources, and Authorization Constraints, along with their Assignments, i.e., which permissions are assigned to which roles, which actions are assigned to which permissions, which resources are assigned to which actions, and which constraints are assigned to which permissions. Notice also that actions can be either Atomic or Composite. The atomic actions are intended to map directly onto actual operations of the modeled system. The composite actions are used to hierarchically group more lower-level ones and are used to specify permissions for sets of actions.

Nowadays, SecureUML provides five AtomicActions: update, read, create, delete and execute. Such actions can be related to different types of resource, such as attributes, methods, classes, etc. If new resources will be used from SecureUML, they should extend the meta-class Resource illustrated in Figure 1. Thus, we decided to extend its meta-model in order to specify:

1. The model element types of the system design modeling language that represent protected resources related to test teams. These element types are modeled as specializations of the class Resource. In this case, TestContext, TestCase, and OrderedSuite are protected resources.

2. The actions these resource types offer and the hierarchies classifying these actions (see Figure 2). This is accomplished by introducing the different action types as specializations of the classes CompositeAction and AtomicAction. In Table 1 are presented the actions for each resource of test protected, where underlined actions are composite actions.

3. Depending on the context that a test is executed (e.g. tests executed in the production environment) a user can or can not have permission to perform some

action. Thus, to represent this idea we decided to create the meta-class Context that is related to the meta-class Permission.

Aiming to constrain which resources are accessible from the actions and which actions are subordinated to the composite actions, we used OCL meta-model invariants. Below invariants which guarantee that TestContextFullAccess always act on TestContexts is presented. TestCaseFullAccess and OrderedSuiteFullAccess actions have similar OCL invariants to define that they act on TestCases and OrderedSuites, respectively. Besides, each composite action contains atomic actions (e.g. read, update, create, delete and execute) upon the corresponding artifact.

```
context TestContextFullAccess
 inv targetsATestContext:
 self.resource.ocllsTypeOf(TestContext)
 inv containsSubactions:
 self.subordinatedactions = self.resource.action
->select (a | a.ocllsTypeOf(AtomicRead))
-> union (self.resource.action
-> a | a.ocllsTypeOf(AtomicUpdate)))
-> union (self.resource.action
-> a | a.ocllsTypeOf(AtomicCreate)))
-> union (self.resource.action
-> a | a.ocllsTypeOf(AtomicDelete)))
```

Table 1. Actions of the resources protected

Resource	Actions
TestContext	create, read, update, delete, fullaccess
TestCase	create, read, update, delete, execute, fullaccess
OrderedSuite	create, read, update, delete, execute, fullaccess

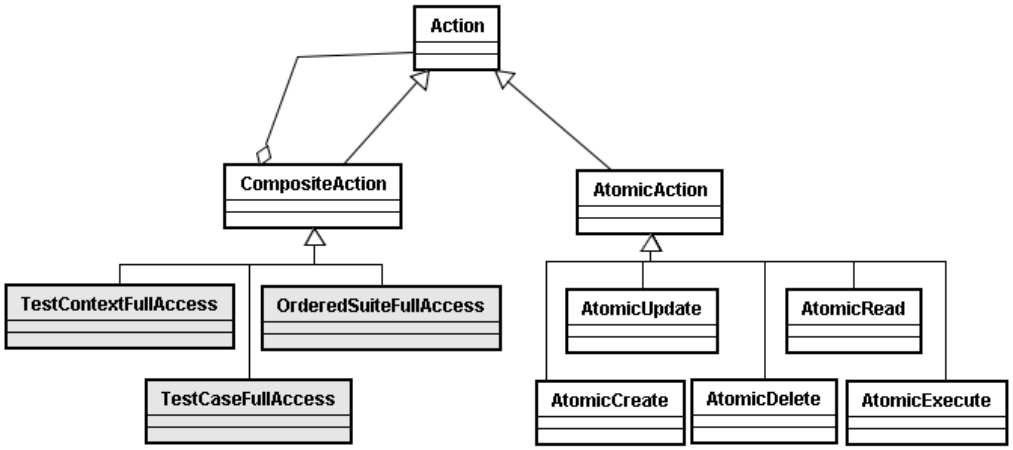

Figure 2. Actions used by SecureUMLX

3 Modeling Test Concepts in an Industrial Domain

This section presents the UML Test Profile eXtended (UTPX) and SecureUML extended (SecureUMLX) based on the conceptual framework presented in Section 2. The main reasons for choosing UTP and SecureUML were the following: (i) they are OMG official standards for modeling, and (ii) they are approaches that represent important concepts related to the test domain (e.g. test contexts, test cases and permissions mentioned in Section 2). UTP is a proposal that was defined by a consortium of institutions (Ericsson, Fraunhofer/FOKUS, IBM/Rational, Motorola, Telelogic, University of Lübeck). Thus, aggregating our experience with the experience of such institutions allowed the creation of a test modeling language more complete and that can be applied in different systems. Aiming to exemplify these extensions, we used a large-scale system tested by a Software Engineering Lab team. Initially, we describe this system and, then, illustrate examples of diagrams created.

