# Introducing Usability Requirements in a Test/Model-Driven Web Engineering Method[1]

Esteban Robles Luna[1,2], Julián Grigera[1], Gustavo Rossi[1,2], José Ignacio Panach[3], Oscar Pastor[3]

[1]LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
{esteban.robles, julian.grigera, gustavo}@lifia.info.unlp.edu.ar
[2]Also at Conicet
3Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{jpanach, opastor}@pros.upv.es

**Abstract.** The success of web applications is constrained by two key features: usability and fast evolution. Current web engineering approaches follow a "unified" development style which tends to be unsuitable for applications that needs to evolve fast. In this paper, we show how to address usability requirements in an agile test/model driven web engineering method. Usability requirements are contemplated from the very beginning of each cycle by creating a set of meaningful tests that drive the development of the application and ensure that no functionality is altered unintentionally through development cycles. The approach is illustrated with an example in the context of the WebML / WebRatio suite.

## 1 Introduction

It is not new that Web applications require short development cycles with constantly changing requirements, and must also be extremely usable to satisfy customers. The success of this kind of software strongly depends on fulfilling these two conditions. This combination constraints the current trend towards Model-Driven Web Engineering (MDWE) approaches [13, 3, 7, 9, 20] which include automatic or semi-automatic code derivation from conceptual models, thus minimizing coding errors and making the software development faster. However, MDWE approaches tend to use “unified” rather than agile styles for development, lacking the appeal of other approaches like extreme programming (XP) [11] or Test-Driven Development (TDD) [2].

TDD uses small cycles to add behavior to the application. Each cycle starts by gathering requirements in the form of use cases [10] or user stories [11] that describe the expected behavior of the application informally. Next, the developer abstracts

[1] This work has been developed with the support of MEC under the project SESAMO TIN2007-62894 and co-financed by FEDER

concepts and behavior and concretizes them in a set of meaningful test cases. Those tests should fail on their first run, showing that the application does not meet the requirements yet. In order to fix this problem, the developer writes the necessary code to pass the tests and runs them again until the whole test suite passes. The process is iterative and continues by adding new requirements. In these cycles, the developer can refactor [8] the code when it is necessary. Studies have shown that TDD reduces the number of problems found in the implementation stage [18] and therefore its use is growing fast in industrial settings [14].

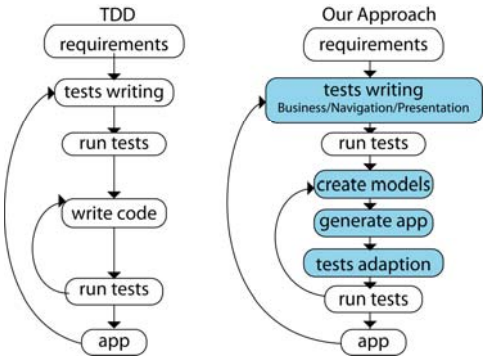



**Fig 1:** Summary of the approach

We have recently defined an approach which injects a test-driven development style into a model-driven development life cycle [19]. The basic idea is to apply the principles of TDD to a MDWE approach. In this manner, tests are run over the application, but if they fail, changes are applied to the models and not to the code. The application can be generated again from these models thanks to automatic transformations, and tests can be run again, continuing with the cycle. We have also defined the concept of navigation unit testing to extend the well-known concept of unit testing to the navigation realm. Navigation Unit tests check that a User Interaction Diagram (UID) is satisfied in the application by testing an interface mockup first and the derived prototype later. A summarized schema of the approach confronted with "conventional" TDD is presented in Fig 1.

In this paper, we show how to deal with usability requirements in the approach described above. Usability requirements are requirements that capture the characteristics to build a usable system for the user. We illustrate the idea with two usability requirements that have functional implications, in other words, usability requirements that affect the system arquitecture. Following requirement guidelines of the literature [12], a list of usability characteristics that must be considered in an MDWE process are identified and a set of test cases are generated from them. These tests will lead the modeling of usability requirements in the same way that traditional TDD leads the coding phase of functional requirements. The main contributions of the paper are the following:

- We present a systematic way to deal with usability requirements incrementally.

