Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 83 (2016) 221 – 228

Procedia Computer Science

The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)

Automated Model Driven Testing Using AndroMDA and UML2 Testing Profile in Scrum Process

Meryem ELALLAOUI[a,*], Khalid NAFIL[b], Raja TOUAHNI[a], Rochdi MESSOUSSI[a]

[a]*Department of Physics, Faculty of Sciences, Ibn Tofail University, Kenitra 14000, Morocco*
[b]*Faculty of Economics, University Mohammed V, Rabat 1000, Morocco*

Abstract

Software testing is an important step in the life cycle of agile development; it represents an efficient way to ensure the good functioning of the product. But as the complexity of a system increases, the effort and expertise to test it also increases. To significantly reduce these efforts, and reduce the cost and time; several studies have been carried out and various tools and test automation techniques have been proposed. In this paper, we present an approach to automatic generation of test cases from UML 2 Models at the Scrum agile process. This approach automates two important steps: the transformation of design models into test models and generating test cases, based on an open source MDA framework.

Peer-review under responsibility of the Conference Program Chairs

Keywords: Testing; Model-Driven Testing; UML2; UML2 Testing Profile (U2TP); MDA

1. Introduction

Currently, many software development companies have migrated to the agile methodology to overcome various problems encountered during development. Testing, is considered as a necessary step in this process to ensure the good functioning of the product, but the difficulty of this task depends on the complexity of the system under test, more the complexity increases more development cost also increases, which also requires a lot of effort, time and expertise. However, such an increase in terms of cost, time and effort is not feasible.

* Corresponding author. Tel.: +212-642-640-814.
E-mail address: elallaoui.meryem@univ-ibntofail.ac.ma

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2016.04.119

To face these challenges, new development and testing techniques must be applied. The modeling language UML2[1] has emerged as a standard for modeling software systems, which also applies in designing test cases. The Model Driven Engineering[2] (MDE) promotes the use of models, which are placed at the core of the development process, to cope with the growing complexity of the design and production software, and to provide solutions to difficulties of interoperability between systems. The Model Driven Architecture[3] (MDA) is the most famous MDE approach, which aims to organize the Model-Driven Development layers. Furthermore, the Model Based Testing (MBT[4]) is a test automation technique, which aims to improve the quality and efficiency of testing, and reduce the cost. The U2TP[5] profile (UML 2.0 Testing Profile) allows designers and testers to communicate with the same language. It provides an efficient way to use both UML for modeling the system and for specifying the test. Scrum[6] is a management methodology, improvement and maintenance of an existing system or production prototype. It is based on dividing a project into iterations called "sprints". The duration of sprint is fixed in advance and is normally between two weeks and one month[7].The main artefact in Scrum[8] is the product backlog, which represents a prioritized features list, this list is specified by the product owner, who creates, controls and manages the Backlog.

The advantage of Scrum is its ability to respond to changing with some flexibility, allows modifications of the projects and deliverables at any time, in order to deliver the most appropriate release[7].

The approach proposed in this paper discusses two main issues: model transformation into another model (M2M) and the transformation of model into the code (M2T)[9]. The first transformation takes as input the UML sequence diagrams and generates U2TP sequence diagrams. The second transformation takes as input U2TP sequence diagrams and generates test cases. These transformations are performed using two cartridges that we have implemented for the open source AndroMDA[10] Framework.

Our paper is organized as follows: After discussing some related work in section 2, section 3 presents our model transformation approach. The proposal is applied to an example from literature in section 4. Finally, section 5 concludes the paper.

2. Related work

Related work can be divided into two categories: Model Based Testing in Agile Development and automatic generation of test cases from UML sequence diagrams.

There are some propositions on the integration of Model Based Testing in agile development. Katara and Kervinen[11] introduce an approach to adapting practices MBT in software development organizations using agile processes. The methodology is built around a domain specific modelling language with action words and keywords. They use labelled transition systems (LTS) for behavioural modelling which allow a complete functional specification. They also defined a test covering language.

