Model-driven Test Engineering
A Practical Analysis in the AQUA-WS Project

C. R. Cutilla[1], J. A. García-García[1], J. J. Gutiérrez[1], P. Domínguez-Mayo[1], M. J. Escalona[1], L. Rodríguez[2] and F. J. Domínguez-Mayo[1]
[1]*IWT2 Research Group. University of Seville, Seville. Spain*
[2]*Emasesa, Seville, Spain*

Keywords: Model-driven Engineering, Quality, Software Metrics, Model-driven Web Engineering, Methodologies.

Abstract: The effective application of test phases has been one of the most relevant, critical and cost phases in the life cycle of software projects in the last years. During the test phase, the test team has to assure the quality of the system and the concordance with the initial requirements of the system. The model driven paradigm is offering suitable results in some areas and the test phase could be one of them. This paper presents how the application of this paradigm can help to improve this aspect in the functional test generation and it analyses the experience in a real project developed under this approach.

1 INTRODUCTION

The quality assurance of a system is one of the most studied and analysed aspects in Software Engineering. The finding of methods, techniques and tools to reduce quality assurance costs and increase the guarantee of the results becomes an essential aim for enterprises and development teams (Ahmed, 2012).

The test phase is one of the most important in quality assurance. It guarantees that the system meets its necessities in relation to requirements (Binder, 1999).

However, bad estimations, time problems or other inconveniences in projects make the amount of resources oriented to the test phase not to be enough.

For this reason, both research and enterprise are looking for solutions to reduce the cost of this phase. In the last years, the use of the model-driven paradigm for test generation has become an important fact that is offering good results (Heckel and Lohmann, 2003).

This paper presents how a Model-Driven Web approach, named NDT (Navigational Development Techniques) (Escalona and Aragon, 2008) was enriched with a set of models and transformation so as to generate functional test cases from the functional requirements. This improvement of NDT enriches the approach offering a suitable solution for quality assurance in systems developed with NDT. In order to illustrate this improvement, this paper presents its application in a real project named AQUA-WS Project.

This paper is an evolution of the paper presented in (Cutilla et al., 2011) where we adapted our original and theoretical approach for functional test generation presented in detail (Gutierrez et al., 2011) for being applied in a practical environment. This new paper presents the complete adaptation of the theoretical approach and learned lesson in the enterprise environment.

The paper is structured as follows: it starts with a global vision of NDT and presents how the approach of (Gutierrez et al., 2008) was adapted to generate functional tests cases from functional requirements in NDT. Section 3 introduces the AQUA-WS project and presents how the process was adapted to be explained in this paper and learned lessons from the experience are later concluded. Finally, it offers some related work and conclusions as well as suggests some future work in this line of research.

2 AN OVERVIEW OF NDT

NDT methodology is a Web methodology proposed

Cutilla C., García-García J., Gutiérrez J., Domínguez-Mayo P., Escalona M., Rodríguez L. and Domínguez-Mayo F..
Model-driven Test Engineering - A Practical Analysis in the AQUA-WS Project.
DOI: 10.5220/0003991301110119
In *Proceedings of the 7th International Conference on Software Paradigm Trends* (ICSOFT-2012), pages 111-119
ISBN: 978-989-8565-19-8

under the Model-Driven paradigm. Initially, NDT dealt with the definition of a set of formal metamodels for the requirements and analysis phases. In addition, NDT defined a set of derivation rules, expressed under the standard QVT (Query-View-Transformation) (OMG 2008), which generated the analysis models from requirements model. QVT standard defined a declarative and imperative language proposed by the OMG (Object Management Group) for model transformation in the Model-Driven Engineering context.

Nowadays, NDT defines a set of metamodels for every phase of the life cycle of software development: the Feasibility Study phase of the project, the Requirements phase, the Analysis phase, the Design phase, the Implementation phase, the Testing phase, and finally the Maintenance phase. Besides, it states new transformation rules to systematically generate models.

