Chapter 7

ALAMEDA Ecosystem: Centering Efforts in Software Testing Development

José González Enríquez,
Julián Alberto García-García,
Francisco José Domínguez-Mayo and
María José Escalona Cuaresma

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/66043

Abstract

One of the most important and critical aspects to improve the quality assurance in software is to improve the testing process by utilizing techniques and tools, which will enhance the software testing process, making it more effective and efficient. This chapter presents ALAMEDA ecosystem, a software package that centers its efforts in software testing development and is a result from a real-world project. ALAMEDA provides support to lifecycles focused on the generation, implementation, and testing organization from the earliest stages of software development. In addition, the ecosystem provides an environment of rating the degree of compliance of organizations with the International Standard for Testing ISO/IEC-29119. It is proposed as a tool to use during the various iterations that may occur in an agile software development process.

Keywords: testing development, model-driven testing, quality assurance

1. Introduction

The area of Information Technology and Communications (ICT) is an essential element of innovation in other business sectors such as industrial, logistics, health, etc., where the effectiveness and efficiency in software development is critical for a fundamental and safe operation.

The process of software development involves a series of activities in which the chances of a human error is high (mistakes can happen in the beginning of the process, in which the objectives can be inadequately specified, as well as during later steps). Therefore, it is important to research and define techniques and methods (as well as supporting tools) in order to detect and solve any area of special attention as soon as possible. In this sense, over recent decades, a number of studies have analyzed the cost of resolving software defects. The cost increases with the time of the defect in the system [1]. As such, the cost of rectifying an error is lower, the earlier the problem is detected.

The budget of software projects is critical and limited in many cases [2]. For this reason, both research institutions and enterprises look for solutions to improve quality assurance of the software development process [3]. The finding of methods and tools to improve the quality, reduce costs and increase the guarantee of results becomes an essential aim for enterprises and development teams [4].

One of the most important and critical aspects to improve the quality assurance in software is to improve the testing process by utilizing techniques and tools, which will enhance the software-testing process, making it more robust. It is a systematic method to determine inter-related factors and parameters affecting a process and its desired output [5]. It is usually performed for one of two reasons: to detect defects and to estimate reliability. The key to software testing is trying to find the modes of failure—something that requires testing the code on all possible inputs and depending on the environment it can be implemented at any time during the development process [6].

This chapter presents a testing solution (named ALAMEDA) to innovate in all these aspects related to software testing. ALAMEDA offers to project managers and developers a systematic and tool-based approach for managing (i.e., defining, controlling, measure, etc.) test cases. This systematic and automatic management allows the detection and correction of software errors before they become costly interruptions in the production line.

The ALAMEDA ecosystem aims to provide a set of applications which allows software development companies to ensure the management, automation, and integration of test cases from an early stage of the develop lifecycle. For this purpose, ALAMEDA integrates the well-known Model-Driven Engineering (MDE) [7, 8] paradigm on software testing and its automation of software and testing development by use of the NDT (Navigational Development Techniques) methodology [9] and NDT-Suite [10].

MDE has been used as a direction vector of the ALAMEDA ecosystem because it is one of most entrenched paradigms within the software engineering area. As such, on the one hand it will help to define the approach and on the other hand, it will manage the conceptual complexity the process engineering area entails. In addition, MDE has been also successfully applied to real software environments and it has shown a considerable area of impact on reducing time-to-market and improving software product quality. For instance, in the business process management [11], healthcare, software product lines [12], or web engineering among others.

The remainder of this chapter is organized as follows: Section 2 presents the theoretical foundations of ALAMEDA. Section 3 presents the ALAMEDA ecosystem for software testing. Finally, Sections 4 and 5 briefly discuss some related work and highlight conclusions.

2. Theoretical foundations of ALAMEDA

The model-driven software development has become one of the most important paradigm in computer engineering. Indeed, it has shown a considerable impact on reducing time-to-market and has improved overall quality.

However, the development of high-quality software solutions does not only require systematic processes of software development, but it also requires systematic processes of software testing. At present, this latter aspect has not been attracting attention within the software industry.

It is a fact that if software companies do no invest resources and efforts in the process of software testing, it is quite likely that there will be a waste of money in the short term because of a large volume of errors. Therefore, if a systematic process is not used in tasks of testing, software developers will spend most of the time to check quality and application functionality rather than developing new projects [13].

The ALAMEDA ecosystem aims to innovate in the systematic and automatic application of the process of software testing in the real environment. For this purpose, the ecosystem presented herein takes into account the new standard for software testing (ISO/IEC/IEEE 29119) and a well-validated methodology named NDT. The latter proposes techniques and mechanisms to generate test cases from the early stages of the software lifecycle. These theoretical foundations are explained in the next sections.

2.1. ISO/IEC/IEEE 29119 software testing

The International Software Testing Standard (ISO/IEC/IEEE 29119) [14] is a standard created between 2013 and 2015. It was created in order to unify all knowledge on software testing and establish basic knowledge concerning this discipline.

On the one hand, this standard provides useful methodological and conceptual guidelines from the perspective of professional or software providers. In addition, it provides (offers) a common terminology, certifications, well-defined techniques, professional qualification, continuous improvement of the process of software testing, inter-operability, and consistency, among others.

On the other hand, and from the perspective of customers, ISO/IEC/IEEE 29119 provides customer confidence on their software provider, an industry benchmark for good practice, and a contractual relationship between the customer and the software provider, among others.

ISO/IEC/IEEE 29119 focuses on a process model of three levels based on the risk factor in the software-testing phase. This model provides guidelines on strategies and policies so as to manage the process of software testing (i.e., defining testing plans, monitoring and controlling software tests, establishing the test environment, executing tests, etc.). All these aspects have been described in detail in five volumes, as follows:

- ISO/IEC 29119-1: Concepts & Definitions. It introduces the vocabulary on which all standards in the 29119 series are built. It also provides examples of the application of each concept in practice.
- ISO/IEC 29119-2: Test Processes. It defines a generic process model for software testing that can be used to govern, manage, and implement software testing in any organisation, project, or testing activity. This model is based on three layers: (i) organizational test specifications (e.g., organizational test policy, organizational test strategy); (ii) test management; and (iii) dynamic testing.
- ISO/IEC 29119-3: Test Documentation. It defines templates for test documentation that cover the entire software testing life cycle.
- ISO/IEC 29119-4: Test Techniques. It defines techniques and methods to design test cases.
- ISO/IEC 29119-5: Keyword Driven Testing. It defines guidelines for supporting keyword-driven testing which is a way of describing test cases by using a pre-defined set of keywords. These keywords are names which are associated with a set of actions that are required to perform a specific step in a test case. By using keywords to describe test steps instead of natural language, test cases can be made easier to understand, to maintain, and to automate.