Rumpe[12] suggests the use of UML models as the primary artefact for requirements and design documentation, code generation and development of test cases. Model transformation as concepts focused for agile processes especially for testing activities. However, the proposal does not introduce any tool for the approach, neither a concrete technique on how to manage the challenges of Scrum.

Renate Löffler and al.[8] propose a technique for improving the documentation and communication of customer requirements using UML diagrams, and automation of test activities by generating test scripts from the test model in the Scrum process. They used the sequence diagrams and interaction overview diagrams as a modelling notation, and as test models by testers for generating test scripts.

Several techniques have been made for generating test cases, either from the use cases such[13], they introduced a new approach to generate test cases from a need expressed by use case. Information about the test have been modeled by the test profile UTP. Others are based on UML sequence diagrams: Javed[14] proposes a method of generating unit test cases from the PIM using MDA tools. They performed a horizontal transformation using Tefkat and a vertical transformation using MOFScript. For the generation of test cases from a sequence of method calls, they have implemented a prototype tool.

Shanthi and al[15] propose an approach of generating test cases from UML sequence diagrams based on a genetic algorithm. The best test cases are optimized and validation of test cases is performed by priority.

Panthi[16] defined a method to generate test cases from UML sequence diagrams. This technique consists in a first step to construct sequence diagrams, and then convert them into graphical sequence; the graph is traversed to select

the predicate functions. Then this predicate are transformed into source code. From this code, they built extended finite state machine (EFSM). Finally, they generated test data corresponding to transformed predicate functions.
Baker and al.[17] define the test models using the profile UTP, and they performed the transformation manually. Dai[18] describes a set of concepts for UTP models from UML models. The test models can be transformed either directly to obtain the test code or to have a model (PST). Mussa[19] discuss a survey of 15 different tools, approaches and techniques for MBT. A most of this techniques focuses on the approaches for automatic generation of test cases from UML models. The study shows the benefit of using MBT for various application fields, but an implementation example in the surveyed approaches is missing.

3. Model Driven testing using AndroMDA and U2TP in scrum process

The model driven testing (MDT) approach that we present automate the generation of test cases at the Scrum agile process, using the transformation model based on the idea of Dai[20]. The first step is to choose user stories from a list of requirements, then construct appropriate Platform-Independent Model (PIM). The second step will be to perform a transformation of design model (PIM) into another test model (PIT). This transformation is done via the implementation of a cartridge for AndroMDA Framework. An AndroMDA cartridge is a collection of source code template files and Java helper classes (called Metafaçades) packaged into a .JAR file.

The cartridge takes as input sequence diagrams as design model, these models are automatically transformed into test models U2TP. Then the implementation of a second cartridge which takes as input PIT models and automatically generates test cases.

The advantage of the integration of MDT is its ability to link requirements to design, and facilitate to developers the generation of the results in a common language that combines the participants in the design process. This allows clear communication of design and improving workflow. Also, gain time because it allows developers to perform tests in an identical language to that they design in UML.

In the next section, we will define the important elements for the implementation of the cartridge, which transforms designs models in test models, but before we start, we situate and briefly present the features currently available in AndroMDA.

AndroMDA[21] is code generation framework. It generates code for any desired target platform from UML models. AndroMDA provides several cartridges for different platforms and technologies such as Java, Spring, Web Services, etc. It accepts only subsets of UML diagrams as input; these diagrams are use cases, activity diagrams and class diagrams.

We chose AndroMDA because it is an effective Framework in the generation of applications from design for the static aspects, and we believe that with the improvement of the generation of test cases for the dynamic aspects, we will have a powerful tool to generate both systems under test and test cases.