- We propose using black box interaction tests as essential elements for driving the development phase and validating usability requirements in a web application.

The structure of the paper is the following: In Section 2 we summarize the background of our proposal. In Section 3 we present our approach, and using a proof of concept we explain how we map usability requirements into test models, and how the cycle proceeds after generating the application. In Section 4 we briefly review some related work and in Section 5 we conclude and present some further work.

## 2 Background

Our proposal synergizes two different research approaches: first, the specification of usability requirements in a Model-Driven Development (MDD) schema and second the introduction of TDD in MDWE approaches. We briefly discuss the two of them:

### 2.1 Specifying Usability Requirements

There is a type of usability recommendations that are related to the system architecture. These recommendations involve building certain functionalities into the software to improve user-system interaction. A big amount of rework is needed to include these features in a software system, unless they are considered from the first stages of the software development process. The idea of dealing with usability from the early stages was developed by authors as Bass [1]. Those works propose including the usability when the system architecture is designed.

The best way of ensuring that usability is taken into account in the architecture design is dealing with it from the requirements capture step. These requirements lead the architecture design. In the literature, there are several works to capture usability requirements. For example, by means of i* models [28] or by means of the concept called usability patterns [21]. According to the proposal of Bass, we have proposed in previous works [17], a method to include usability features with functional implications in any MDD approach. The idea is to include new Conceptual Primitives in the Conceptual Model that represents all usability features.

### 2.2 Bridging TDD and MDWE

Model-Driven approaches favor the construction of software by handling abstract models, thus raising the abstraction level over plain source code writing, and leading to less error-prone applications. On the other hand, agile approaches and their iterative, short-cycled way of development, result in a very appealing methodology when it comes to cope with fast change and evolution; both typical of web applications. We propose a combined methodology that takes the best of both approaches, by introducing MDD as part of a TDD cycle. The process has the same structure as TDD, but several artifacts have been adapted/added to fit in MDD approaches. The main differences with traditional TDD include: using models (for business, navigation and presentation) instead of code writing, gathering requirements with HTML mockups to improve communication with stakeholders and developing tests to drive the model development using a black box testing approach taking advantage of mockups. The

approach includes automated tests that validate functional and usability requirements and deal with Web refactoring interventions.

## 3 An Overview of Our Approach

Our approach is based on tests that are written before the application is developed. Tests specify usability requirements in advance so they drive the development phase and later validate that the application satisfies them. As a first step, we partially capture user requirements (Sections 3.1 and 3.2), focusing on expected behavior of the application as well as usability concerns. We next state these requirements as tests (Section 3.3), and since they are written *before* the application is developed, we specify them against UI mockups (stub html pages used to convey application's aspects with the stakeholders). A running application is then derived using a MDD tool (Section 3.4), and tests are run against that application. Because tests are written against mockups, they may not pass due to small differences between the XHTML mockups and the markup generated by the MDWE tool. Consequently, tests must be adapted to reflect the final generated layout (Section 3.5). Some usability aspects may appear after the application has been developed or even deployed. Those changes involve dealing with existing functionality which may involve refactoring (Section 3.6). As in TDD, the whole method is repeated with all use cases until a full-featured prototype is reached.

We illustrate the approach using *TDDStore*, a simplified online bookstore. Specifically, we will validate some usability requirements in the checkout process. This process will be carried out by the user in order to finish a purchase. In the following subsections we illustrate the checkout process from capturing requirements to validate the usability requirements with tests.

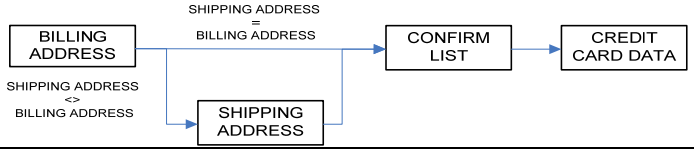
### 3.1 Introducing Usability Requirements

Our work focuses on usability requirements with impact on the architectural design. This set of usability requirements are referred to as *Functional Usability Features* (FUFs) [12]. Examples of these FUFs are providing cancel, undo and feedback facilities. Each FUF has a set of subtypes called *Usability Mechanisms***.** Each Usability Mechanism is a different variety of the usability feature. For example the FUF called *Feedback* is composed by several Usability Mechanisms, for example: (1) *System Status Feedback*: it informs the user about the internal system state; (2) *Interaction Feedback*: it informs the user that his request is being processed.