Figure 1 represents a diagram of the life cycle of NDT. Although it is represented sequentially, NDT supports different life cycles like iterative or agile processes.

The Feasibility Study phase gathers the result of the studies that verify the viability of a particular software project. The needs of the project can be analysed, but not in depth. NDT proposes derivation rules to generate the basic model of the Requirements phase from these requirements specifically defined.

The next one is the Requirements phase. It focuses on defining the catalogue of requirements which contains the needs of the system to be developed. It is divided into a series of activities (see Figure 2): capture, definition and validation of requirements.

The objectives of the system are stated in the first activity of the requirements phase. From these objectives, the different system requirements are captured and defined.

NDT proposes to classify project requirements according to their nature: information storage requirements (storage requirements and new natures), functional requirements, actor requirements, interaction requirements and non-functional requirements. NDT provides special patterns and UML techniques (OMG 2005) to define them, such as the use cases technique for functional requirements specification.

Once the requirements have been defined, NDT recommends validating them. If there are no errors in the definition of requirements, the System Requirements Document can be generated. This document is the starting point for the specification of the analysis phase.

On the contrary, if there is just an error in the definition of requirements, the previous activities would be carried out once again. This process can be repeated until obtaining a suitable System Requirements Document.

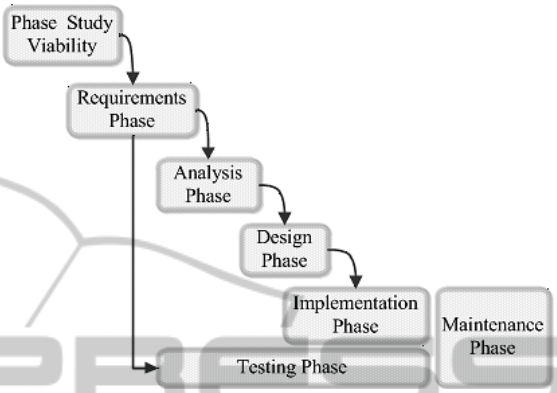

Figure 1: Phases covered by NDT.

The use of patterns or templates to define every requirement offers a structured and in-depth description.

Moreover, the fact that some fields of such patterns only admit particular values makes results be obtained systematically in the remaining life cycle process of NDT.

The following phase, the Analysis phase, will include the resulting products from the analysis, definition and organization of requirements in the previous phase.

NDT proposes four models in this phase (see Figure 2) described as follows:

1. The Conceptual Model, which represents the static structure of the system.
2. The Process Model, which represents the functional structure of the system.
3. The Navigation Model, which shows how users can navigate through the system.
4. The Abstract Interface Model, which are a set of prototypes of the system interface.

The transition between the requirements model and the analysis model is standardized and automated and based on QVT transformations, which change the concepts of requirements metamodels to design the first versions of the analysis models. These models are known in NDT as basic models of analysis. For instance, the basic conceptual model of analysis is obtained from the storage requirements defined during the Requirements phase.

Thereafter, the team of analysts can transform these basic models to enrich and complete the final

model of analysis. As this process is not automatic, the expertise of an analyst is required. Transformations are represented in Figure 2 through the stereotype *«NDTSupport»*.

In order to ensure consistency between requirements and analysis models, NDT controls these transformations by means of a set of defined rules and heuristics.

After completing the analysis models, NDT define transformations to generate the basic models of the Design phase.

Then, the Design phase provides the relevant knowledge to carry out the analysis in the machine. It is oriented to the specific platform used and must be related to the structure of the future code developed during the Implementation phase.