3.1. UML 2.0 Testing Profile

The U2TP stereotype[5] implemented in our approach are:

- **SUT**: is applied to the attributes of a class (TestContext) to form the system under test (SUT). The characteristics of this system are provided by the type of attributes to which the stereotype is applied.
- **TestComponent**: It is a component of the test environment. His role is directing test cases by stimulating the system under test. It applies to class.
- **TestContext**: Includes a set of test cases which have the same context. It contains test case as behaviors especially interactions, and applies to all classifier that may contain behaviors.
- **DetermAlt**: Means the combined fragments where the operands are evaluated exactly in the same order. This stereotype is applied on the combined fragments.
- **TestCase**: Means the behavior that describes how the testComponent interacts with the SUT to achieve the objectives of this test. A test case should return a verdict that decides if the test failed or not. This verdict is generally rendered following a validation action. It is applied to the interactions.
- **ValidationAction**: is used to fix the verdicts in tests behavior. It is applied to the CallOperationAction element of our model.

- **DataPool**: constitutes an explicit data container used in test cases. It is applied to a class and on the properties or attributes.
- **DataSelector**: Means the operations that select data from the DataPool. Applied to the operations of the DataPool class.

3.2. Test cases generation from test model (M2T)

In this section we will detail the important elements for the construction of our cartridge (m2t). The figure below shows the necessary Metafaçades for the generation of test cases from U2TP sequence diagrams.