2.2. Model-driven engineering on software testing

The model-driven engineering paradigm [15] came up in order to tackle the complexity of platforms and the inability of third generation languages to relieve this complexity. It effectively expresses the domain concepts of the problem. This new paradigm, apart from raising the level of abstraction [16], intends to increase automation during the life cycle of software development.

MDE works, as the primary form of expression, with definitions of models (which can be defined, e.g., by means of Unified Modeling Language (UML) [17]) and transformations (which can be defined, e.g., by means of Query/View/Transformation (QVT) [18]) among these models which entail the production of other models. Both elements are essentials to apply MDE. In addition, it is important to take into account the concepts of metamodel and transformation rules. On the one hand, a metamodel is composed of a set of basic elements, its relationships, and its semantic constraints to build well-defined models. On the other hand, a transformation between two models defines a set of relationships between elements of metamodel A (source) and elements of metamodel B (target) [19].

Although MDE offers a theoretical framework, it has not been standardized exactly in these terms in order to facilitate the application of MDE in real projects. OMG presents MDA, which stands for model-driven architecture [20], as a platform to support the MDE paradigm. MDA proposes to base the software development on models which make transformations in order to generate a code or another model with characteristics of a particular technology (or lowest level of abstraction). As transformations go on, it may be noted that the models become more concrete and the abstract model changes into another one compatible with a particular technology or platform. MDA is based on four types of levels or models. These models are (of highest to lowest level of abstraction):

- The CIM (computation-independent model) level is considered the highest level of business model and is associated with the most abstract level. It focuses on requirement specification and regards that anyone who knows the business and its processes can understand a CIM model, as this avoids any contact with a specific system.
- The PIM (platform-independent model) level represents the business process model and system structure, without any reference to the platform on which the application will be implemented. It is usually the entry point for all the support tools for MDA.
- The PSM (platform-specific model) level explicitly relates to the platform on which the system will be implemented, for example, with operating systems, programming languages, or middleware platforms, among others.
- The Code level refers to the codification and suitable implementation of the system.

MDE in general (or MDA in particular) has been successfully applied in different environments and research areas [21], software testing is one of them. In fact, this particular case is named as model-driven testing (MDT).

MDT is becoming more standardized in order to approach the automation of software testing, thanks to different research works and proposals [22–24]. This approach can significantly reduce the most painstaking cycle of all software development efforts—testing. Testing currently comprises between 30 and 70% of all software development projects, which calls for a significant amount [25] of resources that must be properly managed. Therefore, MDT and supporting tools enable software developers and testers to become far more productive and reduce the time-to-market, while maintaining high standards of software quality.

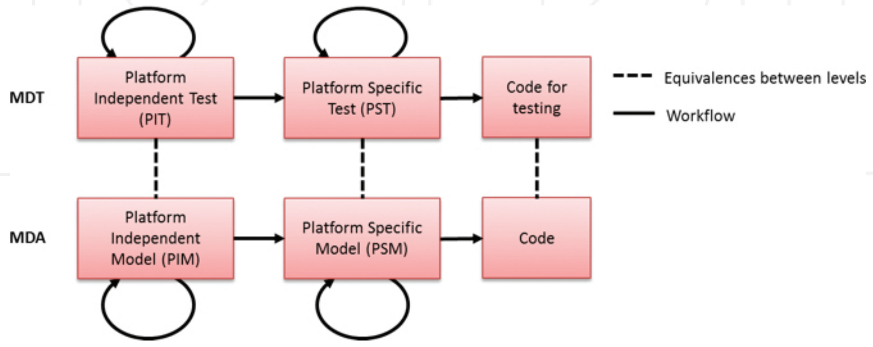

MDT
Platform Independent Test (PIT)
Platform Specific Test (PST)
Code for testing
Equivalences between levels
Workflow
MDA
Platform Independent Model (PIM)
Platform Specific Model (PSM)
Code

Figure 1. General scheme for generation of test cases integrated in ALAMEDA.

Figure 1 shows the vision of MDT from the perspective of MDA. As this suggests, models belonging to the PIM level can be transformed into models belonging to the PSM level, getting the code from the systems derived from PSM through another transformation step. This is the general philosophy of work proposed by MDA, but if this one is applied to software testing, some concepts change. In this case, the process of testing starts with models of System's Test which belong to the PIM level; or also named platform independent test (PIT). The quality team will elaborate on these tests to provide additional information. Next the PSM, which contains specific information about the implementation of the system, can be translated into the Platform Specific Test (PST), using the previous information from PIT and the knowledge of the quality team. Finally, these tests are morphed into codes for testing the system, ensuring the quality of the test and covering all the project-specific requirements.

All these theoretical foundations of MDT have been systematically applied and adapted within the ALAMEDA ecosystem. For this purpose, ALAMEDA implements a methodological framework to generate case tests from functional requirements (FR) as well as managing and controlling these case tests. This framework is based in NDT, a well-contrasted methodology. The foundations of NDT and how this one has been integrated into ALAMEDA are detailed in Section 2.3.

2.3. The NDT methodology from the perspective of software testing

NDT is a web methodology and is covered by the MDE paradigm. At present, NDT supports classical and agile software development lifecycles. For this purpose, NDT defines a set of metamodels for each phase of the software lifecycle (viability study, requirements, analysis, design, implementation, testing and maintenance) and it establishes a set of transformation rules (based on QVT) to generate models from other ones systematically. This implies a lower cost for the software development because of the automation of the process.

The application of MDE-based methodologies (such as NDT) and particularly, the application of transformations among models may become monotonous and very expensive if there are no supporting tools. These tools automate the process in order to get all the potential of MDE which provides a practical and useful environment to the enterprises. This aspect is one of the virtues of NDT by which this methodology has been applied successfully in many real projects. In this sense, NDT defines a set of supporting tools grouped in NDT-Suite [26]. The main tools in NDT-Suite are: (i) NDT-Profile, which defines UML profiles for each NDT metamodel; (ii) NDT-Quality [27], which measures the quality of use of NDT and checks semantic constraints of NDT; and (iii) NDT-driver [28], which allows running each QVT transformations between models in an automatic manner (for instance, this tool generates test cases from the specification of FRs of a project).

From the perspective of the process of software testing, ALAMEDA uses mechanisms defined by NDT to generate test cases from functional requirements defined in the first phase of the software lifecycle (i.e., the requirement phase). ALAMEDA also proposes how functional testing can be deeply improved by means of *early testing*. This paper aims to briefly present how this mechanism of generation of test cases works [29, 30].