NDT proposes the following models for the Design phase:

1. The Design Class Model, which is based on design-oriented layers. In fact, this model consists of three other models:
   a. The Presentation Class Model, corresponding to the presentation layer. This model is obtained from the navigation classes of the Analysis phase.
   b. The Business Class Model, corresponding to the business logic layer. This model is obtained from the process class model of the Analysis phase.
   c. The Data Access Classes Model, corresponding to the data access layer. This model is obtained from the content class model of the Analysis phase.
2. The Prototype Design Model, which deals with the prototypes design of the application that have to be agreed with the client. This model is obtained from the navigation model of the Analysis phase.
3. The Physical Data Model, which describes the structure of a database such as data, data relationships and the restrictions they must comply with. This model is obtained from the content class model of the Analysis phase.

After creating the basic models, the team of analysts can execute controlled transformations of such models in order to enrich and create the final design models.

This idea is being followed in the remaining life cycle with implementation. The NDT life cycle ends with the Maintenance phase. This process begins when the project has gone into production and ends when the system is out of date. This process is very complex and critical consequently, this phase is one of the most expensive in the life cycle.

To sum up, NDT offers an environment conducive to the development of Web systems since it completely covers the life cycle of software development. Furthermore, this methodology provides a set of Java-free tools that speed up the work for the development of every phase in the life cycle. This toolkit is distributed under the NDT-Suite package (NDT-Suite 2012), which is composed of the following tools:

1. NDT-Profile is a specific profile for NDT, developed using Enterprise Architect (Enterprise Architect 2012). This tool offers the chance of having all the artefacts that define NDT easy and quickly as they are integrated within the tool Enterprise Architect.
2. NDT-Driver is the key tool for executing transformations among NDT models. It implements a set of automated procedures that enables to perform all MDE transformations among the different models of NDT that were previously described. The data source to use this tool is a project developed with NDT-Profile.
3. NDT-Quality is a tool that automates most of the methodological review of a project developed with NDT-Profile. It checks the quality of using NDT methodology in each phase of software life cycle and the quality of traceability of MDE rules of NDT.
4. NDT-Prototype is a tool designed to automatically generate a set of XHTML prototypes from the navigation models described in the Analysis phase, of a project developed with NDT-Profile.
5. NDT-Glossary consists in implementing an automated procedure that generates the first instance of the glossary of terms of a project developed by means of NDT-Profile tool. This tool is useful for the validation of requirements captured during the Requirements phase of the project.

2.1 Including Early Testing in NDT

To be applied in several real projects is one of the most important advantages of NDT. The necessity of ensuring the quality of the system as well as improving the approach with the Test phase was detected during these applications. Thus, the life cycle of NDT was enriched with a new Test phase.
Using the same ideas that in the rest of its phases, a Test phase was added in NDT. Once the requirements specification phase has been completed and the catalogue of system requirements has been