3.1 Oil Control System

Over the past five years, the Software Engineering Lab has coordinated and carried out tests of software systems developed for a big Brazilian oil company. Seeking to illustrate the use of the UTPX and SecureUMLX, we decided to choose one of the biggest systems that we tested. This system is responsible for controlling the inventory and supply of oil and derived products (e.g. gasoline, kerosene, etc). Some goals of such systems are: (i) register routes (i.e. paths) based on ducts and ships that could be used to transport the derived products (e.g. gasoline, lubricating oil, kerosene, etc); (ii) predict when such products will arrive in terminals and refineries located in different places; and (iii) plan the best routes to transport a particular product.

The system was developed by four teams responsible for the following elements: interface, database, requirements and test team. The test team was composed of seven people that executed functional, database and performance tests. Table 2 gives some details about the system characteristics.

Table 2. Data of the petroleum control system

Project size	7 years
# of staffs	25 people
Test team	7 people
Status	Working
Model size	49 use cases 712 classes
Data base	213 tables
Test Cases	46 database tests 17 performance tests 120 automated functional tests 600 manual tests

3.2 Modeling Test Concepts from UTPX and SecureUMLX

When we work with profiles in UML, new stereotypes and attributes can be defined. Following this idea, Figure 3 illustrates an example of class diagram that applies UTPX to model these test concepts.

This example models tests created to create and update routes (i.e. paths) based on terminals and refineries that represent places in Brazil that store oil and derived products (e.g. gasoline, lubricating oil, kerosene, etc).

The CreateRoutes class is a test context composed of one test case named testCreation. This test case is considered mandatory and automatic in version 7.0 of the current system (represented from the attribute desiredSystemVersion). Besides test case, two other auxiliary methods are modeled: populateTerminals and populateRefinaries.

These methods are responsible, respectively, for populating a database with terminals and refineries, and they are used by the test case. On other hand, the UpdateRoutes class is a test context composed of three test cases: testUpdate1, testUpdate2 and testUpdate3. The first test case is optional and manual in version 6_00 of the current system. While the second and third test cases are mandatory and manual in version 7.0 of the current system.

Each test case contains three more pieces of important associated information: (i) the system version with which the current test case is associated and updated (described by using the attribute version); (ii) its type of test (e.g. unit, functional, stress, etc.); and (iii) the priority of execution (e.g. high, medium, low) that is related to the obligatoriness type (e.g. mandatory or optional) of the test case.

In order to represent dependences between tests, we decided to reuse the dependence concept proposed by UML. Moreover, we have defined a set of stereotypes to be used related to the dependence relationships in order to provide the semantic of such relations, as described below:

- **<<artifactCreated>>**: It is used when a test case depends on the creation of an artifact (e.g. file, component, entity, etc.) performed by another test case.
- **<<artifactRemoved>>**: It indicates that the test case depends on the exclusion of another artifact of a system.
- **<<artifactUpdated>>**: It states that the test case depends on the actualization of an artifact (e.g. change the name, path, etc.).
- **<<setArtifactCreated>>**: It indicates that the test case depends on the creation of a set of artifacts.
- **<<setArtifactRemoved>>**: It is used when the test case depends on the exclusion of a set of artifacts.
- **<<setArtifactUpdated>>**: A test case depends on the modification of a set of artifacts.
- **<<environmentChanges>>**: The test case depends on changes in the environment where it is executing, such as, changes on the operation system, environment variables, etc.

New types of dependencies can be included depending on the project being tested. Such dependencies were defined from real situations identified in systems tested by the Software Engineering Lab teams. Figure 3 illustrates a dependence used in this project between the tests described in the CreateRoutes and UpdateRoutes classes. Besides showing its dependence, the class diagram informs the semantic of each one.

When a suite describes a specific order for the execution of its tests, the stereotype OrderedSuite can be used. In addition, the dependency relationship is also used between the suite and the test contexts in order to identify the test cases that compose the suite. The stereotype below can be used for this purpose:

<<executionSuite>> [{names of the test cases}]

Figure 3 illustrates an example of the suite SuiteRoutes that depends on the test contexts modeled. The stereotype "<<executionSuite>>[testUpdate2, testUpdate3]" is used in this case to indicate that the suite depends on two test cases of the UpdateRoutes test context. The same structure can also be adopted in order to inform which test cases a test context depends on by using the stereotypes mentioned previously. This information should be included depending on the level of detail that a designer desires to provide.