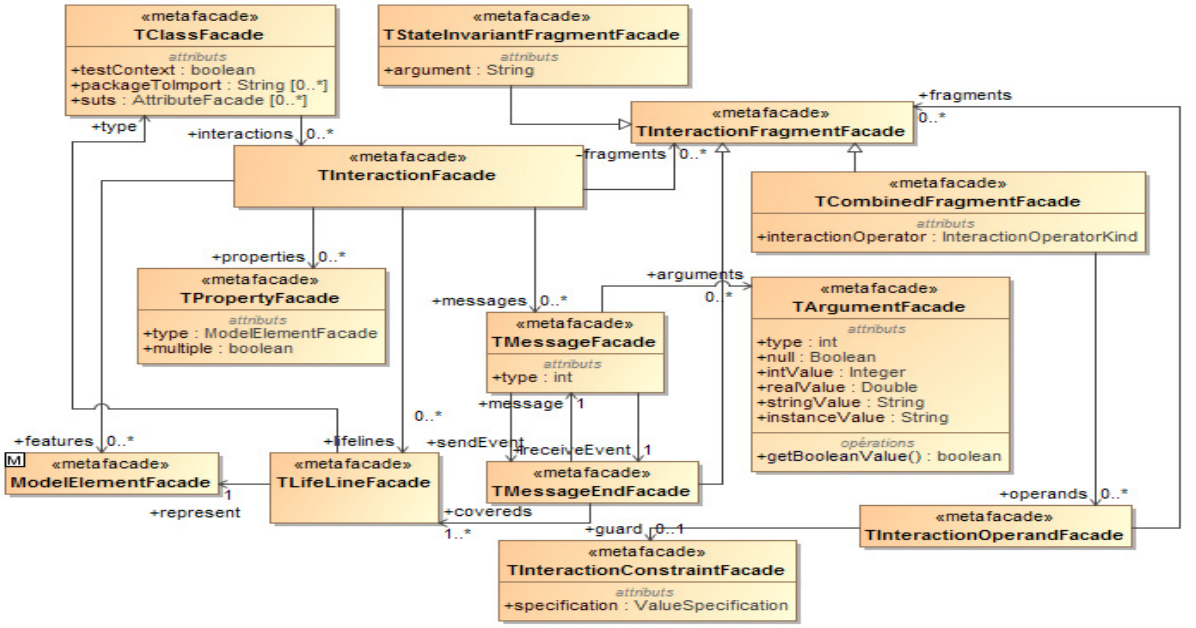

Fig. 1.Sequence diagram meta-model

TClassFacade in this case represents a class of UML model. TInteractionFacade models the behavior of this class. TInteractionFragmentFacade, TMessageFacade and TLifeLineFacade represent the basic elements in a UML sequence diagram. The LifeLines are specific, they communicate via messages. Each message triggers two events: receiveEvent and sendEvent represented here by TMessageEventFacade. Either each LifeLine corresponds to a property of interaction or a class attribute. A message can have several arguments (TArgumentFacade) and a type that specifies the type of message (synchronous, asynchronous or creation). TArgumentFacade models the arguments. TStateInvariantFacade represents StateInvariant expressing validating a test via the stereotype (ValidationAction). TCombinedFragmentFacade models the combined fragments. A combined fragment represents the joints interactions. It is defined by TInteractionOperandFacade that allow us to determine the nature of the Alt, Opt, and Loop; and one or more operands which contain a set of events or other combined fragments. Each operand contains a guard TInteractionConstraintFacade type that describes the condition for this operand. So a fragment in this case is either a TStateInvariantFacade, a TCombinedFragmentFacade or TMessageEndFacade. All these fragments represent the behavior of a UML sequence diagram.

3.3. Test model generation from design model (M2M)

In this transformation, sequence diagrams that describe the SUT represent scenarios. The test model generated from the AndroMDA transformation is in the form of U2TP sequence diagrams. Two bases meta-models are needed for the implementation of the cartridge (M2M): Metafaçades and PSMClass.

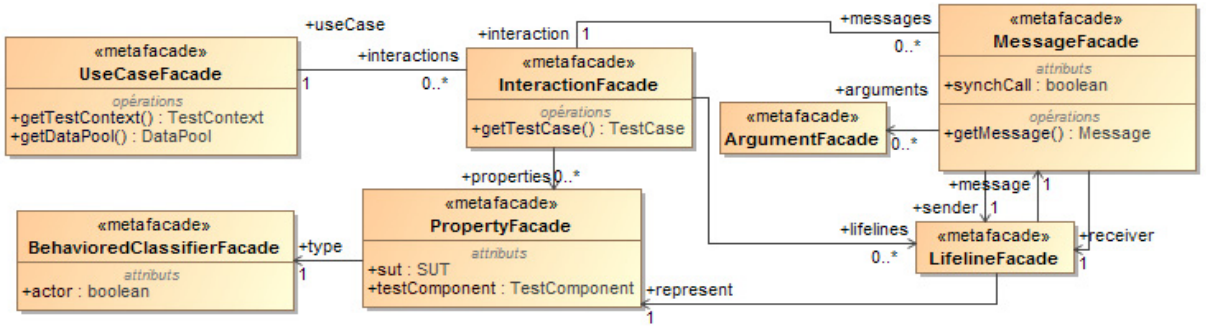

Fig. 2.Metafaçades representing the input of cartridge

Each message contains a StateInvariant. MesageOccurencespecification, ExecutionOccurenceSpecification and BehaviorExecutionSpecification represent fragments that allow binding between messages and LifeLines.