As mentioned above, ALAMEDA implements the concept of *early testing* which means that test cases can be generated from early stages of the software lifecycle. It does not need to define test cases when the system is developed (classic software development lifecycle). In this sense, the generation of test cases of ALAMEDA is based on a four-step process (**Figure 2**):

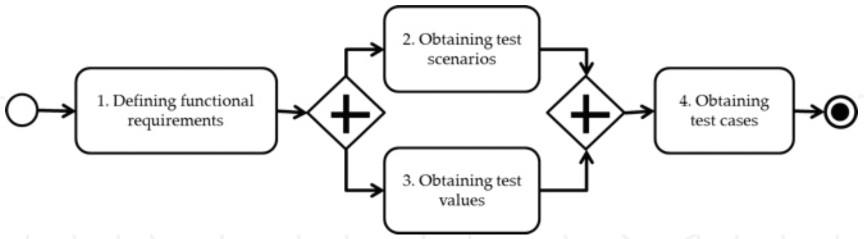

1. Defining functional requirements
2. Obtaining test scenarios
3. Obtaining test values
4. Obtaining test cases

Figure 2. Test generation of ALAMEDA ecosystem.

1. Defining functional requirements. In this step, the user defines the functional requirements of his/her system. For this purpose, the user should use the concrete syntax of the FR metamodel of NDT. This definition is performed using NDT-profile (which has been integrated in ALAMEDA).

2. Obtaining test scenarios. In this stage, ALAMEDA manages each FR as a graph or a state machine. Taking into account this consideration, the methodology applies classic algorithm of path finding in a state machine in order to generate paths. Each path will be a scenario designed together with the system. At the same time, each scenario is a potential test case for assessing the right implementation and functionality thereof.

3. Obtaining test values. This step of the process consists of applying the category-partition method [31] to FRs. This method is based on identifying categories and partitions and to then generate combinations among such partitions. In the context of FRs, a category is any point for which the FRs defines an alternative behavior. Once all categories and partitions are identified, a combination between them becomes a potential test case.

4. Obtaining test cases. Once generated each scenario and possible values, ALAMEDA combines both results to generate final test cases. This step and the previous are systematically generated thanks to the QVT transformation rules defined by NDT. In addition, these transformations are performed using NDT-profile (which has been integrated in ALAMEDA).

3. ALAMEDA ecosystem

The ALAMEDA ecosystem for software testing consists of:

- a methodological framework based on the ISO 29119 International Testing Standard and

- a technological support that allows carrying out of the above methodological framework and is composed of a web platform and a desktop platform. It is worth mentioning at this point that both allow the application of MDT and the automation of development tests by the use of NDT-Suite, respectively. In addition to this, they incorporate a set of tools to automate the dynamic tests and perform the management and quality of their software in an integrated way.

Concretely, the ALAMEDA ecosystem offers a platform as a service (PaaS) ready-to-use, a quick and a progressive methodological implementation and adaptation. It is also flexible and easily scalable and offers an alignment with the ISO 29119 International Testing Standard.

3.1. Methodological framework

In this section, the different actors and activities involved in the ecosystem are shown. The different cases performed by an actor are also described.

Figure 3 shows the three main types of actors who inherit different subtypes.

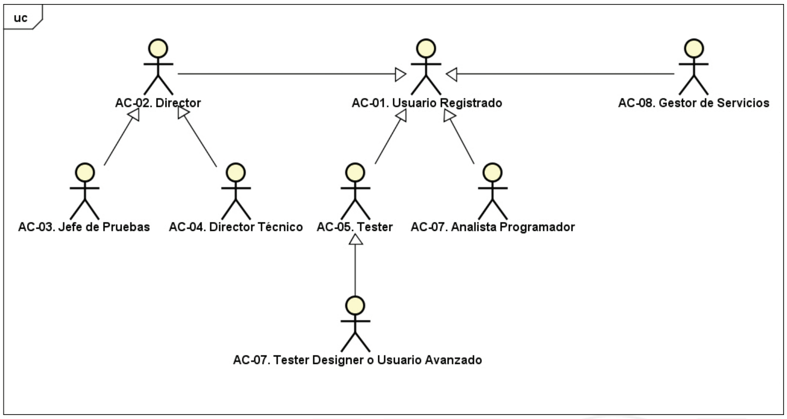

uc
AC-02. Director
AC-01. Usuario Registrado
AC-08. Gestor de Servicios
AC-03. Jefe de Pruebas
AC-04. Director Técnico
AC-05. Tester
AC-07. Analista Programador
AC-07. Tester Designer o Usuario Avanzado

Figure 3. Actors of ALAMEDA ecosystem.

- **AC-01. Registered user**: Actor who represents a team member and is also a registered user of ALAMEDA technological tools. This actor, is nested in:
  - **AC-02. Director**: Actor who represents a registered user of director or head of the corporation type to ensure product quality software. This actor is divided into two subtypes:
    - **AC-03. Test manager**. Actor who must manage a testing a project and initiating the necessary meetings to ensure its success.
    - **AC-04. Technical manager**. Actor who represents the person with overall responsibility for the management or conduct of the projects concerned.

- **AC-05. Tester**. Actor representing the user or users responsible for testing the system. In turn, this actor can be defined as a specific type:
  - **AC-06. Tester designer or advanced user**. Actor who represents an advanced user tester and is responsible for designing the tests to be performed by the system.
- **AC-07. Analyst-programmer**. Actor who represents a team member of the software development project.
- **AC-08. Services manager**. Actor who represents an administrator or a manager user.

There are 13 functionalities that define the methodological framework of ALAMEDA ecosystem; those are:

- **FR-01. Manage methodological guide**. In this scenario, two types of actors are involved: the technical manager and all the other actors who are involved in ALAMEDA. The first step to be undertaken by the technical manager is a self-assessment regarding compliance with ISO 29119. Next, it is necessary to determine whether to make improvements in the methodology of the organization. If this is not necessary, the self-assessment of compliance with the standard will be performed on a regular basis, otherwise, it will be necessary to make the appropriate changes in methodology based on and complying with the ISO 29119. In this process, the "organizational test policy" and the "organizational test strategy" will be analyzed. If there are changes, the methodology will be updated through the web platform making it available for all the users.
- **FR-02. Software product development**. In this scenario, the unique actor involved is the analyst-programmer. First, the functional requirements with the customer ought to be elaborated on and analyzed. Thereafter, the functional requirements through the use of the Enterprise Architect (EA) tool ought to be either defined or updated. Next, the functional requirements will be checked through the EA and NDT-Quality tools. If the format is not correct it will be necessary to redefine or update them, otherwise, those will be subjected to customer validation.
- **FR-03. Assess compliance with ISO 29119**. In this scenario, the unique actor involved is the test manager. First, a self-assessment of compliance with the ISO29119 through the web platform must be performed. Then, the evolution of compliance with the standard will be evaluated through the web platform. If the level of compliance is not satisfactory, it will be necessary to make a formal request for changes on the methodology of the organization.
- **FR-04. Software quality management**. In this scenario, the unique actor involved is the test manager. The first step consists of making an analysis of the code's quality through the web platform. Next, the results obtained will be analyzed. If the level of compliance is not satisfactory, a formal request for changes on the project's quality must be done.
- **FR-05. Test execution report**. In this scenario, the unique actor involved is the test manager. First, the available assets must be registered. Then, the environment must be restored to its initial state. After these steps, it is necessary to recollect the learned lessons after the study of the results of the test execution and finally, to report the completion of the test execution.