These test contexts and suite modeled in Figure 3 are grouped in packages that have the stereotype <<Development>>. This stereotype represents that the related package stores the classes or entities created in a given project. On other hand, the stereotype <<TestClassification>>, illustrated in Figure 4, represents categories of a given test context that state the importance of such a test in testing a release.

In the large-scale system being used as an example, three classifications are adopted: regression, new and updated (as explained in details in Section 2.1). The test contexts illustrated in Figure 4 and in the package New were created in order to test the new requirements that were defined in the current version of the system being mod-

eled. Due to the definition of these new requirements, some test context needed to be updated. They are pointed out in Figure 4 in the package Updated. The regression tests are identified in the package Regression.

Figure 5 illustrates an example of the concrete syntax of SecureUMLX. In this example are defined two roles in a test team: Trainee Tester and Expert Tester. The first role can read and execute tests of the OpenScenarium test context in the test environment. On other hand, people that play Expert Tester role have full access to such test context in the test environment. If a role had different permissions for test cases of a same test context, each allowed operation (e.g. read, execute, update, write or delete) would be defined for test case. Thus, in the place of the stereotype <<testcontextaction>> (of the class that describes the permission) should be used the stereotype <<testcaseaction>> followed by the name of the test case and the desired operation desired. This idea of defining permissions also can be applied to ordered suites from the stereotype <<orderedsuiteaction>> followed by the name of the suite and the desired operation.

