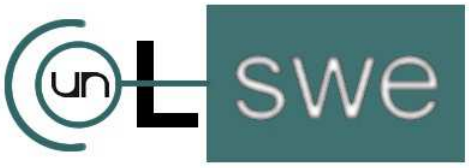

RESEARCH TOPICS IN SOFTWARE EVOLUTION AND MAINTENANCE

Software Engineering Research Group

Universidad Nacional de Colombia

2011

List of Contributors

Angélica Veloza-Suan
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: avelozas@unal.edu.co

Christian Rodríguez-Bustos
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: carodriguezb@unal.edu.co

David Montaño
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: dmontanor@unal.edu.co

Fernando Cortés
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: lfcortesc@unal.edu.co

Henry Roberto Umaña-Acosta
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: hrumana@unal.edu.co

Jairo Aponte
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: jhapontem@unal.edu.co

Juan Gabriel Romero-Silva
Universidad Nacional de Colombia
Bogotá, Colombia

e-mail: jgromeros@unal.edu.co

Laura Moreno
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: lvmorenoc@unal.edu.co

Leslie Solorzano
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: lesolorzanov@unal.edu.co

Mario Linares-Vásquez
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: mlinaresv@unal.edu.co

Miguel Cubides
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: macubidesg@unal.edu.co

Oscar Chaparro
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: ojchaparroa@unal.edu.co

Victor Escobar-Sarmiento
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: vmescobars@unal.edu.co

Yury Niño
Universidad Nacional de Colombia
Bogotá, Colombia
e-mail: yyninor@unal.edu.co

Contents

Chapter 9
Model Driven Development and Model Driven Testing

Henry Roberto Umaña-Acosta, Miguel Cubides

Abstract Software modeling is useful in the specification and comprehension of software requirements. UML specification, for example, helps developers to understand the intended solution with static and dynamic design. In this chapter we explain several software modelling approaches in software development and testing.

9.1 Introduction

ColSWE, the research group in Software Engineering from Universidad Nacional de Colombia, has two main research areas in software modelling: development and test modelling. This has been motivated by the need of supporing the comprehension of the software.

In the software development area, we explain the modelling advantages and disadvantages, and in the software testing area, we focus in writing and generating tests automatically from models. The last technique belongs to one of the wider areas of Model Based Testing. Despite of the high time and effort when developers build models, the productivity of development is increased. This area is promising as in the academic research as in the industry.

Universidad Nacional de Colombia. Software Engineering Research Group - ColSWE
hrumana@unal.edu.co, macubidesgo@unal.edu.co

9.1.1 Challenges of software development

Software development has, amongst others, two identified difficulties: the maintenance and evolution and the comprehension and communication of the different artifacts that compose it.

By the comprehension of the intrinsic evolutive nature of software, it can be demonstrated that along time it has to adapt to its environment, including solutions for new requirements and/or previous requirements modifications. This requires modifications in different aspects that have to be studied and implemented: studied at the level of impact that a modification would have directly as indirectly in the system and implemented by means of making all the modifications that need to be executed as from the impact analysis.

On the other hand, for achieving the development of optimal quality software it is required that the development team totally comprehends each one of the specifications of the different artifacts that describe the previous stages to its point. For example, in the development phase, the specifications in the requirement, specification, requirement and design phases should be perfectly understood. This produces certain difficulties, for example the fact that in the different phases there are also different types of people that work with different levels of expertise and experience, and even with different intellectual and cultural focuses.

9.1.2 Traditional testing process

In the life cycle of a software project, sooner or later the team needs to deal with tests. In the formal way, someone, maybe the analyst or the tester, designs the Test Cases based on the Use Cases or on the functional requirements. Then, the tester executes, step by step each Test Case and compares the result with the expected output and defines if the Test Case was successful or not.

9.1.3 A solution to face the problem

Popular knowledge wisely indicates that "an image says more than a thousand words", it is because of this that for the comprehension, development and communication of the different software artifacts, a specification that allows guiding them graphically has been developed using different tools specialized in modeling to achieve this objective. We have, then, diagrams that represent the software and also what it should do, that support documentation and facilitate communication between parts with different focuses and knowledge.

Also, this diagrams offer software perspective from two different focuses: a static vision and a dynamic one.