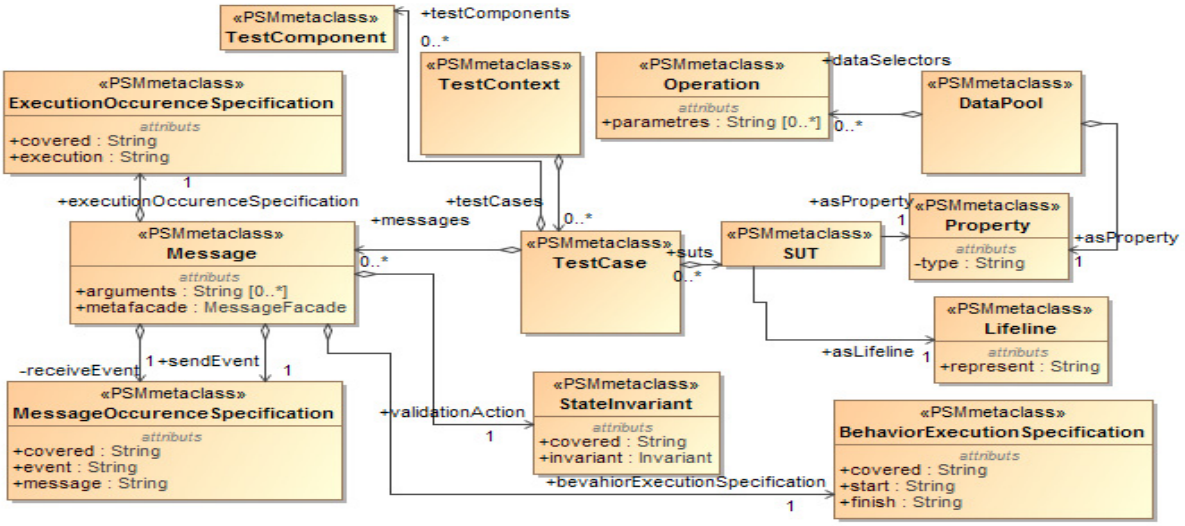

Fig. 3.PSMClass representing the output of the cartridge

For utilities, we have implemented four java class which facilitate the generation. The MetafaçadeUtils class that recovers all the actors of the model, TestComponent, extract the testComponent from a set of properties and extract SUTs from a set of properties (attributes of TestContext). The XmiIdGenerator method returns the instance of the class, which generates ids, and CreateLifeline method that handles the creation of LifeLines from a property. The role of class MessageUtils is the creation of DataSelector for a message via getDataSelectorMessage method, and NameUtils class that is used for formatting names. The XmiIdGenerator class generates an XMI id for an element.

For Templates, we need only one for the generation of the output model. It contains three packages, each one has class and stereotypes.

4. Case study

To illustrate our method, we have applied an example from[22] that represents the user login functionality of a system. Figure 4 shows the main scenario of the use case "Login". The user enters his username and password, then the system checks the data entered, if they are valid the system creates a new session for that user. Figure 5 illustrates the test cases generated to test the functionality of Figure 4.
Figure 6 shows the TestNG test code generated for the "Login" scenario. The setUp method (line 13) is executed first and therefore it initializes all SUTs, if that is services, they load them
since the Spring context or Ejb, then testing methods in order. TestContext (line 08) is the base class for TestContext. TestNG automatically invokes the test cases. The values of uid and psw (lines 26 and 27) are recovered from the DataPool via DataSelector "ds1_loginUser (), ds2_loginUser ()".The result returned by the SUT (line 29) and the result obtained from the DataPool (line 28) are compared (line 30).