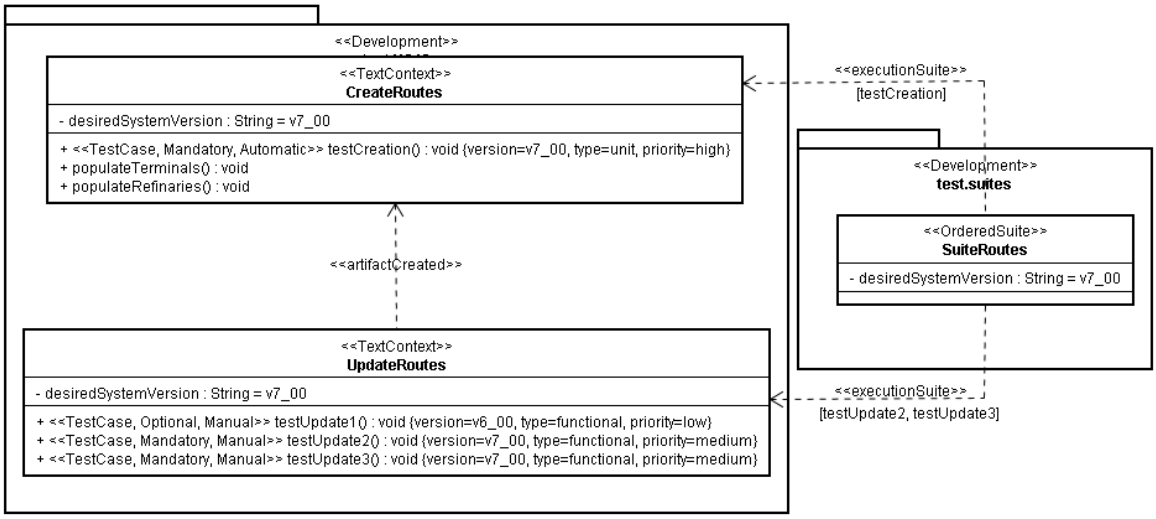

Figure 3. Example of UTPX

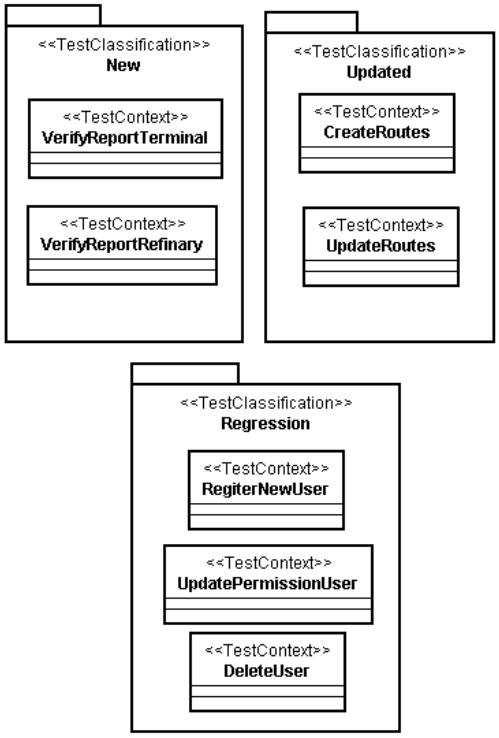

Figure 4. Modeling Test Classifications

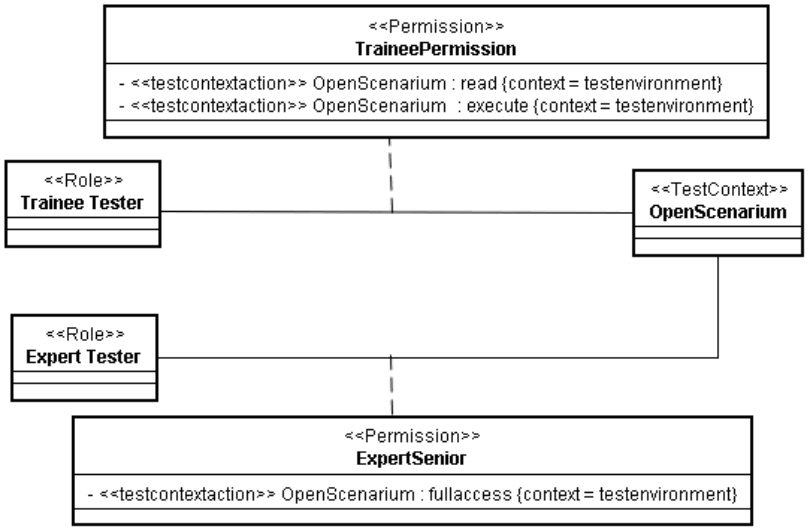

Figure 5. Example of UTPX

4 Evaluation of the Approach

In order to evaluate UTPX and SecureUMLX, presented in Section 3, we decided to conduct a controlled experiment that lasted ten months and was divided in two stages. The focus of the first stage was to analyze hypotheses (H1 and H2) related to the creation of models. While the second stage evaluated hypotheses (H3 and H4) related to the maintenance of models. The considered hypotheses were:

H1: By using UTPX and SecureUMLX the effort to create test models is reduced when compared with the creation based on UTP and UML techniques.

H2: When a designer uses UTPX and SecureUMLX the amount of errors in the creation of test models is reduced when compared with the use of UTP and UML techniques .

H3: UTPX and SecureUMLX reduce the effort in maintaining the test models when comparing with the use of UTP and UML techniques.

H4: When a designer uses UTPX and SecureUMLX the amount of errors in the maintenance test models is reduced when compared with the use of UTP and UML techniques.

Initially, subsection 4.1 explains the empirical procedures followed in stage 1 to apply the experiment that evaluated the hypotheses H1 and H2. Next, in subsection 4.2 the empirical procedures followed in stage 2 to analyze the hypotheses H3 and H4 are presented in detail. At last, in subsection 4.3, the results of the experiments are presented.

4.1 Stage 1: Empirical Procedures to Evaluate Creation of Models

Aiming to analyze the hypotheses H1 and H2, this experiment followed well-known recommendations [Fink, 2003], lasted six months and was applied to forty one participants from two different universities, with different skills and experience (see Table 3). Fourteen subjects were of the Federal University of the Rio de Janeiro and twenty seven subjects of the Pontifical Catholic University of the Rio de Janeiro.

The experiment was divided in six steps. The first step was to train the participants in order to explain: (i) how the subjects could create UML diagrams in UTP, and (ii) how UML allows modeling new concepts using well-known techniques, such as, using commentaries, defining new stereotypes, attributes or methods. Since we had forty one participants, we divided the subjects in six groups. The main reasons to perform this division were: (i) it was hard to book the same time for all subjects to participate on the training, (ii) the use of small groups makes easier the understanding of people's doubts, and (iii) we could divide in small groups the people with similar levels of knowledge on tests and UML. The meeting with each group lasted about forty minutes to one hour and half. This time increased when the knowledge of the participants decreased.

After training, the second step was applied. This step requested to the participants the creation of diagrams using UTP and well-known UML techniques taught in step 1. Such diagrams should model the test concepts (see Section 2) identified in [Costa et al., 2010]. These concepts are related to real situations identified in two large-scale systems of the Software Engineering Lab. These systems are related to the inventory and supply control of petroleum products, being one of them presented in Section 3.1 of the paper.