In order to capture the requirements for the Usability Mechanisms, we have used a set of guidelines defined by Juristo [12]. This approach consists of packaging guidelines that empower developers to capture usability requirements without depending on a usability expert. These guidelines help developers to understand the implications of usability requirements in system architecture and know how to elicit and specify usability features for a system. They have served as a basis for identifying which issues should be discussed with stakeholders during the elicitation process.

The checkout process is a somewhat complex process that needs to be followed by the user. Billing address, delivery address, a summary of the purchase and person details must be fulfilled in order to finish the purchase. A well known usability mechanism called *Step by step* could be applied to break a big, clumsy form, into small and easy to understand steps. This mechanism helps the user to do tasks that require different steps with user input. The application of Step by step results in a wizard. From Juristo's FUF guidelines [12], we have extracted the usability characteristics that the analyst must specify in the conceptual model in order to specify the functionality of the mechanism. Each usability characteristic represents a system property that must be specified by the analyst. Table 1 shows the characteristics that the analyst must specify for the Step by step mechanism and their value in the checkout example

**Table 1.** Usability requirements for Step by step and Abort operation

| **Step by step** | |
|---|---|
| **Characteristic** | **Value specified by the analyst in the checkout example** |
| Service selection | This mechanism will be applied to the *checkout action* |
| Steps division |  |
| Step description | Each step must contain a short description |
| Visual aspect | The user has specified the widgets to fill in each step |
| Remaining steps | The system must inform about the number of remaining steps |

### 3.2 Modeling Functional Requirements

In order to gather navigation, business and usability requirements from our stakeholders, we use two kinds of artifacts: User Interaction Diagrams (UIDs) and UI mockups. UIDs serve as a partial, high-level navigation models, providing abstract information about interface features. On the other hand, UI mockups help to agree with the client on broad aspects of the application look and feel. This is a very convenient way for interacting with stakeholders and gathering quick feedback from them. We also gain two additional benefits from UI mockups: we can perform our usability specifications tests against them and they will be used to create the application's UI. In Fig 2 we show two mockups of the checkout process:

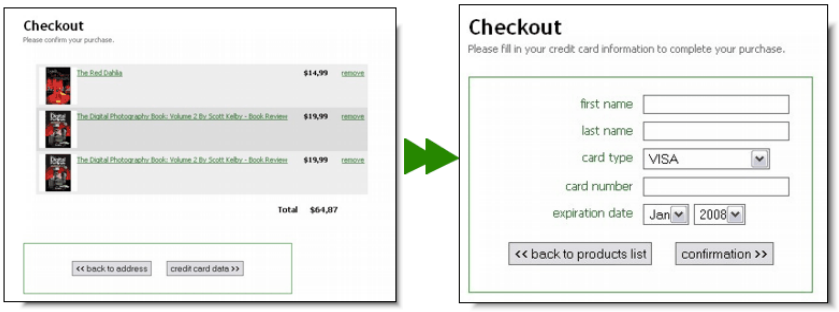



**Fig 2.** UI mockups for steps 2 and 3 of the checkout process.

### 3.3 Defining Tests

Following our approach, usability requirements should also be specified as tests that ensure usability application. These tests will validate usability requirements at any stage of the application development. Moreover, they help during application growth, ensuring that usability characteristics are not altered. To illustrate this stage, we will take the earlier mentioned requirement (divide checkout into steps) as example. In this test, we follow the checkout process filling each node with the necessary information and making assertions about UI and node elements. Using Selenium [22] on Java, the following test validates that the checkout process is divided into the steps previously mentioned (Section 3.1 and 3.2). As all MDWE tools derive XHTML/CSS/ Javascript code, Selenium code is agnostic of the MDWE tool used. This test will drive the development phase of this usability requirement and it looks like we show next:

```
    public class CheckoutTestCase extends SeleneseTestCase {
      public void testSuccessfulCheckout() throws Exception {
(01)    selenium.open("file:///dev/bookstore/Mockups/books-list.html");
(02)    selenium.clickAndWait("/ul[@id='products']/li[1]/div[1]/div[@id='product-info']/a");
(03)    selenium.assertLocation("/cart*");
(04)    assertEquals("The Digital…", selenium.getText("/ul[@id='selected-products']/li[1]/span[1]"));
(05)    selenium.clickAndWait("checkout");
(06)    selenium.assertLocation("/checkoutStepShippingAddress");
(07)    selenium.type("shipping-address", "Calle 58"); selenium.select("country", "label=Argentina");
(08)    selenium.clickAndWait ("//input[@value='product confirmation>>']");
(09)    selenium.assertLocation("/checkoutStepBillingAddress");
(10)    selenium.type("billing-address", "Calle 48”); selenium.select("country", "label=Argentina");
(11)    selenium.clickAndWait ("//input[@value='product confirmation>>']");
(12)    selenium.assertLocation("/checkoutStepProductConfirmation");
(13)    assertEquals("The Digital…", selenium.getText("/ul[@id='selected-products']/li[1]/span[1]"));
(14)    selenium.clickAndWait ("//input[@value='credit card data >>']");
(15)    selenium.assertLocation("/checkoutStepCreditCardData");
(16)    selenium.type("first-na", "Esteban"); selenium.type("last-na", "Robles Luna");
(17)    selenium.type("card-number", "4246234673479");
(18)    selenium.select("exp-year", "label=2011"); selenium.select("exp-month", "label=Apr");
(19)    selenium.clickAndWait ("//input[@value='confirmation >>']");
```

```
(20)      selenium.assertLocation("/checkoutSucceed");
(21)      assertEquals("Checkout succeed", selenium.getText("/div[@id='message"));
      }
}
```

The test opens the book list page (1) and adds an item to the shopping cart (2). Then we assert that the book has been added and proceed to the checkout process (3-5). Shipping information (6-8) and billing information (9-12) are filled and confirmed. Products are confirmed by asserting that product's name (13-15). Credit card data is filled (16-19) and then we confirm the process has succeeded by looking at the text displayed in a *div* element (20-21).

### 3.4 Deriving Design Models

In order to generate incremental prototypes of the application, we have used WebRatio [23], a WebML-based MDWE tool. We have specified the different models (business, navigation and presentation) that will allow to derivate the application. In order to show the specific aspects of our approach, we focus mainly on the navigational (hypertext) model, being the distinctive model in Web applications. Besides, we want to emphasize the differences between typical TDD and TDD in MDWE applications and show how changes in usability requirements may affect navigation.

A first data model is derived using the UIDs as a starting point, identifying the entities needed to satisfy the specified interactions, e.g. by using the heuristics described in [20]. As Web Ratio supports the specification of ER models at this stage of the development, the application behavior will be specified later, in the so-called logic model. As for the navigational model, we show it with the checkout example. According to the test written in the previous section, we need to create a step-by-step checkout. Fig 3 shows this interaction in a WebML interaction model.

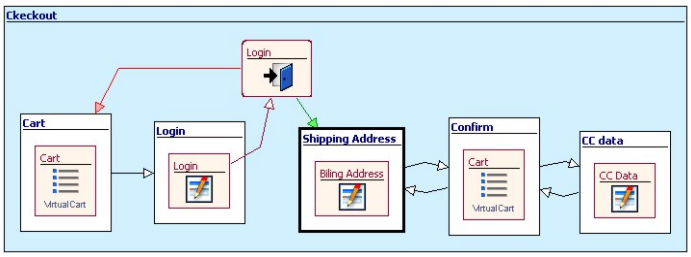



**Fig 3.** Checkout process WebML diagram.

WebRatio is now ready to generate the application. Once we have a running prototype, we can adapt the tests (see section 3.5) and run them to check whether the models (and therefore the application) conform with the requirements or not.

Finally, we adjust the presentation of the application. WebML does not define a presentation model; instead, it is considered as a document transformation from a WebML specification of a page into a specific language page like JSP. In another methodology, mockups and UIDs would be used to specify the presentation model as

well. Since we already had mockups for our current UID, we just slice up the mock-up, and use it as an XHTML template in WebRatio. We can run the tests again to ensure interaction is not corrupted while the template is being modified.