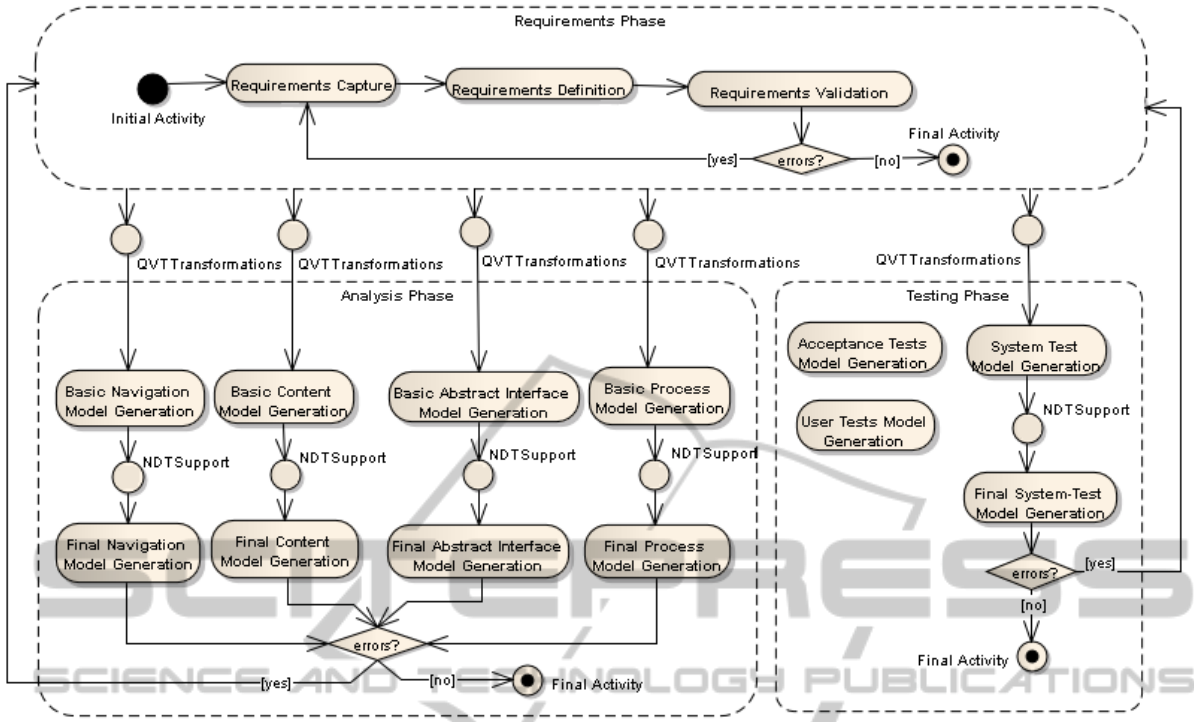

Figure 2: NDT Transformations from Requirements to Analysis and from Requirements to Testing model.

drafted and validated, NDT defines derivation rules to generate the System test model, the Testing phase model and the Analysis phase models. Figure 2 shows all these transformations through the stereotype *«QVTTransformation»*.

NDT conceives the Testing phase as an early phase of software life cycle. Thus, test cases can be defined in relation to the needs of the system that have been gathered during the Requirements phase. Furthermore, this methodology proposes to carry it out together with the remaining phases.

The Testing phase is divided into the following activities: drawing up the test plan, a parallel activity to the Analysis phase; environment specification and test plan design, a parallel activity to the Design phase; and implementation of test plan, a parallel activity to the system construction and Implementation phase.

NDT suggests three models in this phase (see Figure 2): implementation tests model, system tests model, and acceptance tests model. Every model is described by means of the use case diagrams of UML.

The system tests model is the only one that can be generated systematically. NDT proposes derivation rules to generate the basic model of system tests from the functional requirements defined in the Requirements phase. These transformation rules are based on the rules described in the paper (Gutierrez et al 2008).

Figure 3 represents a global view of the transformations used for testing generation in NDT. From the functional requirements model of NDT, obtained during the requirements, a transformation is generated to enrich this model with some specific paths that describe each functional path (T1 in Figure 3). Two transformations are defined from this enriched model. The first one, T2, enables to get Test Scenarios Model where any specific possible scenario is described. The second one, T3, allows the generation of another model, Test Value Model. Moreover, both are integrated within a fourth transformation, T4, to get the Test Case Model.

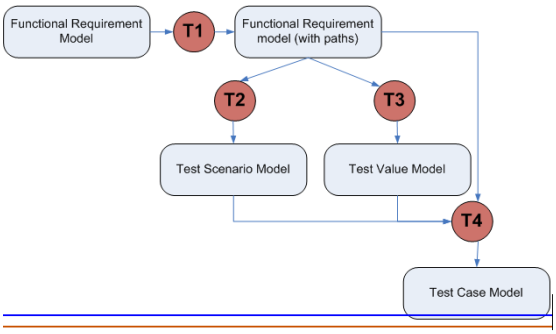

Figure 3: An overview of the transformation process.

All transformations were defined in QVT-Procedural and they are based on NDT metamodels and new models defined for testing. This paper does not aim to present these transformation and metamodels in details as they can be consulted in (Gutierrez et al 2008). It particularly intends to explain how this process can be applied in a real project like AQUA.