Table 3. Profile of the subjects that participated of the experiment

Subjects and their Roles	Academic Background	Description
(9 subjects) 1 manager 1 requirement analyst 5 senior developers 2 test analysts	6 PhD candidates in Software Engineering (SE) 1 MSc candidate in SE 1 post graduate as system analyst 1 grad. in Computer Science (CS)	Very good knowledge and experience in testing, UML modeling and developing concepts. > 3 years of experience
(11 subjects) 2 database admins. 1 requirement analyst 3 senior developers 5 testers	1 MSc candidate in SE 5 grad. in CS. 5 undergrad. in CS	Extensively worked with automated and manual functional tests. Good knowledge of UML modeling. 1..3 years of experience
(21 subjects) 21 junior developers	21 undergrad. students in CS	Poor experience with tests and UML modeling, but good knowledge about the main related concepts. < 1 year of experience

In addition, the participants were required to answer a questionnaire and the time they spent in answering was controlled. .The questionnaire was divided in four activities that focused on: (i) the modeling of test concepts related to each test context, such as, the identification of the test cases, their types, obligatoriness, priorities, related system versions system etc; (ii) the dependencies among the test contexts informing the semantic of such dependencies; (iii) the representation of test classifications; and (iv) the modeling of security policies related to the test teams to inform the roles defined for each team.

In order to remove or minimize the bias in the responses given by the subjects, the questions of the experiment was validated by three experts on testing modeling and experimental software engineering. .

The focus of the third step was to prepare the participants to create diagrams using UTPX and SecureUMLX. To perform such training, the same idea of dividing the subjects in groups, applied in step 1, was adopted. Such training lasted from 30 minutes to one hour, and this time changed based on the knowledge of the participants.

The fourth step involved the design of another questionnaire, which requested to the participants the modeling of the test concepts mentioned in Section 2 by using UTPX and SecureUMLX. Such questionnaire was also validated by three experts on

modeling tests and experimental software engineering and followed well-known recommendations [Fink, 2003]. Besides, these activities were equivalent to the first questionnaire, i.e., they had the same complexity and followed the same structured explained in step 2. Thus, the first activity of the questionnaire requested a similar modeling of test contexts modeled in Figure 3. Second activity requested a modeling with dependencies, as shown in Figure 3, involving suites and test contexts. Third activity asked the creation of a class diagram considering test classifications (similar to the diagram in Figure 4). And the fourth activity requested the modeling of security policies, as shown in Figure 5.

Both the questionnaires applied in the second and fourth steps and the people training were applied in different orders, i.e., part of the subjects responded the questionnaire of the second step followed by questionnaire of the fourth step, and another part followed the inverse procedure. The reason behind it was to avoid actions that could induce the result of the experiment.

Next, interviews were performed with each participant in the fifth step. The main idea of this step was to understand the problems that each subject had when modeling the requested test concepts and the difficulties they found in using the new approaches (UTPX and SecureUMLX). According to the participants answering the UTPX questionnaire was easier than the UTP questionnaire. One of the main reasons was that UTP does not handle several test concepts requested in the questionnaire. Besides, all participants mentioned that UTPX and SecureUMLX is an easy and intuitive approach to work with. Examples of phrases mentioned by the participants are presented below.

> "UTPX and SecureUMLX is an intuitive and easy approach to create class diagrams composed of test concepts."
>
> "I did not have any problem on creating models based on UTPX and SecureUMLX."

Finally, the last and sixty step gathered and analyzed in detail all the responses of the questionnaires responded by the participant. These results are mentioned in details in subsection 4.3.

4.2 Stage 2: Empirical Procedures to Evaluate Maintanance of Models

This part of the experiment analyzed the hypotheses H3 and H4 that lasted four months and was applied to sixteen participants with different skills and experience (see Table 4) of the Pontifical Catholic University of the Rio de Janeiro.

Similar to the stage 1 presented in section 4.1, well-known recommendations [Fink, 2003] were followed and six steps were also adopted in this second stage of the experiment. Above, the main differences between this stage and the stage presented previously are presented.

Table 4. Profile of the subjects that participated of the experiment

Subjects and their Roles	Academic Background	Description
(4 subjects) 1 manager 1 requirement analyst 2 senior developer	4 PhD candidates in Software Engineering (SE)	Very good knowledge and experience with testing, UML modeling and developing concepts. > 3 years of experience
(10 subjects) 2 database admins. 3 senior developer 5 testers	5 MSc candidate in SE 3 grad. in Computer Science (CS). 2 undergrad. in CS	Extensively worked with automated and manual functional tests. Good knowledge of UML modeling. 1..3 years of experience
(2 subjects) 2 junior developers	2 undergrad. students in CS	Poor experience with tests and UML modeling, but good knowledge about the main related concepts. < 1 year of experience

The first step concerns about the training of the participants in order to explain how the subjects could maintain the UML diagrams created by using UTP and well-known UML techniques. Since we had sixteen participants, we divided the subjects in four groups. The meeting with of each group lasted from thirty minutes to one hour. This time vary according to the knowledge of the participants. Realize that the time spent time in this stage was lower than the time spent in the first step of the experiment related to the creation of models (subsection 4.1). The main reason was that fifteen of the sixteen subjects participated on the previous experiment.