### 3.5 Testing Usability specifications

After building the models, we need to make sure that the implementation generated from them is valid according to usability requirements specification. As previously mentioned, if we try to run the tests as they are written, they will fail because they still reference mockups files and the location of several DOM elements may have changed (in terms of an XPath expressions [26]). In both cases tests should be adapted to work with the current implementation as shown in [19]. In the first case, URLs in the tests should be readdressed to the actual location of the generated prototype. On the second case, the adaptation is easy to perform using a tool like XPather plugin [27] for Mozilla browser. Next we re-run the test and verify whether it succeeds.

### 3.6. Refactoring to improve Usability

Usability requirements can also appear *after* the application has been deployed due to user tests ran at this time. In our example, the user might want to abort the operation during the checkout process. To support this functionality, we can include *Abort operation*, a usability mechanism used to cancel the execution of an action [12]. Therefore, it can be added to each step of the checkout process. As first step for including Abort operation, we have to detect usability requirements. From the usability requirements guidelines of this mechanism, we have extracted the characteristics that the analyst must specify in the conceptual model. The process of requirement capture is done providing a value for each one of the characteristics of Abort operation (Table 2).

**Table 2.** Usability requirements for Abort operation

| **Abort operation** | |
|---|---|
| **Characteristic** | **Value specified by the analyst in the checkout example** |
| Service selection | This mechanism will be applied to the *checkout action* |
| Visual aspect | The abort operation will be triggered by a *Cancel* button in each step of the wizard |

In this case we have a test that validates the checkout process (Section 3.5). As the process involves many steps, we should validate that the abort operation works successfully on every step. Whether the user is in the first step or in the last one, the abort operation must cancel the process and remain all products in the shopping cart.

In order to handle this new usability requirement, we need to adapt existing artifacts to satisfy it. Following the process, we start by adding the new cancel button to the existing mockups on every step of the wizard.

Next, we add tests to validate the checkout process and verify that the buttons exist and behave as expected: canceling the process and navigating back to the shopping cart node. In our case, the checkout process is divided into four steps, so we need to validate that the abort operation works as expected on every one. In order to make the test works, we follow the checkout process to reach the node under test, next click the abort button in order to assert that the location has changed to the shopping cart. Finally, we have to check that the product is still present on the shopping cart. In the next piece of code, we show a short version of the tests to validate the abort operation:

```
public class CheckoutTestCase extends SeleneseTestCase {
  public void testAbortInShippingAddress() {
      this.setupShoppingCartAndStartCheckoutProccess();
      this.clickAbortAndCheckLocation();
  }
  public void testAbortInBillingAddress() {
      this.setupShoppingCartAndStartCheckoutProccess();
      this.fillShippingAddress();
      this.clickAbortAndCheckLocation();
  }
  public void testAbortInProductConfirmation () {
      this.setupShoppingCartAndStartCheckoutProccess();
      this.fillShippingAddress();  this.fillBillingAddress();
      this.clickAbortAndCheckLocation();
  }
  public void testAbortInCreditCard() {
      this.setupShoppingCartAndStartCheckoutProccess();
      this.fillShippingAddress(); this.fillBillingAddress(); this.confirmProducts();
      this.clickAbortAndCheckLocation();
  }
  public void clickAbortAndCheckLocation () {
    selenium.click("abort-checkout");
    selenium.waitForPageToLoad("30000");
    selenium.assertLocation("/cart*");
    assertEquals("The Digital…", selenium.getText("/ul[@id='selected-products']/li[1]/span[1]"));
  }
}
```

Next, the process continues adapting the models in WebRatio. To do so, we add the necessary links and navigation control for the cancel step functionality. Then, we derive the application and run the tests against it. Tests could be also adapted if WebRatio doesn't fit location and layout issues. If so, we should adapt them in the same way we did in section 3.5. If tests still fail, then we need to tweak the models and derivate the application again until all tests pass.