It is important to note that, to perform this step, the "FR-10. Test execution" must be performed by the testers.

- **FR-06. Test plan monitoring**. In this scenario, two actors are involved: the test manager and the technical manager. The first thing that the test manager has to do is to manage different tasks for configuring the environment and the test plan and later, to set up the environment for monitoring the test plan. During the process of monitoring, several decisions may be taken. The major duties on the part of the test manager are: (i) report on the progress of the test plan, (ii) control the test plan and report control actions to the technical director. On the side of the technical manager has to: (i) access the ALAMEDA web platform for gathering information about control actions and progress of the test plan.
- **FR-07. Test plan creation and maintenance**. In this scenario, the actor “tester” in the role of “tester designer” or an “advanced user” is involved. The first action in this scenario is to understand the context of the project. Once understood, the functional requirements of the project must be checked with through EA and NDT-Quality tools. Then, it is necessary to generate the requirements and acceptance tests documented in the TestLink using the ALAMEDA web platform, which is based on the functional requirements defined previously. The next activities that the user must take are to identify and analyze risks and the treatment that will be applied, determine how long the test plan and its programming will be and finally, record it.
- **FR-08. Design and implement tests**. In this scenario, the actor “tester” in the role of a “tester designer” or an “advanced user” is involved. The first action of this scenario is to generate the test cases using the ALAMEDA web platform. The following actions to be performed: identify feature sets, derive test conditions, derive test coverage items, derive test cases, assemble test sets and finally, derive test procedures. It is important to note that, to perform this step, the "FR-07. Test plan creation and maintenance “ must be performed.
- **FR-09. Test environment set-up and maintenance process**. In this scenario, the actor “tester” is involved. For performing this scenario, the test plan and the specification test procedure documents generated in previous steps are necessary. The first action of this scenario is to plan and design the test environment. Then, it has to be properly configured and set up. Finally, it is adamant to prepare data that will be used for the tests. It is important to note that, to perform this step, the "FR-08. Design and implement tests" must be performed.
- **FR-10. Test execution**. In this scenario, the actor “tester” is involved. Through the web platform of ALAMEDA ecosystem, the tester will: (i) carry-out the acceptance tests, (ii) check the code’s quality of the project, and (iii) perform the functional and stress tests. From the web platform, the user will register the results for comparing them with older versions. It is important to note that, to perform this step, the "FR-09. Test environment set-up and maintenance process" must be executed.
- **FR-11. Test incident reporting**. In this scenario, the actor “tester” is involved. The first action of this scenario is to analyze the results of the test execution from the web platform. If there are issues, the incident report must be created or updated (where appropriate). It is

important to note that, to carry out this scenario, the "FR-10. Test execution" must be accomplished.

- **FR-12. Users and services management**. In this scenario, the actor “services manager” is involved. The tasks that this actor can perform in this scenario are: to register, to modify or to update users so as to interact with the ALAMEDA ecosystem. These operations can be also applied to the services provided by ALAMEDA.
- **FR-13. Tools set-up management**. In this scenario, the actor “services manager” is involved. The task that this actor can perform in this scenario is to control the different tools that are integrated into the ecosystem for keeping them active and ensuring they work correctly.

3.2. Software tools ecosystem for testing

The software tools that compose the ALAMEDA ecosystem are:

- Functional tests (JUnit, NUnit, Selenium, etc.). These tools depend on a concrete project and its programming language. They are not available from the web platform but it is possible to run their execution (for example, JUnit) through the integrated tools Jenkins or Testlink.
- Load and performance Tests (JMeter). This tool allows the execution of load and performance tests. The execution of these tests can be automated by the use of Jenkins.
- Test management (TestLink). This tool allows the management of tests. The builds of Testlink can be automatically executed from Jenkins.
- Quality software product management (SonarQuBe). The web platform integrates SonarQuBe for analyzing the quality of the code of the product’s development.
- Continuous integration and task automation (Jenkins). This tool allows to integrate different tools from other companies not covered in ALAMEDA ecosystem and to automate tasks.

Table 1 shows how the activities of the methodological framework described before and these tools are implemented in ALAMEDA.

Activities	**Technological support**	**Description of technological support**
FR-01. Manage methodological guide	Web platform of ALAMEDA ecosystem	Access to the web platform where it will be possible to consult and update the methodological guide
FR-02. Software product development	Desktop platform of ALAMEDA ecosystem	Access to the Enterprise Architect tool where the functional requirements can be defined and generated automated code by NDT-Suite
FR-03. Assess compliance with ISO 29119	Web platform of ALAMEDA ecosystem	Access to the ISO 29119 module where user will: - access to the terms glossary - assess the compliance of the standard - check the status of compliance once evaluated - check the evolution of the results if you have made more than one assessment