9.1.4 Model Based Testing

On the other hand, the Model Based Testing starts defining the SUT, System Under Test, or the parts of the system that need to be tested. Then, the team models this SUT with one of several approaches, and generates the Test Cases from this model. Those Test Cases generated are not executable. They need to be transformed into a script test in some language in order to be executed. For last, the tester analyze the result of the test, which can be: fail or pass, and report to the development team.

This chapter represents an introductory description for the good use of model-driven development (MDD) during software development, initially treating the reason for using MDD in agile methodologies (modeling a specification does not slow software development), continuing with a model-designed development description where its philosophy shall be explained. Following this, UML unified language specification is described, which allows to talk about model-oriented architecture (MDA), its capacities and potential , the use of MDA in the software's design and analysis phases and reviewing of the development eases that some CASE tools offer. Finally, some common errors, in which people fall when it is supposed that MDA will act by itself improving the quality of the developed software, will be described and some conclusions of the work will be specified.

In this chapter we will revise two works in the area of Model Based Testing, one of the research areas of ColSWE.

These works were leaded by the structure and explanation gave by Mark Utting and Bruno Legeard[15]. We will cover two techniques of MBT: One based in Finite State Machines and the other one, based in Pre/Post notations, specifically in OCL. In both cases, we use and evaluate tools, ModelJUnit for FSM and Qtronic (QML) from Conformiq for Pre/Post notation.

9.2 Why to use Model Driven Architecture in agile methodologies

According to Jorbi Cabot [18], there are two points to consider: can agile methodologies benefit from modeling? And, can modeling benefit from agile methodologies?

For embracing the first question, it should be considered that agile methodologies, in spite of an emergent design, maintain modeling use, supporting them in different specifications related to the development team's capacity

and experience. There is the case, for example, of the agile modeling proposed by Ambler Scott [13, 18], where he suggests the use of model rain and iterated necessary modeling, keeping up with a light, but representative model of the system. This suggested model goes from abstract representations of the system to representations of the tests to be done in the product.

The second question is embraced from a point of view based on experience and offered as investigation theme, reaching the proposal of the establishment of a methodology or specification that allows doing modeling as a consequence of the application of characteristics that are common in agile methodologies in the modeling process.

Based on this it can be observed that there is, firstly, the possibility to develop a light model to support development following an agile methodology, and secondly, the possibility to initiate an investigative road that allows implementing new characteristics in the actual existing modeling methodologies which are oriented to classic development methodologies.

9.3 Model Driven Development

9.3.1 Philosophy

A model is a generalized representation of the system. This is achieved by the use of specific and specialized diagrams in each of the different parts of the product.

The use of MDD is suggested for three reasons:

9.3.1.1 Documentation and comprehension of the system

Due to the simplicity and globalization of a model, it is easier to get the general idea that is represented by a diagram than the one represented by a text. It is because of this that documentation is a great support (even though a diagram does not replace a textual documentation) helping the reviewers to comprehend the system's characteristics in a better way.

9.3.1.2 Internal communication and client communication

Once the documentation is complete, this acts as a support for the communication with the client, helping to acquire the initial requirements and the representation of what, as a development team, has been understood related to what the application should do.

On the other hand, communication between different internal divisions in the software team gets easier too, as a diagram is designed in a universal language and no matter the specialty, the team members will understand it easily.

9.3.1.3 Automatic code generation and evolution related facilities

Due to the utilization of computing tools that help software engineering (CASE), code can be generated automatically from a model, which reduces the dedication time of the different resources that intervene in the application's development phase.

With the use of these tools, a reverse engineering can also be accomplished from the code of an existing application, which generates diagrams that model the system and allows the generation of its own documentation.

Another characteristic that facilitates to optimize the model-oriented development is the localization of the affected sectors in the system by means of the change applied. This is seen in the evolution process, where the modification of a requirement or the input of a new one makes a change in the product in such way that it is necessary to dimension this change by making an impact analysis (despite that low coupling development is suggested, there are some cases in which the impact is very high).

There is also an advantage given by the union of impact analysis and self-generating code, from which codification due to evolution automation can be reached.