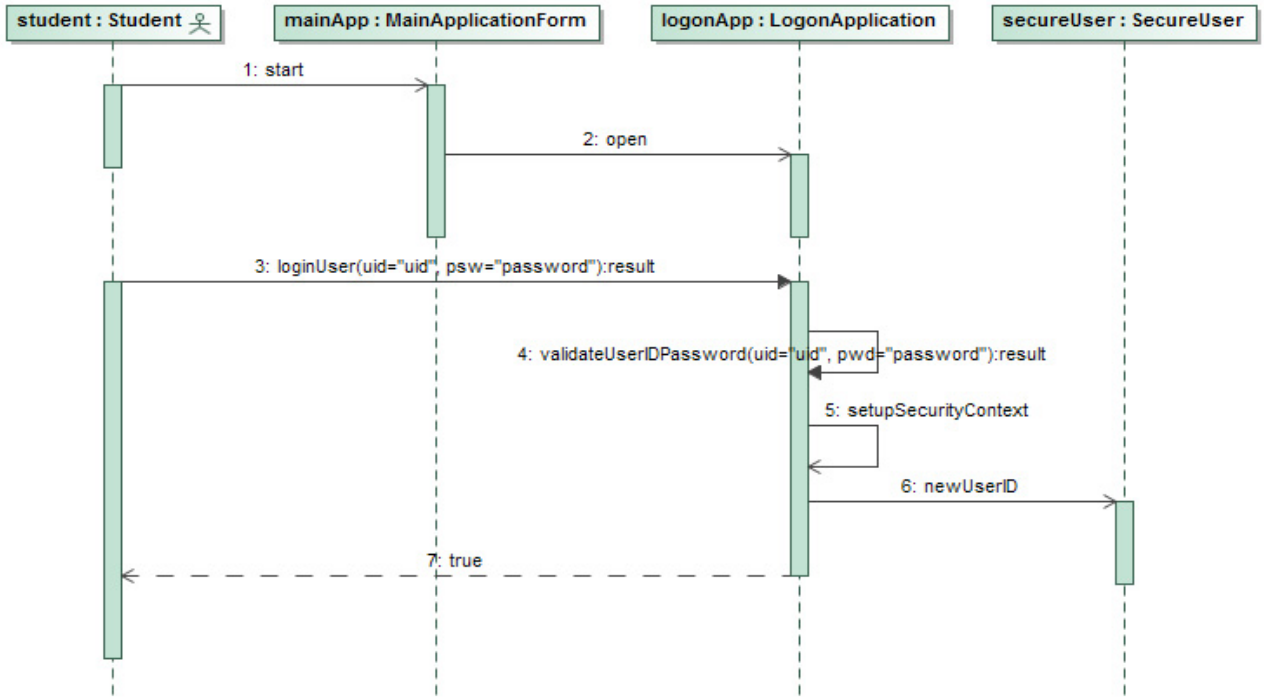

Fig. 4.UML sequence diagram for the use case "Login" main scenario, taken from [22]

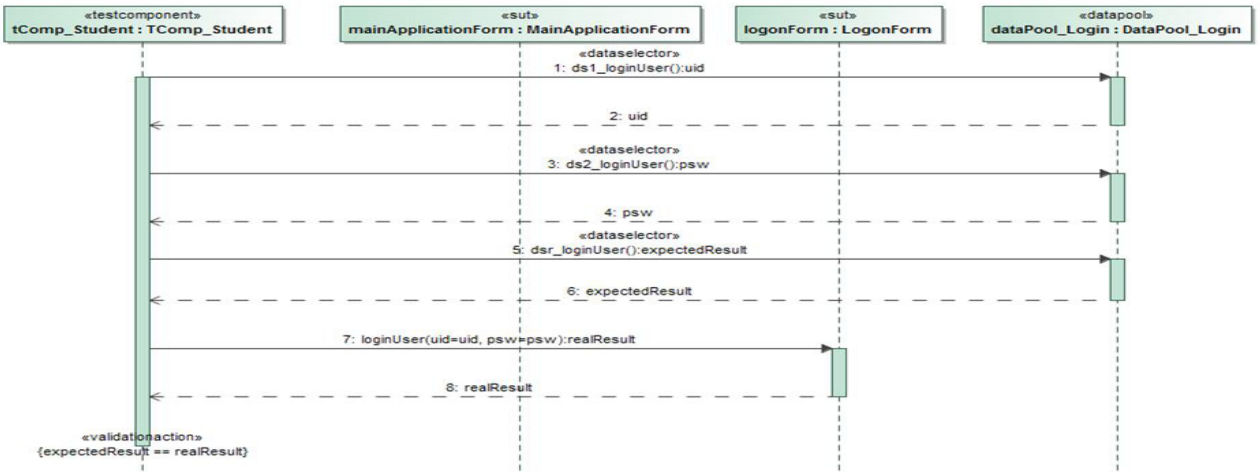

Fig. 5.Test case generated for "Login" main scenario

```
2   import java.util.*;
3   import org.junit.Assert;
4   import org.sample.LogonForm;
5   import org.sample.MainApplicationForm;
6   import org.sample.test.DataPool_Login;
7   ...............*/*import*/
8   public class LoginTest extends TestContext{
9       private MainApplicationForm mainApplicationForm = null;
10      private LogonForm logonForm = null;
11      private DataPool_Login dataPool_Login = null;
12      @BeforeClass
13      public void setUp() {
14          // Initialize logger
15          this.logger = LogFactory.getLog(LoginTest.class);
16          // Initialize others suts
17          this.mainApplicationForm = new MainApplicationForm();
18          this.logonForm = new LogonForm();
19          this.dataPool_Login = new DataPool_Login();}
20      @Test
21      public void testLogin(){
22          String uid = null;
23          String psw = null;
24          boolean expectedResult = null;
25          boolean realResult = null;
26          uid = dataPool_Login.ds1_loginUser();
27          psw = dataPool_Login.ds2_loginUser();
28          expectedResult = dataPool_Login.dsr_loginUser();
29          realResult = logonForm.loginUser(uid, psw);
30          assertTrue(expectedResult == realResult); }}
```