Activities	Technological support	Description of technological support
FR-04. Software quality management	Web platform of ALAMEDA ecosystem	Access to the web platform control panel module where user will check the status of SonarQuBe executions
FR-05. Test execution report	Web platform of ALAMEDA ecosystem	Access to the web platform to inform about the test execution report
FR-06. Test plan monitoring	Web platform of ALAMEDA ecosystem - Tools to automate dynamic testing and quality Software	Access to the web platform where the indicators related to the state of the executions or build/s can be displayed
FR-07. Test plan creation and maintenance	- Web platform of ALAMEDA ecosystem - Tools to automate dynamic testing and quality software	Access to the Enterprise Architect and to the web platform where user will generate tests depending on the defined requirements. Access to Testlink for editing and visualizing test plans.
FR-08. Design and Implement Tests	- Web platform of ALAMEDA ecosystem - Tools to automate dynamic testing and quality software	Access to the Enterprise Architect and to the web platform where user will generate tests depending on the defined requirements. Access to Testlink for editing and visualizing test plans.
FR-09. Test Environment Set-Up & Maintenance Process	- Web platform of ALAMEDA ecosystem - Tools to automate dynamic testing and quality software	Access to the web platform where user will: - manage the configuration of ALAMEDA tools - manage the configuration of the continuous integration of Jenkins Access to Jenkins where user will: - configure the plugins installed in Jenkins - configure the automated tasks with Jenkins
FR-10. Test execution	- Web platform of ALAMEDA ecosystem - Tools to automate dynamic testing and quality software	Access to the web platform where user will: - execute load and performance tests Access to Jenkins where user will: - automate the execution process of load and performance (JMeter) and functional tests (JUnit, NUnit, Selenium, etc.). Access to Testlink where user will: - execute tests of the test plan and generate a build (JUnit y JMeter) Access to JMeter where user will: - execute load and performance tests.
FR-11. Test incident reporting	- Web platform of ALAMEDA ecosystem: - Tools to automate dynamic testing and quality software	Access to the web platform where the indicators related to the state of the executions or build/s can be displayed.
FR-12. Users and services management	- Web platform of ALAMEDA ecosystem	Access to the web platform where user sill: - create users - modify users - delete users - manage the different tools provided by the organization
FR-13. Tools set-up management	- Web platform of ALAMEDA ecosystem	Access to the web platform where user sill: - manage the tools configuration. - manage the ISO versions

Table 1. Description of the implementation of the methodological framework.

3.3. ALAMEDA from a user's perspective

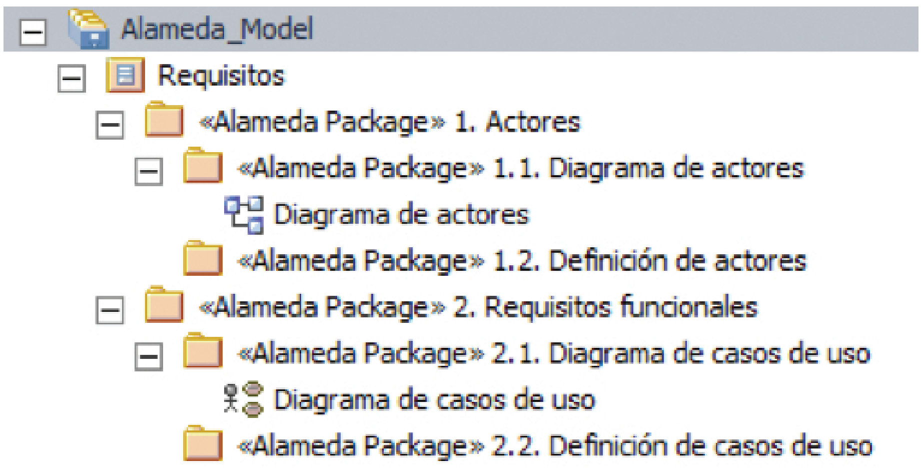
In this section, functionalities that can be made through the desktop and the web platform of the ALAMEDA ecosystem are described.

3.3.1. Desktop platform

The desktop platform of ALAMEDA ecosystem consists of an assistant for the creation of functional requirements as a preliminary step to the process of generating automated acceptance tests associated with them.

It is presented as a software layer above deployed EA assisting in the construction of very specific components: actors and system functional requirements. These components are included in EA by previous installation of NDT-Suite and correspond to the basic components to build and validate when creating a specification's requirements model.

The specification's requirements model must follow the structure shown in **Figure 4**.



Alameda_Model
Requisitos
«Alameda Package» 1. Actores
«Alameda Package» 1.1. Diagrama de actores
Diagrama de actores
«Alameda Package» 1.2. Definición de actores
«Alameda Package» 2. Requisitos funcionales
«Alameda Package» 2.1. Diagrama de casos de uso
Diagrama de casos de uso
«Alameda Package» 2.2. Definición de casos de uso

Figure 4. Specification's requirements model.

The construction of the models of functional requirements is based on two operations:

- Collect the functional requirements of a project and the relationship between them and the actors of the system.
- Automatic generation of acceptance tests (provided by ALAMEDA).

Following the previous structure, steps that are necessary to perform are:

- System actor creation/modification. The elements that represent the actors of the system are stereotyped as "AC" and can be found in the EA toolbox provided by NDT-Suite called <NDT System Requirements> DRS Actors. There are two ways of creating these elements:

    - Drag the item from the toolbox to the corresponding diagram.
    - Select the directory and include the item making right click, choosing the "Add Element" option and selecting the actor item.

- Functional requirements creation/modification. The elements that represent the functional requirements of the system are stereotyped as "FR" and can be found in EA toolbox provided by NDT-Suite called <NDT System Requirements> DRS Functional Requirements. FRs are the most complex elements of the model; they do not just present a definition of a particular value for a specific property, but rather have collections of properties (pre- and post-conditions) and may be specified based on scenarios (sequence of steps) or activity diagrams. The way the RFs are created are in the same way with the actors one. There are different types of relations between the RF and depend on the side of the connection.
    - Extends or include: only applied between RF elements.
    - Use case: only applied between AC and RF elements.

3.3.2. Web platform

The web platform of the ALAMEDA ecosystem brings together the bulk of the functionality of ALAMEDA. In this section, the applicability of each module of this platform is addressed.