9.3.2 Unified Modeling Language

The OMG [1] group worked on the specification of a language that permitted creating a model of any system that could be understood universally, for this goal, Unified Modeling Language (UML) was created, which in its 2.0 version specifies more than 10 diagrams that divide between dynamic and static [13].

On this specification case of use, class, package, sequence, communication, state, activity, components, deployment and object diagrams are found (amongst others). With this group of diagrams a system can be represented from a general perspective to a more granular level required.

Through the case of use diagrams the specification of system requirements can be represented. With the classes, package, state, components and deployment diagrams the system is represented in a static way, while in sequence, communication, activity and objects diagrams the system is dynamically represented. To be strict, in this context, dynamic is taken as the way in which

[1] Acronym for Object Management Group, more information `http://www.omg.org/`

the system behaves given certain characteristics, this is because, for example, the object diagram is a representation of the system in a specific state.

One of the great advantages that UML represents is its capacity to be extended through profiles, like Lidia Fuentes exposes it in her investigation [8], by which the generation of a particularized specification and a custom necessity specific development or the company's development politics, can be achieved.

9.4 Model Driven Architecture

Due to the fact that UML has been developed to standardize systems modeling through diagrams, a methodology from MDD that implements UML was born naturally and it is known as model-based architecture (MDA), developed by the OMG group. The difference between both methodologies is that MDD is a generalized methodology and MDA is the one that uses UML.

It regards to tools for automatic coding generation from models, the more evolved of these two methodologies is the one that uses UML standard, i.e. MDA to be clear, just as Oksana proves it in [10].

MDA proposes the creation of a Meta model and a model[1, 4, 7, 11, 12]. The Meta model is a model that is language independent and gives an initial holistic vision of the system, which brings up a first approximation for communication with the stakeholders; while the model is a directed specification to a particular programming language wich permits to automate coding generation tasks, using UML profiles for a greater acurracy and a higher independency of the activities.

9.4.1 Software Engineering

Software creation implies an engineering process that, depending on the implemented methodology, will embrace different phases.

In agile methodologies, despite being managed in a iterative way, planning or analysis, architecture and design, development, testing and revision and implementation phases are contemplated. It is important to point out the fact that, depending on a specific methodology, these phases have greater or lower emphasis, for example XP contemplates an emergent design[17].

How it has been explained, software development will be strengthened and facilitated with the implementation of MDD. Specially, by creating a focused complete engineering in the design and analysis phase, it will be possible, in an earlier manner, to comprehend the product that is wished to create in a better way.

9.4.1.1 The Analysis and Design Phase

Both of these phases are focused to the comprehension of the application to be developed, creating different artifacts to seize for the representation of the different characteristics of the system, what the product has to do and its limits, how it should be developed and which development characteristics will be done.

It is in the design phase where different purpose models are created to analyze how the product will be created, what modules it will contain and in which manner will the modules communicate between them. It is so, that in this phase where representative system models implementation will be strongly used and where UML standard is suggested for obtaining greater facilities in implementation and development (for example code automatic generation).

9.4.1.2 MDA for Software Engineering

Since UML standard provides different diagrams that can represent a system from different focuses[13, 6], it can be used during design to obtain a more complete abstraction easily, which will make comprehend, in earlier production phases, what the system has to do and how to do it.

A system will be more complete as more perspectives it contemplates. For example, a system should be analyzed from the user's point of view (what it should do) as from the performance´s point of view (how it should do it). For this objective, case of use and state diagrams can be used to see what it must do and as object, activity and sequence diagrams to see how it must do it. The first ones allow seeing the system from a static point of view, while the second ones represent a dynamic view.

9.5 What the Model Driven Architecture does not do

A typical error made in software development companies that introduce themselves into model-oriented design, as Bell explains it [2], is to believe that creating an initial system's model will optimize their development practices by itself. It should be considered and clearly understood that model-driven design is just a tool that, well used, will permit obtaining great advantages in software development.

9.6 Notations for modeling tests

In order to build the model, Utting and Legeard [15] give some recommendations:

- Choose only the classes related with the SUT
- Include only the methods to be tested
- Include only the class´s attributes needed to reflect the behavior of the methods

With this scope you may choose some notation for your model. We will explain some of them that were studied in our research.