After training, the second step was applied. This step requested to the participants the maintenance of class diagrams created by using UTP and well-known UML techniques presented in step 1. The structure of this questionnaire was similar to the structure defined in step 2 of the previous experiment that was composed of four activities.

The third step prepared the participants to maintain diagrams created by using UTPX and SecureUMLX. The subjects were divided in four groups, and this activity lasted from 20 to 40 minutes.

The fourth step involved the design of another questionnaire, which requested to the participants the maintenance of class diagrams based on UTPX and SecureUMLX. This questionnaire followed the same structured and complexity adopted by the questionnaire of step 2.

The questionnaires of the steps 2 and 4 were validated by three experts on testing modeling and experimental software engineering and followed well-known recom-

mendations [Fink, 2003]. Besides, similar to stage 1 of the experiment, such both questionnaires and the people trainings were applied in different orders. The reason behind it was to avoid actions that could induce the result of the experiment.

Next, interviews were performed with each participant in the fifth step. All participants mentioned that they did not have problems to maintain the diagrams based on UTPX and SecureUMLX. However, some participants mentioned that depending on the amount of classes modeled, the maintenance can be a problem. Next, there are some phrases mentioned by the participants:

> "Maintaining diagrams based on UTPX and SecureUMLX was easy, but depending on the amount of classes can be a problem."
>
> "UTPX and SecureUMLX together is good approach to model test concepts. However, an evaluation involving a bigger amount of classes should be performed."

The problem of maintaining big amounts of diagrams and classes modeled in the same diagram is well-known in the literature. Nevertheless, aiming to identify the impact of the new approaches in such situation, we intend to apply them in real test projects to perform better analyses.

Finally, the last and sixth step gathered and analyzed in detail all the responses of the questionnaires responded by the participant. These results are mentioned in details in subsection 4.3.

4.3 Results of the Experiment

In Figure 6 are illustrated the results obtained in stage 1 of the experiment described in subsection 4.1. Figure 6(a) shows that 95.63% of the participants responded the second questionnaire (step 4) in less than 50 minutes, while only 43.91% of the participants spent the same time in the first questionnaire (step 2). Another data presented in Figure 6(a) is that 39.02% of the participants took more than 60 minutes to respond the first questionnaire, while in the second questionnaire no participant spent such time.

Figure 6(b) presents the amount of errors modeled by the participants on each applied questionnaire. In the second questionnaire 21.95% of the participants modeled correctly 100% of the exercises requested. On other hand all the participants had some problem when responding the first questionnaire. Besides, 90.24% had more than ten modeling errors in the first questionnaire. While in the second questionnaire there was no participant that made more than ten errors.

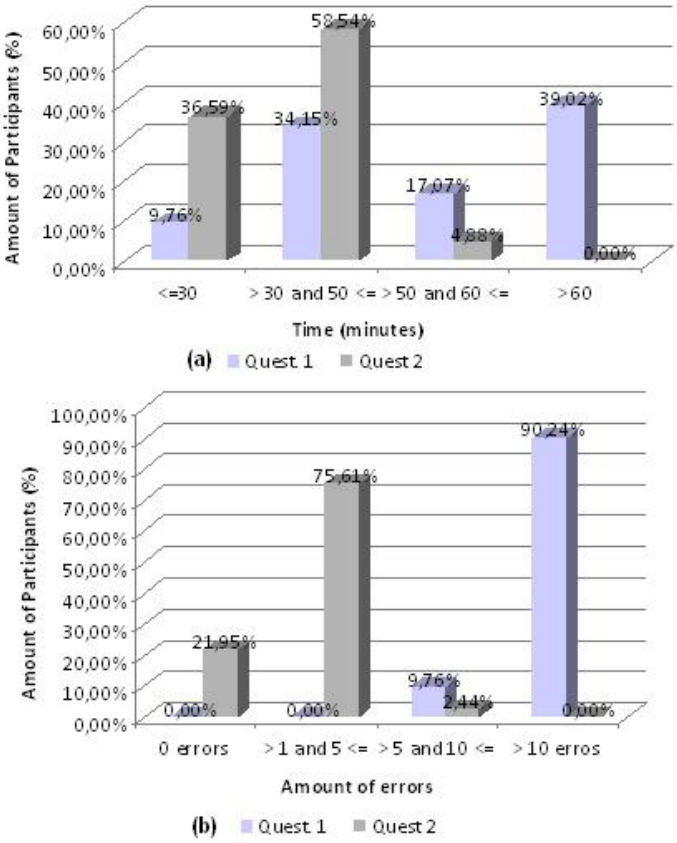

Figure 6. Results of experiment – stage 1.