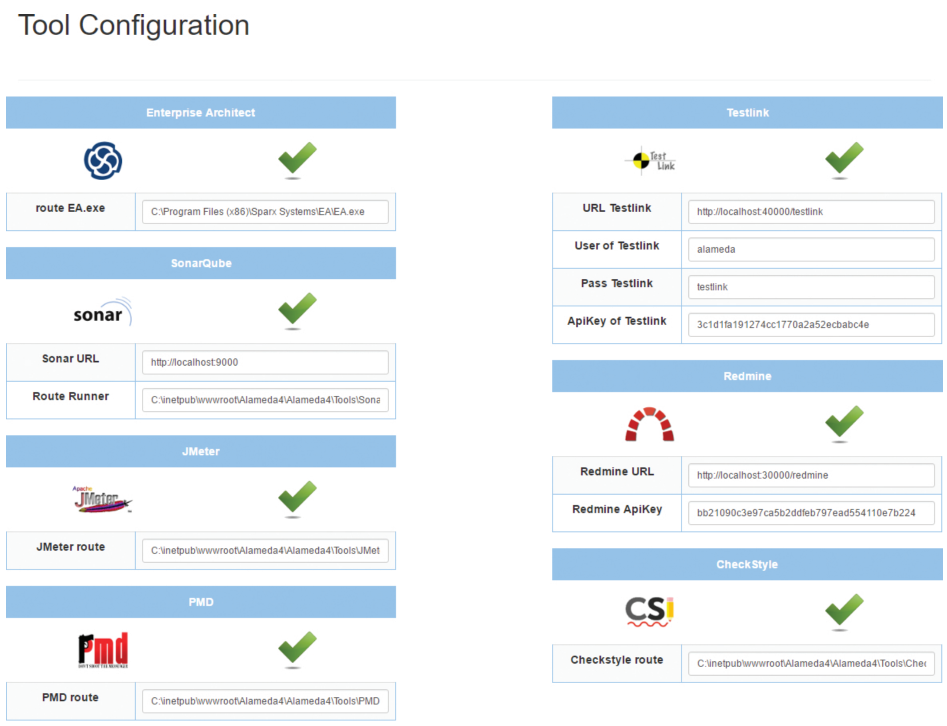
- Platform management: for access to the management module it is necessary to access as a manager. The administration panel of the platform (**Figure 5**) displays the following options:
    - New user: it allows registering new users.
    - Edit user: it allows to edit both basic information about users and performs configuration management tools that are assigned to users. Removing users is also allowed.
    - New self-assessment ISO/IEC-29119: it allows the installation of a new version of the ISO/IEC-29119 standard. A new version carries a glossary of terms associated with that role and an associated self-assessment questionnaire.
    - ISO/IEC 29119 management: it allows the management of the versions of the ISO/IEC-29119 standard loaded in the system, which includes removing and updating previous versions.
- Compliance with ISO/IEC-29119: the module in accordance with ISO/IEC-29119 standard (**Figure 6**) has four sections; it depends on an existing version of the standard installed on the platform. Therefore, if there is no version installed on the platform, it will not be possible to access to the other sections of the module. These sections are:
    - Glossary of terms. The glossary of terms shows the collection of terms that defines the norm in the first part for standardizing the terminology used in the field of Testing. In the glossary of terms, each term is shown along with the associated definition.
    - Self-assessment. The self-assessment consists of a series of questions about compliance with the rule that the user must answer based on what is known in reference to his/her

organization. The format shown in the questions are: "Yes", "No" or "Partly", so that each answer carries a score. The questionnaire is broken down into the parts of the standard and within each part, every field and block is referred to the corresponding question. There is an option for storing the state of the questionnaire and complete it later.

- ◦ Status of compliance. This section shows the current level of compliance with the standard, corresponding to the last result obtained by the user in a self-assessment. The statistics of the results are broken down into several levels: parts, areas, and blocks.
- ◦ Evolution of compliance. This section shows a comparison of the results obtained in the different self-assessments previously completed. The breakdown shown for this option is at the level of parts of the standard.

- Test generation. The test generation module (**Figure 7**) provides all the functionalities associated with the model's requirement specification validation and also, the automation of the creation of acceptance tests. This module is one of the most critical levels of the ALAMEDA ecosystem and it involves the use of three tools: EA, Redmine and TestLink. Two blocks can be distinguished, the one responsible for the tests generation and the one responsible for the tasks generation. The first one allows the creation and management test cases through the use of EA and TestLink and the other one, allows the generation of tasks depending on the unsuccessful test cases managed in the test plan in TestLink. The functionality associated with this module can be divided into four tasks:
  - ◦ Tests generation. The test generation process receives as input a specification requirements model in EA format and it returns a test plan generated in the TestLink tool with a build that allows its execution in this tool. The build process begins with a validation phase input model. If errors are found in the model definition, the test generation is not addressed. Instead, a list with the errors found during the process is provided.
  - ◦ Generated tests display. The result of the test generation process, regardless of whether they have been generated with errors or not during the validation phase, are displayed in the result viewer.
  - ◦ Tasks generation. The process consists of a scanning of the test cases executed in TestLink to generate tasks associated to the Redmine management tool.
  - ◦ Tasks monitoring and visualization. The result viewer is similar to the one shown in the test generation module with the difference that the tool deployed is Redmine. The available project tree corresponds to the generated tests. From the generated tests, the viewer can carry out the task monitoring.
- Model generation. The model generation module provides all the functionalities associated with the verification of models and its automated generation for the different phases of the development. This module only uses the EA tool.
- Software and testing analysis. The software analysis module (**Figure 8**) provides functionalities for software analysis and test execution, as well as monitoring the degree of project quality. In this module, the user will:

  - ◦ measure the quality of the code source of software products through static tests (SonarQube tool).
  - ◦ execute acceptance tests previously generated in the test execution module (Testlink).
  - ◦ perform dynamic tests in order to monitor the web nature software product measurement (JMeter).
- Furthermore, as in the test generation module, a result viewer to monitor and track all the conducted activities is provided.
  - ◦ Static code analysis. The tool used to perform static code analysis using the web platform is SonaQube. The access form to the execution of static code analysis requests two inputs to perform the test: the project to analyze and its version. The use of compressed projects files as rar or zip format is allowed.
- Control panel. This module allows the monitoring of the elements generated in all the different modules of the web platform.


Tool Configuration

Enterprise Architect ✓

route EA.exe	C:\Program Files (x86)\Sparx Systems\EA\EA.exe

SonarQube ✓

Sonar URL	http://localhost:9000
Route Runner	C:\inetpub\wwwroot\Alameda4\Alameda4\Tools\Sona

JMeter ✓

JMeter route	C:\inetpub\wwwroot\Alameda4\Alameda4\Tools\JMet

PMD ✓

PMD route	C:\inetpub\wwwroot\Alameda4\Alameda4\Tools\PMD

Testlink ✓

URL Testlink	http://localhost:40000/testlink
User of Testlink	alameda
Pass Testlink	testlink
ApiKey of Testlink	3c1d1fa191274cc1770a2a52ecbabc4e