9.6.1 Transitions-based modeling

We model the behavior of the system as transitions between several states due to events. Usually the model is represented by a Finite State Machine. 9.1

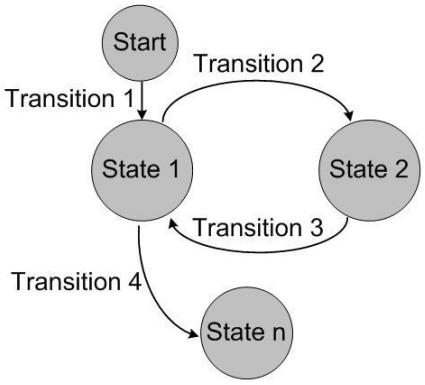

Fig. 9.1: Representing states and state transitions using a State Diagram

9.6.2 Pre/Post modeling

Another kind of system representation is through a series of variables with their respective values in a specific point of time. We model that in the specification of the use cases with pre-conditions and post- conditions. We also can be more formal and write this specification in Z language, Spec#

or OCL (Object Constraint Language). Pre-conditions specify the conditions that must be true before the operations be executed. As an example [5], let see these instructions in OCL:

```
Context Player::calculateFinalScore():: Integer
    Pre: self.isComplete = true
```

The precondition related to the operation "calculateFinalScore()" states that the player has completed the game. Post-conditions specify the conditions that must be true after the operations have been executed. From the same authors, another specification in OCL

```
Context GameEvent::processPlayerChoices():: Integer
    Post: result = 0.
```

According to this example, the post-condition of the operation "processPlayerChoices" states that the player has no choices to play.

9.7 Testing from Finite State Machines

The next sections show the work developed by Miguel Cubides, researcher of ColSWE, and presented as a requisite to get the undergraduate level at the Systems Engineering career in 2009 [14].

9.7.1 FSM and ModelJUnit

ModelJUnit [3] is a plugin that extends the JUnit functionality. His implementation is based in a Finite State Machine as a model. Besides, ModelJUnit gives us several figures about testing process as is presented by Utting [19]

9.7.2 Case Study

We develop a small prototype in order to show the functionality of ModelJUnit. The case chosen is a very basic ATM with two operations: credit, debit and few restrictions. We start modeling the system with a finite state machine. 9.2

Next, we code this model using the classes offered by ModelJUnit.

```
public class PruebasModel implements FsmModel {
        public enum Estados {VERIFICAR_DATOS, MOVIMIENTO, CANCELAR,
TERMINAR};
         private Estados estado;
```

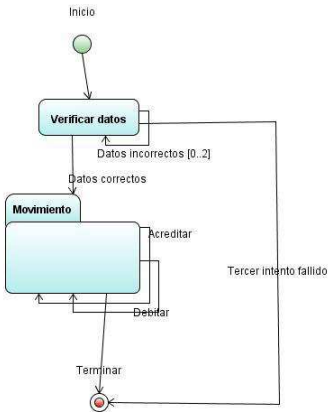

Fig. 9.2: State diagram for the case study. [14]

```
private ControlCajeroTest test;

public PruebasModel() {
test = new ControlCajeroTest();
estado = Estados.VERIFICAR_DATOS;
}

public String getState() {
return String.valueOf(estado);
}

public void reset(boolean testing) {
test.reset(); estado = Estados.VERIFICAR_DATOS;
}

public boolean verificarDatosCorrectosGuard() {
return estado == Estados.VERIFICAR_DATOS;

@Action
public void verificarDatosCorrectos() throws Exception {
    test.testVerificarDatosCorrectos();
    estado = Estados.MOVIMIENTO;
}

public boolean verificarDatosIncorrectosGuard() {
return estado == Estados.VERIFICAR_DATOS;

@Action
public void verificarDatosIncorrectos() throws Exception {
    test.testVerificarDatosIncorrectos();
    estado = Estados.VERIFICAR_DATOS;
}

public boolean debitarGuard() {
```

```
        return estado == Estados.MOVIMIENTO;

        @Action
        public void debitar() throws Exception {
            test.testDebitar();
            estado = Estados.MOVIMIENTO;
        }

        public boolean cancelarGuard() {
        return estado == Estados.MOVIMIENTO;

        @Action
        public void cancelar() throws Exception {
            test.testCancelar();
            estado = Estados.MOVIMIENTO;
        }

        public boolean salirGuard() {
        return estado == Estados.MOVIMIENTO;

        @Action
        public void sali() throws Exception {
            test.testCancelar();
            estado = Estados.MOVIMIENTO;
        }
    }
```