2.2 Implementing Early Testing in NDT-Suite

In order to carry out this early testing solution in NDT for its practical application, we have to implement it in NDT-Suite.

Metamodels for testing were included in NDT-Suite and transformations were implemented and included in NDT-Driver. Besides, both NDT-Quality and NDT-Report were enriched to support this new phase

In NDT, functional requirements can be described in two different ways: using the scenario technique or UML activity diagrams.

NDT provides different transformation rules to generate functional test cases, depending on how you define a functional requirement. For instance, if a functional requirement is described by means of scenarios, it may generate as many test cases as scenarios that requirement has. Nevertheless, if a functional requirement is described by means of activity diagrams, it may produce as many test cases as paths between the initial and the final activities exist.

Once the basic model of the system tests has been created, the team of analysts can execute transformations in this model to enrich and complete it. As this step is not automatic, the expertise of the analyst is needed. These transformations are represented in Figure 2 by the stereotype *«NDTSupport»*.

3 AQUA-WS PROJECT

Emasesa (Emasesa 2012) is a company operating the general management of the urban water cycle, providing and ensuring water supply to all the citizens in Seville. For this reason, its objectives are to guarantee the quality of this supply, solve problems occurred in the supply and control the correct use of water in Seville.

AQUA-WS (AQUA-WebServices) project consists in developing and implementing an integrated business system for customer management, intervening in water distribution and cleaning up, and managing projects and work.

This project is born when Emasesa requires integrating their existing systems in a single one as well as upgrading the technological platform of the system. The present systems are the customer management system (AQUA-SiC), network management system (AQUA-ReD) and the works and projects management system (AQUA-SigO).

The AQUA-WS project comprises four main objectives:

- To migrate AQUA-SiC, AQUA-ReD and AQUA-SigO to a common technological platform (JavaJ2EE platform) and join them in a single system: AQUA-WS. This is the core AQUA.
- To develop an integrated system to support decision making in AQUA. This is DSS (Decision Support System).
- To offer a suitable way, based on an integration bus, to connect AQUA with other transversal systems in Emasesa.
- To implement a friendly interface with a suitable connection for both clients and employees. This is FrontEnd.

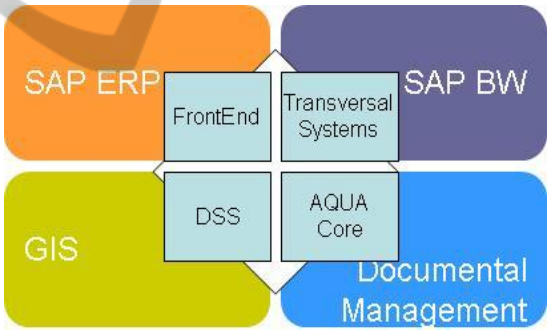

Figure 4: AQUA-WS Architecture.

Figure 4 shows the architecture of AQUA-WS. AQUA Core, FrontEnd, DSS and the communication with transversal systems are based on the use of SAP ERP (SAP Enterprise Resource Planning) (SAP ERP, 2012) and SAP BW (SAP Netweaver Business Warehouse) (SAP BW, 2012) on a GIS (Geographical Information System) and a documentary resources system.

The development of AQUA is fronted by a mixed group composed of two international companies and two research groups from two Spanish universities. Our group constitutes the Technical Quality Office of AQUA-WS project.

4 TEST PHASE IN AQUA PROJECT

During the development of AQUA-WS, the working team is using NDT and their associated tools. In the Test phase NDT-Driver (a tool described in Section 2) has been used to generate the test plan.

In Cutilla et al., 2011 the study presents an estimation of the hour of work that would suppose realizing the test plan manually and the estimation realizing with the NDT-Driver tool. In these preliminary estimates the project saves approximately 1808 hours of work. The analyst team was divided in two groups. The group A performed the test plan manually and the group B generates the test plan with the NDT-Driver tool. Group A devoted approximately 40 minutes for each test case, about 10 minutes more than was estimated. After the review of the Technical Quality Office of AQUA-WS Project, the group A's test plan was incomplete, there were many test case not identified. Consequently, the group A had to make a one more cycle of delivery and revision of the test plan than the group B, increasing the number of hours of work.