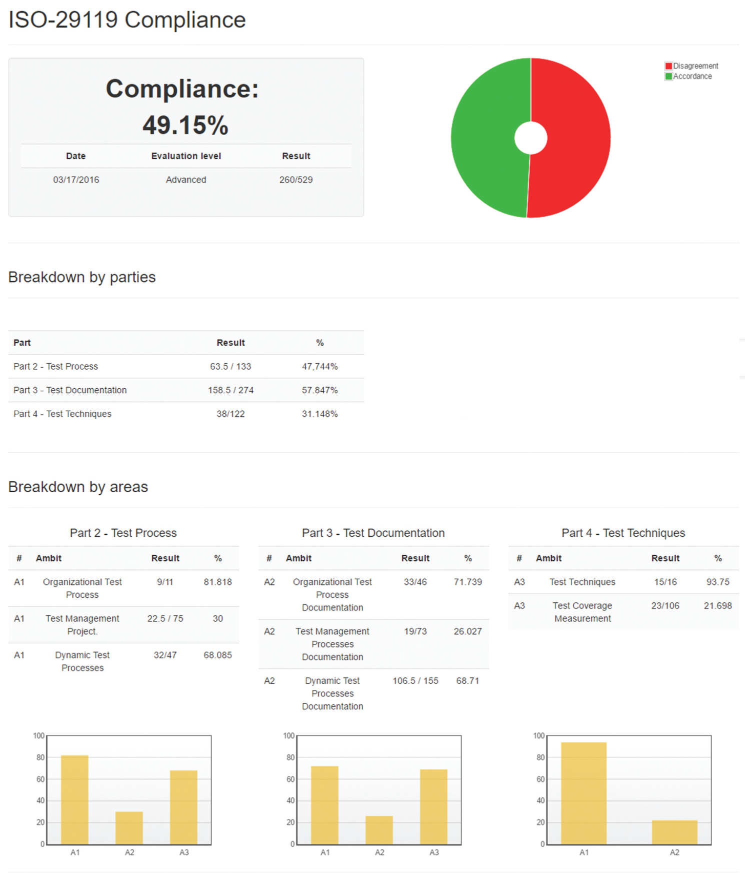
Redmine ✓

Redmine URL	http://localhost:30000/redmine
Redmine ApiKey	bb21090c3e97ca5b2ddfeb797ead554110e7b224

CheckStyle ✓

Checkstyle route	C:\inetpub\wwwroot\Alameda4\Alameda4\Tools\Chec

Figure 5. Management platform.



ISO-29119 Compliance

Compliance:
49.15%

Date	Evaluation level	Result
03/17/2016	Advanced	260/529

🟥 Disagreement
🟩 Accordance

Breakdown by parties

Part	Result	%
Part 2 - Test Process	63.5 / 133	47,744%
Part 3 - Test Documentation	158.5 / 274	57.847%
Part 4 - Test Techniques	38/122	31.148%

Breakdown by areas

Part 2 - Test Process

#	Ambit	Result	%
A1	Organizational Test Process	9/11	81.818
A1	Test Management Project.	22.5 / 75	30
A1	Dynamic Test Processes	32/47	68.085

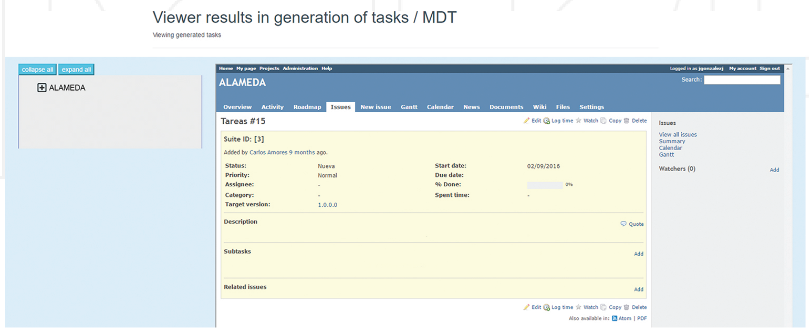
Part 3 - Test Documentation

#	Ambit	Result	%
A2	Organizational Test Process Documentation	33/46	71.739
A2	Test Management Processes Documentation	19/73	26.027
A2	Dynamic Test Processes Documentation	106.5 / 155	68.71

Part 4 - Test Techniques

#	Ambit	Result	%
A3	Test Techniques	15/16	93.75
A3	Test Coverage Measurement	23/106	21.698

Figure 6. Compliance with ISO/IEC-29119.



Viewer results in generation of tasks / MDT

Viewing generated tasks

Tareas #15

Suite ID: [3]

Added by Carlos Amores 9 months ago.

Status:	Nueva	Start date:	02/09/2016
Priority:	Normal	Due date:	
Assignee:	-	% Done:	0%
Category:	-	Spent time:	-
Target version:	1.0.0.0		

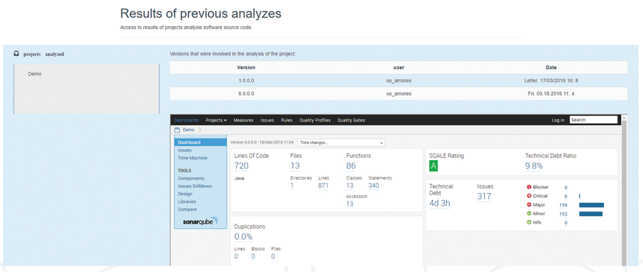
Description

Subtasks

Related issues

Also available in: Atom | PDF

Figure 7. Test generation.


Results of previous analyzes

Access to results of projects analysis software source code.

projects analyzed

Demo

Versions that were involved in the analysis of the project:

Version	user	Date
1.0.0.0	us_amores	Letter. 17/03/2016 16: 8
6.0.0.0	us_amores	Fri. 03.18.2016 11: 4

Version 6.0.0.0 - 18/Mar/2016 11:04

Lines Of Code 720 (Java); Files 13; Directories 1; Lines 871; Functions 86; Classes 13; Statements 340; Accessors 13

Duplications 0.0%; Lines 0; Blocks 0; Files 0

SQALE Rating A; Technical Debt Ratio 9.8%

Technical Debt 4d 3h; Issues 317

Blocker	0
Critical	6
Major	159
Minor	152
Info	0

Figure 8. Software and testing analysis.

4. Conclusions and future work

This chapter presents the ALAMEDA ecosystem for software testing. Over the last decade, a number of studies, as well as technical and theoretical proposals have emerged to improve the process of software testing. However, the software industry has often carried out this process in a manner that was not standardized and hence, proved difficult to imply to systematization, management and continuous improvement of this process.

The ALAMEDA platform has been developed so as to cover the observed gap in the industry. In this sense, ALAMEDA provides a methodological framework based on the ISO/IEC 29119 standard and the application of the MDE paradigm of software testing (known as MDT).

Regarding the ISO/IEC 29119, this one is a new standard which implies a certain novelty of the software industry. ALAMEDA integrates the main foundations of this standard within its ecosystem in order to facilitate the adoption of this standard by industry. In addition, ALAMEDA provides mechanisms to measure the degree of compliance to the ISO/IEC 29119.Thus, any software company can measure its evolution to meet this standard and carry out a process of continuous improvement in its software development processes.

Regarding the implementation of MDT, ALAMEDA has included the concept of early testing, thanks to it for integrating the NDT methodology within its ecosystem. Thus, it is possible to obtain test cases from the functional requirements and manage these tests effectively. Moreover, this mechanism provides traceability between tests and functional requirements allowing always to know which functionality is being tested.