In Figure 7 are illustrated the results obtained in the stage 2 of the experiment described in subsection 4.2. Figure 7(a) shows that 93.75% of the participants responded the second questionnaire (step 4) in less than 40 minutes, while in the first questionnaire (step 2) there were 87.50% of the participants. Another data presented in Figure 7(a) is that 12.50% of the participants took more than 40 minutes to respond the first questionnaire, while in the second questionnaire there were only 6.25% of the participants.

Figure 7(b) presents the amount of mistakes made by the participants on each applied questionnaire of the stage 2 of the experiment. In the second questionnaire 18.75% of the participants modeled correctly 100% of the requested activities. On other hand all the participants had some problem when responded the first questionnaire. Besides, 25.00% made more than ten modeling mistakes in the first questionnaire. While in the second questionnaire there was no participant with more than ten errors.

By analyzing these data we could conclude that the hypotheses H1, H2 and H3 are true, because the time and the amount of errors modeled were substantially lower in the second questionnaire than the first one. However, the experiment was not conclusive in case of hypothesis H4 because the gathered results did not present relevant differences when comparing both questionnaires. Thus, an analysis involving more participants is being elaborated to achieve a conclusion about H4.

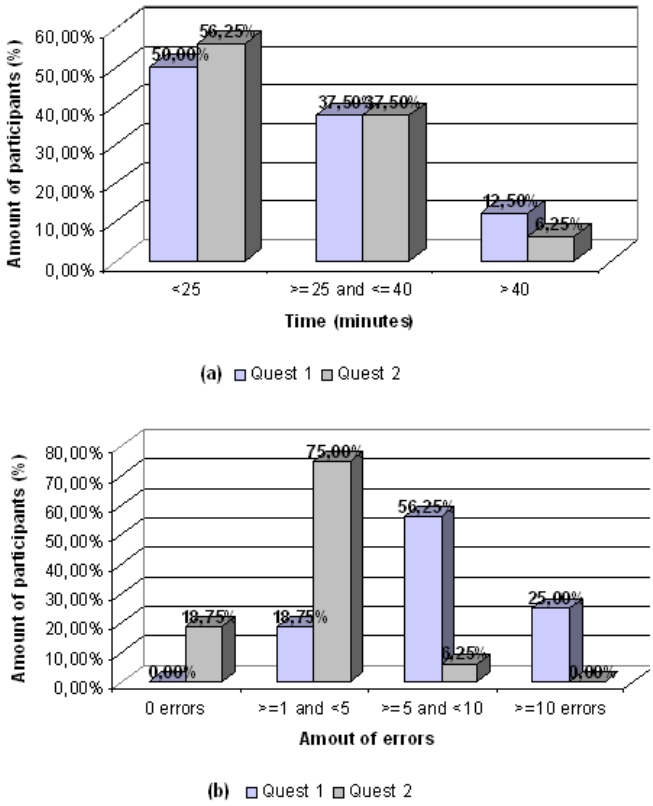

Figure 7. Results of experiment – stage2.

5 Discussion

After analyzing a set of famous test modeling languages proposed in the literature, such as UML Testing Profile [OMG, 2007], AGEDIS modeling language (AML)[Trost et al., 2012], Testing and Test Control Notation (TTCN-3) [TTCN-3, 2012] and Unified Test Modeling Language for Pattern-Oriented Test Design (UTML) [UTML, 2012], we verified that they do not provide important concepts that can be useful for test teams, such as the identification of (i) which system version each test is able to test, (ii) which tests are mandatory, (iii) which test types were created (e.g. functional, database, security and integration test), (iv) which types of dependences exist between tests (such as data dependence), and (v) which tests are automated and manual.

Besides, several proposed tools, which manage test cases (e.g. Rational Test Manager - RTM [RTM, 2012] and Rational Quality Manager - RQM [RQM, 2012]), provide interesting test views from the interface that are not provided by the other mentioned approaches, such as the ability to group conceptually test cases. This grouping becomes easier their identification. Besides this, they allow viewing which suites are available, and which test cases each suite executes. On the other hand, the tool does not provide important test concepts, such as: (i) dependences between tests; (ii) the identification of which tests are mandatory and optional; (iii) which tests are automated and manual; (iv) the identification of test types; and (v) which tests are updated to a given version.

Aiming to provide an approach that could treat this gap, we decided to propose a new test modeling language that could represent these forgot test concepts. From this new modeling, we identified the necessity of defining security policies for test teams.