AQUA-WS is following an iterative life cycle; NDT-Driver is applied for each piece of the system that is presented in order to generate its test plan.

The Testing phase relevance is noticed in the proper AQUA-WS project planning, which assigns nearly 40% of the duration of the project on the different types of tests.

This phase has been provided by a group of experts headed by some members of the AQUA-WS Technical Quality Office. This group consists of members of the Technical Quality Office, members of the company responsible for the maintenance of the current system and members of the companies carrying out the development of AQUA-WS system.

The test cycle is very complex and there are some different kinds of tests in different part of the life cycle. These tests are: The Unit tests are performed by the developments team when testing the design and behaviour of every element of the system is built. For this test, the developments teams use JUnit (JUnit 2012) tool, which allows the automation of code Java test cases.

The Integration tests verify the correct relationship among the system components through their interfaces and their compliance with the established functionality. JMeter's (JMeter 2012) tool is being executed to deal with unit testing performances. It is a loading tool useful to prove simulations on any software resources. It was initially designed to strength testing on Web applications. The performance on AQUA-WS project is considered relevant due to the large number of the staff who is simultaneously working at the company.

The Acceptance tests, a subset of the System testing, are performed by the testing group and are intended to verify both, the system functionality and quality attributes. These tests are carried out in an environment as similar as possible to the operational one. As we have previously mentioned, the system tests are automatically generated by NDT-Driver from the functional requirements defined in the Requirements Phase of the project. In test implementations some errors or incidents are detected and reported to the development team who works simultaneously to solve such incidents.

The User tests are a subset Acceptance tests that are executed with final users. They focus on ensuring the correct implementation of the system.

The use of NDT is oriented towards Acceptance and User tests. After studying the different tools for the automatic execution of system tests, all of them were discarded. Due to the high level of casuistry and interrelation of data with the different systems, controlling the automatic test was more tedious than the individual accomplishment of the testing team.

The system tests are the most important in AQUA-WS project. Their accomplishment is structured in two cycles of tests executed by the testing team. In the first cycle, a large number of tests are applied with the aim of validating a major number of casuistries. The tests cycles have been carried out according to the Test Plan outline, which has been generated by NDT-Driver (a tool described in Section 2). This Test Plan has shown all the functionality and casuistry of the project.

The incidents detected during the different cycles have been managed by a procedure designed by the Technical Quality Office in liaison with the project manager and all the implied actors. It is based on different types of incidents and degrees of views.

Once the development team has solved all incidents, the second tests cycle begins. The tests gathered in the Test Plan are executed to check and assess that all the reported incidents have been correctly managed and no new incident has been generated due to collateral effects.

Provided that this project arises from the unification of previously well-established systems,

the tests lead, even more, to keep the functionality and a high degree of similarities to the previous application, but without forgetting the optimization of the system and its performance.

5 LEARNED LEASSONS

Thanks to the collaboration with the Technical Quality Office as the Testing group in this project, we must highlight the importance of delivering the different cycles of planned tests during the course of the project. One of the key points to perform a successful Test phase is to have a defined Test Plan, as complete and thorough as possible, in order to verify that all the functionality offered by the developed system works correctly, according to the end user's expectations.

In this regard, AQUA-WS project reveals that due to the wide range of possible test cases and the large number of potential casuistry, the analyst team must complete, correctly define and structure the Test Plan for all subsystems conforming the project (indicated in Section 4). The lack of involvement when providing the Testing group with a correct Test Plan has been essentially motivated by the delays occurred during life cycle of the project.

Other important learned aspect was the necessity of "hiding" the mechanism of transformations, metamodels and Model-Driven paradigm in general to the development team. In fact, these concepts are very abstract and complex for the development team and, if they are put into practise, suitable tools, like NDT-Suite, based on UML diagrams that are widely known by software teams, are definitely required.