Finally, and taking into account the features of ALAMEDA, it has not been possible to find a technological solution to cater for all needs associated with the process of software testing from the early stages and complying with the international standard ISO/IEC 29119. In this sense, ALAMEDA improves the state-of-the-art within the management of software quality assurance because it provides an innovative technological solution which is based on well-validated theoretical foundations.

Acknowledgements

This research has been supported by the MeGUS project (TIN2013-46928-C3-3-R), by the SoftPLM Network (TIN2015-71938-REDT) of the Spanish Ministry of Economy and Competitiveness and Fujitsu Laboratories of Europe (FLE). In addition, we would like to acknowledge the leadership of Servinform during the development of the ALAMEDA ecosystem. Our research group has collaborated in the conceptual design of ALAMEDA as well as in analysis, design, and testing tasks of the platform.

Author details

José González Enríquez*, Julián Alberto García-García, Francisco José Domínguez-Mayo and María José Escalona Cuaresma

*Address all correspondence to: jose.gonzalez@iwt2.org

Computer Languages and Systems Department, University of Seville, Seville, Spain

References

[1] Boehm B, Abts C, Brown AW, Chulani S, Clark BK, Horowitz K, Madachy R, Reifer D & Steece B. (2000). Software Cost Estimation with COCOMO II. Prentice Hall PTR, Upper Saddle River, NJ. ISBN 0-13-026692-2.

[2] Jiang J & Klein G. (2000). Software Development Risks to Project Effectiveness. Journal of Systems and Software. 52(1):3–10.

[3] Heckel R & Lohmann M. (2003). Towards Model-Driven Testing. Electronic Notes in Theoretical Computer Science. 82(6):33–43.

[4] Ahmed A. (2012). Software Project Management. A Process-Driven Approach. CRC Press.

[5] Taguchi G, Chowdhury S & Wu Y. Introduction to Design of Experiments. Taguchi's Quality Engineering Handbook, 501–505.

[6] Binder B. (1999). Testing Object-Oriented Systems. Addison Wesley.

[7] RiazAhamed SS. (2010). Studying the Feasibility and Importance of Software Testing: An Analysis. IJEST. 1(3):119–128.

[8] Schmidt DC. (2006). Model-Driven Engineering. Computer-IEEE Computer Society. 39(2):25–31.

[9] Miller J & Mukerji J. (2003). Model Driven Architecture 1.0.1 Guide. Object Management Group, Inc.

[10] Escalona MJ & Aragón G. (2008). NDT. A Model-Driven Approach for Web Requirements. IEEE Transactions on Software Engineering. 34:377–394.

[11] García-García JA, Ortega MA, García-Borgoñon L & Escalona MJ. (2012). NDT-Suite: A Model-Based Suite for the Application of NDT. In Web Engineering. Springer, Berlin Heidelberg. pp. 469–472.

[12] García-García JA, Escalona MJ, Martinez-Garcia A, Parra C & Wojdyeski T. (2015). Clinical Process Management: A Model-Driven & Tool-Based Proposal. In Information Systems Development: Transforming Healthcare through Information Systems.

[13] Tassey G. (2002). The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards and Technology, RTI Project, 7007(011).

[14] López AS, Valle DC, Escalona MJ, Lee V & Goto M. (2015). Patient Lifecycle Management: An Approach for Clinical Processes. In Bioinformatics and Biomedical Engineering. Springer International Publishing. pp. 694–700.

[15] Kent S. (2002, May). Model Driven Engineering. In International Conference on Integrated Formal Methods. Springer, Berlin Heidelberg. pp. 286–298.

[16] ISO: ISO/IEC/IEEE 29119 Software Testing. http://www.softwaretestingstandard.org/ (2013). Accessed 10 May 2016.

[17] Fondemenet F & Silaghi R. (2004). Defining Model Driven Engineering Process. 3rd Workshop in Software Model Engineering (WISME2004). October 11–15, Lisbon, Portugal. 2004.

[18] OMG. (2005). UML2.0, Unified Modelling Language. Object Management Group, formal/2005-07-05, 2005.

[19] OMG. (2008). Documents Associated with Meta Object Facility (MOF) 2.0 Query/View/ Transformation. Object Management Group. http://www.omg.org/spec/QVT/1.0/. 2008.

[20] Thiry L & Thirion B. (2009). Functional Metamodels for Systems and Software. Journal of Systems and Software. 82(7):1125–1136. DOI: http://dx.doi.org/10.1016/j.jss.2009.01.042. 2009.

[21] Whittle J, Hutchinson J & Rouncefield M. (2014). The State of Practice in Model-Driven Engineering. IEEE Software. 31(3):79–85.

[22] OMG. (2003). MDA Guide of OMG. Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf

[23] Object Management Group (OMG). http://utp.omg.org/. Last accessed: July, 2016

[24] Dai ZR. (2004). Model-Driven Testing with UML 2.0. In Proceedings of the 2nd European Workshop on Model Driven Architecture, Computing Laboratory, University of Kent, Canterbury, UK, 2004, pp. 179–187

[25] Andersin J. (2004). TPI–A Model for Test Process Improvement. In Seminar. University of Helsinki, Helsinki-Finland.

[26] Gutiérrez J, Aragón G, Mejías M, Domínguez F & Cutilla CR (n.d.). Automatic Test Case Generation from Functional Requirements in NDT. WebRE 2012.

[27] García-García JA, Alba M, García-Borgoñon L & Escalona MJ. (2012). NDT-Suite: A Model-Based Suite for the Application of NDT, LNCS 7387, pp. 469–472

[28] García-García JA, Victorio J, García-Borgoñón L, Barcelona MA, Dominguez-Mayo FJ & Escalona MJ. (2013). A Formal Demonstration of NDT-Quality: A Tool for Measuring the Quality using NDT Methodology. The 21st Annual Software Quality Management (SQM) Conference. ISBN 978-0-9563140-8-6.

[29] García-García JA, Cutilla CR, Escalona MJ, Alba M & Torres J. (2011). NDT-Driver, A Java Tool to Support QVT Transformations for NDT. The Twentieth International Conference on Information Systems Development (ISD 2011). ISBN: 978-1-4614-4950-8.

[30] Escalona MJ, Gutiérrez JJ, Mejías M, Aragón G, Ramos I, Torres J & Domínguez FJ. (2011). An Overview on Test Generation from Functional Requirements. Journal of Systems and Software. 84(8):1379–1393, ISSN 0164-1212, http://dx.doi.org/10.1016/j.jss.2011.03.051.

[31] Ostrand TJ & Balcer MJ. (1988). Category-Partition Method. Communications of the ACM. 31(6):676–686.