Thus, we decided to use SecureUML [Lodderstedt et al., 2002] that is based on role based access control (RBAC). According [Basin et al., 2007], a distinct characteristic of SecureUML is that it spells out and follows a precise methodology that guarantees that query evaluation is formally meaningful. This methodology requires, in particular, precise definitions of both the metamodel of the modeling language and the mapping from models and scenarios to the corresponding snapshots of this metamodel. From this idea, we decided to bring SecureUML to the test domain. Thus, security policies could be defined on test concepts not supported for famous approaches of the literature.

6 Conclusion and Future Work

This paper presented a test conceptual framework that aims to represent a set of test-relevant concepts not supported by current modeling languages proposed in the literature, as is clearly stated in Section 2. This framework was created based on our five years experience in coordinating and carrying out tests in different large scale projects within the scope of the Software Engineering Lab.

The paper also presented extensions in the UML Testing Profile and SecureUML in order to represent these test concepts not handled. Aiming to evaluate such extensions, a controlled-experiment divided in two stages and composed of subjects with different levels of skills on software testing and modeling was applied.

From good results of the experiment, we have been developing a tool that allows generating test scripts and other useful artifacts (e.g. reports) for test teams based on UTPX and SecureUMLX diagrams. Such tool will be a plug-in to the Rational Software Architect [RSA, 2012], which is an IDE tool used in development projects of the Software Engineering lab and allows modeling different types of diagram, such as UML. From this plug-in test teams of the lab will adopt the Model Driven Test paradigm in order to create class diagrams that are documentations requested by our clients. Based on this motivation, the lab created a group responsible for developing plug-ins to tools used in different projects.

References

Basin, B., D., Clavel, J., Doser: "A Metamodel-Based Approach for Analyzing Security-Design Models, in the Proceedings of the ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems, 2007.

Black, R.: Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, Publisher: Wiley, 2nd edition, ISBN: 0471223980, 2002.

Booch, G., Rumbaugh, J., and Jacobson, I.: Unified Modeling Language User Guide, 2nd Edition, The Addison-Wesley Object Technology Series, 2005.

Costa, A. D., Silva, V. T., Garcia, A., Lucena, C. J. P.: "Improving Test Models for Large Scale Industrial Systems: An Inquisitive Study", Proceedings of the ACM/IEEE 13th International Conference on Model Driven Engineering Languages and Systems, Part I, LNCS Springer 6394, pp. 301-315, Oslo, Norway, 2010.

DBUnit web site, http://www.dbunit.org/. Último acesso realziado em Maio de 2012.

Fink, A.: The Survey Kit: How to ask survey questions, Volume 2, Publisher: SAGE, ISBN 0761925791, 2003.

Harrold, M. J.: Testing: A Roadmap. Proceedings of ICSE 2000, Future of Software Engineering, pp. 61-72, 2000.

Harrold, M. J.: Testing Evolving Software: Current Practice and Future Promise. Proceedings of ISEC 2008, pp. 3-4, 2008.

IEEE-SA Standards Board, IEEE Standard for Software Test Documentation http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=741968&userType=inst

Kaner, C., Bach, J., Pettichord, B.: Lessons Learned in Software Testing, Publisher: Wiley,1st edition, ISBN: 0471081124, 2001.

Kaner, C., Falk, J., Nguyen, H. Q.: Testing Computer Software, Publisher: Wiley, 2nd edition, ISBN: 0471358460, 1999.

Lodderstedt, T., Basin, D., Doser, J.: "SecureUML: A UML-Based Modelling Language for Model-Driven Security", in the Proceedings of the 5th International Conference on the Unified Modeling Language, 2002.

RQM – Rational Quality Manager, http://www-01.ibm.com/software/rational/products/rqm/,. Último acesso realziado em Maio de 2012.

RSQ - Rational Software Architect, http://www.ibm.com/developerworks/rational/products/rsa/. Último acesso realziado em Maio de 2012.

RTM - Rational TestManager and Rational ManualTest, http://www-01.ibm.com/software/awdtools/test/manager/. Último acesso realziado em Maio de 2012.

OMG - Object Management Group, UML Testing Profile, version 1, http://www.omg.org/cgi-bin/doc?formal/05-07-07, 2007.

Trost, J., and Cavarra, A. AGEDIS Language Specification, http://www.agedis.de/documents/d127_1/AGEDIS-ls-fpd.pdf. Último acesso realziado em Maio de 2012.

TTCN-3 web site, http://www.ttcn3.org/. Último acesso realziado em Maio de 2012.

UTML - The Unified Test Modeling Language for Pattern-Oriented Test Design, http://www.fokus.fraunhofer.de/en/motion/ueber_motion/technologien/utml/index.html21. Último acesso realziado em Maio de 2012.