Another essential aspect is the early detection of errors through the early testing. As Functional tests are directly and automatically derived from requirements, both the development team and users can check what they demanded in the requirements by detecting inconsistencies and other errors in early phases.

6 RELATED WORK

As it was concluded from comparative studies (Escalona et al. 2007), the Test phase is one of the less studied in Web Engineering. Recently, only some approaches are working in this respect. If this study also focused on early testing, that is, in test derived from requirements, the set of approaches that support them would reduce.

Thus, WebML (Brambilla et al., 2009) includes BPMN (Business Process Management Notation) as Computation Independent Model and proposes the systematic generation of test cases by means of Model-Driven paradigm.

Robles et al. (Robles et al., 2009) are carrying out something similar generating mockups from requirements as a base for testing the application. Thus, in this approach a Model-Driven approach begins with a set of requirements described by means of some mockups and then generates user interfaces from them as well as, simultaneously, a set of tests to assess them. Additionally, the approach uses XML to describe the system.

Another relevant work, developed by Perez et al.(Pérez et al., 2009), highlights product lines, although it offers specific examples for Web environments. It uses the Model-Driven paradigm for the systematic generation of test cases in product lines described with a dynamic model.

Some classical Web approaches like UWE (UML Web Engineering) (Koch et al., 2008) or OOHDM (Object-Oriented Hypermedia Method) (Rossi and Schwabe, 2008) try to support the Test phase in their life cycle. Nevertheless, they have not provided a concrete solution yet.

In Software Engineering field, the use of mechanisms for early testing in the generation of functional tests cases is not a new idea. However, as it can be concluded from the study in (Escalona et al.2011) there is an important gap between the development process and the test process, which are frequently disconnected. The Model-Driven paradigm is offering good results when trying to improve this aspect.

In comparison with these works, our work is quite relevant since it applies the Model-Driven paradigm to Web environment for early testing.

7 CONCLUSIONS AND FUTURE WORK

In Software Engineering and especially in Web Engineering, it is important to conduct a complete an exhaustive Test phase in the project life cycle for the developed product may assure quality and meet clients and final users' needs and expectations.

To achieve these aims, the Test phase must be planned with enough time to be attained and, additionally, it must be provided with the necessary resources, both technical and human.

This work presents how this aspect has been

managed and how the different tests cycles have taken place in a relevant project: the AQUA-WS project.

As a research objective, the project AQUA-WS has started an important line of investigation that should be continued. The use of automatic technologies to generate system tests from functional requirements captured during the project phase of requirements has reduced the time analysts teams would have spent in developing the different Test Plans.

In this sense, the application of NDT methodology and its package of tools (NDT-Suite) for such important project have undoubtedly meant a great challenge.

One of the most recent ideas carried out in the last years by several authors is the use of Model-Driven Engineering (MDE) paradigm in test generation (Escalona et al., 2011). MDE is a new paradigm that focuses on defining a set of metamodels, which are instanced in the development process, and a set of transformations among them. Consequently, some related work in this line of research has recently emerged.

The application of the MDE paradigm in the test generation context is commonly referred to as Model-Based Testing (MBT).

As future research papers, we propose using MBT to improve the Testing phase in the NDT methodology. Nowadays, NDT only supplies test cases generation. In the future we intend to progress on this, so that, it will not only generate test cases automatically, but also the data set used to test.

Moreover, we propose future lines of research with the aim of creating control panels to carry out an automatic management of the Testing phase, not as manual as it is being executed today.

Finally, we are working on improving the approach presented in Figure 3 by adding mechanisms to give priority to derived testing. The approach presented in this paper generates a high number of testing and, in real projects, there is a common necessity of performing a set of testing exclusively. We are focusing our work on offering mechanisms for giving priority to the most critical tests.