ModelJUnit has a graphical tool that shows us the FSM embedded in the code. Figure 9.3 depicts the FSM of the code for the case study

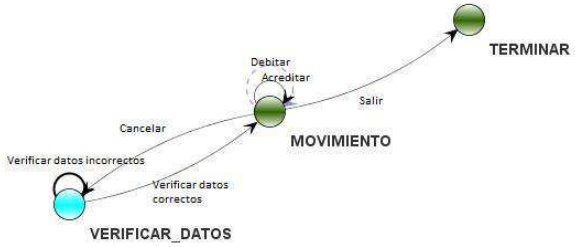

Fig. 9.3: FSM for case study [14]

The tool also offers some features of the tests:

- Number of tests: 10
  - State coverage = 2/3

- Transition coverage = 4/6
- Transition pair coverage = 6/16
- Action coverage = 4/6

The main advantage using the tool is getting the possible sequences of events in the system through the implementation of the Finite State Machine. In other words, we got the Tests Cases running the code.

9.8 Testing from pre/post models

The next sections show the work developed by Luis Alberto Bonilla, researcher of ColSWE, and presented as a requisite to get the undergraduate level at the Systems Engineering career in 2010. Bonilla [3].

9.8.1 Object Constraint Language OCL

The OCL notation is used to do more precise the modeling through UML diagrams. For example, OCL allows us to specify preconditions and post conditions of the operations in a class. Each expression in OCL has a context that usually is the class or method which is belonging to. In table 9.1 there are some important constructors of OCL

Constructor OCL
context class inv: predicate
context class def: name: type = expr
context class::attribute init: expr
context class::method pre: predicate
context class::method post: predicate
context class::method body: expr

Table 9.1: Main OCL constructs [15]

9.8.2 Case study

The system to be modeled is the triangle classificator from Myers's book (2004), [9]

> "The program reads three integer values from an input dialog. The three values represent the lengths of the sides of a triangle. The program displays a message that states whether the triangle is scalene, isosceles, or equilateral."

The context is the Triangle with its three values (length of sides) and a text message with the evaluation as an output.

```
Context Triangle :: kindOfTriangle(a:int, b:int, c:int) : String
```

The precondition is referring to the fact that the integers should be positive

```
pre: a>0 and b>0 and c>0
```

The post conditions validate that the three integers built a triangle where $a + b > c$ for all the possible combinations. Also they contain the rules to classify the triangle as equilateral, isosceles or scalene

```
post: if ( a + b <= c or a + c <= b or b + c <= a) then
        result = "notriangle" else
        if (a=b or b=c or a=c) then
            if(a=b and b=c) then
                result = "equilateral" else
                        result = "isosceles"
                endif
            else
                result = "scalene"
         endif
    endif
```

In this assessment of the OCL utilization as a modeler for generating tests cases, Luis Alberto uses QML from Qtronic [16], a OCL like, that allows to generate the test cases from the specification as you can see in table 9.2

The main advantage of this technique is the coverage. In fact, the algorithm behind the tools is based on Decision Coverage, also known as *branch coverage*.

9.9 Research agenda

Right now, we are conducting our research in another area related to MBT: How can we derive test cases automatically from models that represent GUIs?. In the midterm, we are thinking in covering other challenge: making test executable or the concretization of the abstract tests generated from the model.

Test	Port/Field	Value
1	in / (-1, 0, 0)	out / "badside"
2	in / (0, 0, 0)	out / "badside"
3	in / (1, -1, 0)	out / "badside"
4	in / (2, 0, 0)	out / "badside"
5	in / (1, 1, -1)	out / "badside"
6	in / (1, 2, 0)	out / "badside"
7	in / (1, 1, 2)	out / "notriangle"
8	in / (1, 1, 9)	out / "notriangle"
9	in / (1, 2, 1)	out / "notriangle"
10	in / (1, 9, 1)	out / "notriangle"
11	in / (2, 1, 1)	out / "notriangle"
12	in / (9, 1, 1)	out / "notriangle"
13	in / (1, 1, 1)	out / "equilateral"
14	in / (5, 5, 9)	out / "isosceles"
15	in / (9, 1, 9)	out / "isosceles"
16	in / (9, 9, 1)	out / "isosceles"
17	in / (1, 9, 9)	out / "isosceles"
18	in / (3, 9, 7)	out / "scalene"
19	in / (9, 3, 7)	out / "scalene"

Table 9.2: Test cases generated. Bonilla [3]

On the other hand, in the software modelling area, our research is in two sub-areas: MDD for movile applications and the optimization of data transference applying MDA, focused in the data acces object in a multilayered architecture.