## 4 Related Work

The aim of this paper is to put together the advantages of using agile approaches in Web application development [15] and MDWE approaches in Web application development. Most Web Engineering methods like WebML, UWE, OOHDM, OOWS or OOH do not support agile approaches. In particular, this paper focuses on the agile

approach called Test-Driven Development where tests are developed before the code in order to guide the system development. Some works propose generating these tests automatically, for example the work of Bryc [30], while in other works tests are done manually [14]. Both techniques are valid for our proposal.

We state that usability must be included in the TDD process from the requirements capture step. Several authors, as Juristo [12], have dealt with usability as a requirement. Juristo has defined a set of Functional Usability Features that are related to system architecture. The requirements of these features are captured by means of guidelines. These guidelines include questions that the analyst must ask to end-users in order to adapt the features to users' requirements. Lauesen [16] also includes usability in the requirements capture, discussing six different styles of usability specification and showing how to combine them in a complex real-life case to meet the goals. Finally, it is important to mention the work of Cysneiros [4], who has defined a catalogue to guide the analyst through alternatives for achieving usability. The approach is based on the use of the i* [28] framework, having usability modeled as a special type of goal. The difference between our proposal and the aforementioned works is the context of use. We deal with usability requirements in a TDD process using an MDWE approach, while mentioned authors deal with usability requirements in a traditional software development process.

As for including TDD in a Model-Driven Development (MDD) process, it is important to mention the proposal of Zhang [29]. Zhang has defined a process that involves automatic testing through simulation and using executable models. In other words, this author has defined a process to create tests that must be applied in a simulation of the system. This simulation is obtained by the Conceptual Model which represents the system abstractly. The disadvantage that we have found in Zhang's proposal is that tests are not applied to the final code but a simulation. If the final code differs from the simulation, test results are not useful.

A similar work that proposes testing the system by means of Conceptual Models has been developed by Dinh-Trong [5]. This author has defined a technique for testing design models that are expressed in the Unified Modeling Language (UML) [24]. Test cases are generated using information from class diagrams and sequence diagrams. Our proposal is different to the work of Dinh-Trong. We state that tests must check the generated code because they are closer to the user. Therefore, users can participate in the test definition. However, Dinh-Trong proposes testing the system by means of design models, where users cannot take part for ignorance.

Finally, other authors have proposed testing the system in the code generated from a Conceptual Model, as we propose. The work of Wieczorek [25] is included in that group. This author proposes a black-box testing that uses structural and behavioral models, described in UML, to automatically generate test cases. After automatically generating part of code from the Conceptual Model, developers are starting to create unit tests for the functions that they are going to implement. Changes derived from testing are applied directly to the code. This fact differs from our proposal, where changes are directly applied to the Conceptual Model and the code is automatically generated, making the software development process more efficient.

## 5 Concluding Remarks and Further Work

We have presented a novel approach for introducing usability requirements in a test driven model development approach. Usability characteristics are captured using a set of guidelines described in natural language. In order to fit these kinds of requirements in the TDD cycle, we add tests that drive the development and check that the generated application is valid according to such requirements. The approach maintains the agile style while dealing with usability requirements in an incremental way.

We are currently working on several directions: First, as usability requirements are repeated through many applications and hence can be catalogued [12], we are creating template classes for capturing the functionality that has to be tested on each pattern. The idea is to use or extend these classes to replace all the existing lines on the tests. Using small "testing" classes as first class objects we can compose and improve the time of testing creation and also raising the level of abstraction. Second, we are doing some field experiences with usability requirements on RIA applications [6]. For this matter we are analyzing how to validate those requirements in tests and where they should appear in the TDD cycle. Finally, in order to integrate all these features, we are working on a tool that generates interaction tests from high level UID models and UI mockups in a semi-automatic way..