ACKNOWLEDGEMENTS

This research has been supported by the Tempros project (TIN2010-20057-C03-02) and the project Red CaSA (TIN 2010-12312-E) of the Ministerio de Ciencia e Innovación, Spain, as well as by the NDTQ-Framework project of the Junta de Andalucía, Spain (TIC-5789).

REFERENCES

Ahmed, A. *Software Project Management. A Process-Driven Approach*. CRC Press. 2012.

Binder, B. *Testing Object-Oriented Systems*. Addison Wesley.1999.

Brambilla, M., Fraternali, P., Tisi, M. A Transformation Framework to Bridge Domain Specific Languages to MDA. *4th Workshop on Model-Driven Web Engineering. LNCS* 5421. France, 2009. pp. 167-181.

C. R. Cutilla, J. A. García-García, M. Alba, M. J. Escalona, J. Ponce, L. Rodríguez, Aplicación del paradigma MDE para la generación de pruebas funcionales - Experiencia dentro del proyecto AQUA-WS, *6th Conferencia Ibérica de Sistemas y Tecnologías de Información*, Chaves, 2011 (CISTI 2011) vol 1, pp 827-831

Heckel, R. Lohmann, M. Towards Model-Driven Testing. *Electronic Notes in Theoretical Computer Science* 82. Nº6. pp. 1-11. 2003.

Enterprise Architect. http://www.sparxsystems.com. Last Accessed 02-2012

Escalona, M. J., Torres, J. Mejías, M., Gutiérrez, J. J., Villadiego, D. The Treatment of **N**avigation in Web Engineering. *Advances in Engineering Software*. Vol. 38. pp.267-282. Elsevier. 2007.

Escalona, M. J.**,** Aragón, G., 2008. NDT: A Model-Driven Approach for Web Requirements, *IEEE Transactions on Software Engineering*, 34(3). pp. 370-390.

Escalona M. J., Gutiérrez J. J., Mejías M., Aragón G., Ramos I., Torres J., Domínguez F. J. An overview on test generation from functional requirements. *The Journal of Systems and Software*. Elsevier. ISSN: 0164-1212. 2011.

Gutiérrez, J. J., Escalona, M. J., Mejías, M., Torres, J., Torres-Zenteno, A. H. A Case Study for Generating Test. *Proceedings of IEEE International conference on Research Challenges in Information Science*. pp. 209-214. Morocco, 2008.

JMeter 2012. http://jakarta.apache.org/jmeter/. Last Accessed 02-2012

JUnit. 2012. http://www.junit.org/. Last Accessed 02-2012

Koch, N., Knapp, A., Zhang, G. UML-Based Web Engineering. In *Web Engineering: Modelling and Implementing Web Applications*, Springer. pp.157-191. 2008.

NDT-Suite. http://www.iwt2.org. Last Accessed 02-2012

Pérez, B., Polo, M., Piatini, M. Software Product Line Testing - A Systematic Review. *4th International*

Conference on Software and Data Technologies. (ICSoft 2009). 2009.

OMG, 2008. Documents Associated with Meta Object Facility (MOF) 2.0 Query/View/Transformation. http://www.omg.org/spec/QVT/1.0/. Last Accessed 02-2012

OMG, 2005. Unified Modelling Language: Super structure. Specification. http://www.omg.org/cgi-bin/doc?formal/05-07-04.

Robles, E., Grigera, J., Rossi, G. Bridging Test and Model-Driven Approaches in *Web Engineering. 9th International Conference on Web Engineering. LNCS*. 5648 (2009) pp.130-150. 2009.

Rossi, G., Schwabe, D. Modeling and Implementing Web Applications with OOHDM. In *Web Engineering: Modelling and Implementing Web Applications*, Springer. 2008.

SAP ERP, 2012. www.sap.com/ERP. Last Accessed 02-2012.

SAP BW, 2012. http://www.sap.com/solutions/benchmark/bw.epx. Last Accessed 02-2012