9.10 Summary

Model based software development has three main advantages.

1. It is an excellent support for documenting different artifacts on the development process.

2. It allows people involved in development to easily comprehend what software is and what it is designed for.

3. It makes development more agile by involving it with self generating code tools, easing the process in evolution tasks.

The implementation of model oriented software development methodology enables the analysis of a project from both, dinamic and static standpoints.

One of the most advanced methodological lines in MDD is the one offered by the OMG group: MDA, that, by implementing UML uses MDD with a standardization that allows comprehending and evaluating results easily between different parts.

9.11 References

[1] M. Belaunde, C. Burt, and C. Casanave, *MDA Guide Version 1.0.1*, J. Miller and J. Mukerji, Eds. OMG, 2003.

[2] A. E. Bell, "Death by uml fever," *DSPs*, vol. 2, pp. 11–23, 2004. [Online]. Available: http://queue.acm.org/issuedetail.cfm?issue=984458

[3] L. Bonilla, "Como escribir modelos pre/post adecuados para la automatizacion de pruebas," *Paper presentado en el proceso de trabajo de grado en Ingenieria de Sistemas*, 2010.

[4] A. Caramazana, "Tecnologias mda para el desarrollo de software," in *I Jornada Academica de Investigacion en Ingenieria Informatica*, 2004. [Online]. Available: http://albertocc.tripod.com/

[5] L. B. F. D. Ericksson H., Penker M., *UML 2 Toolkit*. Jow Wikert, 2004.

[6] L. Favre, "Formalizing mda-based reverse engineering processes," in *Australian Software Engineering Conference*, aug. 2008, pp. 153 –160.

[7] M. C. Franky, "Mda: Arquitectura dirigida por modelos," 2010.

[8] A. V. Lidia Fuentes, "Una introduccion a los perfiles uml."

[9] G. J. Myers, *The Art of Software Testing*, 1st ed. John Wiley & Sons, Feb. 1979.

[10] N. P. Oksana Nikiforova, Antons Cernickins, "Discussing the difference between model driven architecture and model driven development in the context of supporting tools," *Fourth International Conference on Software Engineering Advances*, vol. 1, pp. 446–451, 2009.

[11] D. C. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, pp. 25–31, 2006.

[12] ——, "Guest editor's introduction: Model-driven engineering," *Computer*, vol. 39, pp. 25–31, 2006.

[13] A. Scott, *The Elements of UML 2.0 Style*, C. U. Press, Ed. Cambridge University Press, 2005.

[14] H. Umana and M. Cubides, "Pruebas basadas en maquinas de estado finitas (fsm)," *Revista Tendencias en Ingenieria de Software e Inteligencia Artificial*, vol. 4, 2009.

[15] M. Utting and B. Legeard, *Practical model-based testing: a tools approach*. Morgan Kaufmann, 2007.

[16] "Automated test design | Model-Based testing (Conformiq)," http://www.conformiq.com/. [Online]. Available: http://www.conformiq.com/

[17] "XP design and documentation | xProgramming.com," http://xprogramming.com/articles/ferlazzo/. [Online]. Available: http://xprogramming.com/articles/ferlazzo/

[18] "Agile and modeling / MDE : friends or foes? | MOdeling LAnguages," http://modeling-languages.com/blog/content/agile-and-modeling-mde-friends-or-foes. [Online]. Available: http://modeling-languages.com/blog/content/agile-and-modeling-mde-friends-or-foes

[19] "The ModelJUnit test generation tool," http://www.cs.waikato.ac.nz/~marku/mbt/modeljunit/. [Online]. Available: http://www.cs.waikato.ac.nz/~marku/mbt/modeljunit/