Fig. 6.TestNG automatically generated test code

5. Conclusion

In this paper, we proposed an MDA approach based on the transformation of the models at the Scrum agile process, using standardized meta-models such as U2TP (UML 2.0 Testing Profile) and UML 2.
Our methodology shows how to get a test model U2TP from a UML 2 design model, and how test cases are generated from this model U2TP using AndroMDA Framework.
We believe that this method facilitate the work of developers and testers, using two cartridges that we have developed, we can generate test code, test models and also the system under test.
We wrote this paper to show the first results, the next step in this research will be the validation of the proposal on real projects companies and comparison

References

1. Unified Modeling Language, http://www.omg.org/spec/UML/2.0/
2. IEEE, IEEE Standard Glossary of Software Engineering Terminology, Tech. rep., IEEE, 1990.
3. Model Driven architecture, http://www.omg.org/mda/
4. M. Utting and B. Legeard. Practical Model-Based Testing: A Tools Approach. Morgan Kaufmann, 2007.
5. Uml Testing Profile, http://www.http://utp.omg.org/
6. Ken Schwaber and Mike Beedle. Agile Software Development with Scrum. Prentice Hall, Upper Saddle River, 2002.
7. Business InterActif.fr, livre blanc: Extreme Programming - méthodes agiles: état des lieux.
8. Löffler, R., Güldali, B., Geisen, S.: Towards Model-based Acceptance Testing for Scrum. Software-technik-Trends, GI, 2010
9. Lamancha, B. P. et al., 2013. Automated generation of test oracles using a model-driven approach. Information and Software Technology, 55(2), pp. 301- 309.
10. AndroMDA, http://www.andromda.org/
11. Mika Katara and Antti Kervinen. Making Model-Based Testing More Agile: A Use Case Driven Approach. In Haifa Verification Conference, pages 219-234, 2006.
12. Bernhard Rumpe. Agile test-based modeling. In Software Engineering Research and Practice, pages 10-15, 2006.
13. J. J. Gutíerrez, M. J. Escalona, M. Mejıas, and J. Torres, "An Approach to Generate Test Cases from Use Cases," in Proceedings of 6th International Conference on Web Engineering, Palo Alto, CA, 2006, pp.113–114.
14. A. Z. Javed, P. A. Strooper, and G. N. Watson. Automated generation of test cases using model-driven architecture. In AST'07, page 3, Washington, DC, USA, 2007. IEEE Computer Society.
15. A. V. K. Shanthi, and G. MahanKumar, "Automated Test Cases Generation from UML Sequence Diagram" in 2012 International Conference on Software and Computer Applications.
16. Panthi, V. (2012). Automatic Test Case Generation using Sequence Diagram, 2(4), 22–29.
17. P. Baker, Z. Dai, J. Grabowski, I. Schieferdecker, O. Haugen, C. Williams, Model-Driven Testing: Using the UML Testing Profile, Springer, 2007.
18. Z. Dai, Model-driven testing with UML 2.0, Canterbury, England, 2004.
19. M. Mussa, S. Ouchani, W. Al Sammane, and A. Hamou-Lhadj, "A survey of model-driven testing techniques," in 9th International Conference on Quality Software (QSIC '09)., Aug. 2009, pp. 167 –172.
20. Dai, Z. Model-Driven Testing with UML 2.0. 2nd European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations. 2004.
21. Sven Jörges. Construction and Evolution of Code Generators: A Model-Driven and Service-Oriented Approach, Springer, 2013
22. Basanieri, F., A. Bertolino, y E. Marchetti, The cow suite approach to planning and deriving test suites in UML projects. Lecture notes in computer science, 2002: p. 383-397.