## References


1. Bass, L., Bonnie, J.: Linking usability to software architecture patterns through general scenarios. The journal of systems and software 66 (2003) 187-197
2. Beck, K.: Test Driven Development: By Example (Addison-Wesley Signature Series), 2002
3. Ceri, S., Fraternali, P., Bongio, A. Web Modeling Language (WebML): A Modeling Language for Designing Web Sites. Computer Networks and ISDN Systems, 33(1-6), 137-157 June (2000).
4. Cysneiros, L.M., Kushniruk, A.: Bringing Usability to the Early Stages of Software Development. International Requirements Engineering Conf. IEEE(2003) 359- 360
5. Dinh-Trong, T.T., Ghosh, S., France, R.B.: A Systematic Approach to Generate Inputs to Test UML Design Models. 17th International Symposium on Software Reliability Engineering (2006) 95-104
6. Duhl, J. Rich Internet Applications. A white paper sponsored by Macromedia and Intel, IDC Report, 2003
7. Fons J., P.V., Albert M., and Pastor O: Development of Web Applications from Web Enhanced Conceptual Schemas. ER 2003, Vol. 2813. LNCS. Springer (2003) 232-245
8. Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.
9. Gómez, J. and Cachero, C. 2003. OO-H Method: extending UML to model web interfaces. In information Modeling For internet Applications, P. van Bommel, Ed. IGI Publishing, Hershey, PA, 144-173.
10. Jacobson, I, Object-Oriented Software Engineering: A Use Case Driven Approach, ACM Press, Addison-Wesley, 1992.

11. Jeffries, R. E., Anderson, A., and Hendrickson, C. 2000 Extreme Programming Installed. Addison-Wesley Longman Publishing Co., Inc.
12. Juristo, N., Moreno, A.M., Sánchez, M.I.: Guidelines for Eliciting Usability Functionalities. IEEE Transactions on Software Engineering, Vol. 33 (2007) 744-758
13. Koch, N., Knapp, A.. Zhang G., Baumeister, H.: UML-Based Web Engineering, An Approach Based On Standards. In Web Engineering, Modelling and Implementing Web Applications, 157-191. Springer (2008).
14. Maximilien, E. M. and Williams, L. 2003. Assessing test-driven development at IBM. In Proceedings of the 25th international Conference on Software Engineering (Portland, Oregon, May 03 - 10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 564-569.
15. McDonald, A., Welland, R.: Agile Web Engineering (AWE) Process: Multidisciplinary Stakeholders and Team Communication. Web Engineering, Springer US 2003, ISBN: 978-3-540-40522-1, 253-312.
16. Lauesen, S.: Usability Requirements in a Tender Process. Computer Human Interaction Conference, 1998, Australia (1998) 114-121
17. Panach, J.I., España, S., Moreno, A., Pastor, Ó. Dealing with Usability in Model Transformation Technologies. ER 2008. Springer LNCS 5231, Barcelona (2008) 498-511
18. Rasmussen, J.: Introducing XP into Greenfield Projects: lessons learned. *IEEE Softw*, 20, 3 (May-June 2003) 21- 28
19. Robles Luna, E.; Grigera, J.; Rossi, G.: Bridging Test and Model Driven Approaches in Web Engineering. Still to be published….
20. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications using OOHDM. In Web Engineering, Modelling and Implementing Web Applications, 109-155. Springer (2008).
21. Seffah, A., Mohamed, T., Habieb-Mammar, H., Abran, A.: Reconciling usability and interactive system architecture using patterns. Journal of Systems and Software 81 (2008) 1845-1852
22. Selenium web application testing system. http://seleniumhq.org/
23. The WebRatio Tool Suite. http://www.Webratio.com.
24. UML: http://www.uml.org/ Last visit: April 2009
25. Wieczorek, S., Stefanescu, A., Fritzsche, M., Schnitter, J.: Enhancing test driven development with model based testing and performance analysis. Testing: Academic and Industrial Conf Practice and Research Techniques, TAIC PART '08 (2008)82-86.
26. XML Path Language (XPath). http://www.w3.org/TR/xpath
27. XPather - XPath Generator and Editor. https://addons.mozilla.org/en-US/firefox/addon/1192
28. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: IEEE (ed.): IEEE Int. Symp. on Requirements Engineering (1997) 226-235
29. Zhang, Y.: Test-driven modeling for model-driven development. IEEE Software 21 (2004) 80-86
30. Bryc, R.: Automatic Generation of High Coverage Usability Tests. Conference on Human Factors in Computing Systems (CHI), Doctoral Consortium. ACM, Portland, USA (2005) 1108-1109