Model-based Regression Testing—Process, Challenges and Approaches[1]

Qurat-ul-ann Farooq
Matthias Riebisch
Steffen Lehnert
Ilmenau University of Technology, Germany

To improve is to change; to be perfect is to change often (Winston Churchill)

ABSTRACT

Evolution is the consequence of the continuous changes, which a software system has to perform due to changes in requirements and various maintenance activities. Regression testing provides a means to assure the wanted properties of the system after the introduction of the changes; however, testing requires high effort. Model-based regression testing (MBRT) has the potential to perform test tasks with a much better efficiency. MBRT uses analysis and design models for identifying changes and their corresponding test cases to retest the system after modifications. MBRT promises reduction in cost and labour by selecting a subset of the test cases corresponding to the modifications. However, the identification of modifications in a system and the selection of corresponding test cases is challenging due to interdependencies among models.
This chapter aims to provide a detailed insight into MBRT, how it is related to the general software lifecycle and what are the challenges involved. We evaluate the state of the art in MBRT with a detailed analysis of the strengths and weaknesses of the existing approaches. For the analysis we develop a set of comprehensive analysis criteria based on the identified challenges. Furthermore, we demonstrate the applicability of MBRT by presenting our state-based MBRT approach as an example. The chapter targets researchers and practitioners who want to achieve a detailed comprehension of the field and want to know the strengths and weaknesses of the existing approaches in MBRT. This chapter also identifies the areas within MBRT which require further attention by the researchers.

I. INTRODUCTION

Evolution is inherent to the software systems. Due to the growing size and complexity of modern systems, the evolving nature of a system can cause adverse side effects and even system failures. Besides many other measures to prevent these unintended effects of evolution, it is essential to test a system after modifications; often referred to as *Regression Testing*. Regression testing is performed during the software maintenance phase and during various maintenance activities including *corrective*, *perfective* and *adaptive* maintenance (Wu & Offutt, 2003).

[1] The research presented in this chapter was partly funded by the Federal State Thuringia and the European Regional Development Fund ERDF through the Thüringer Aufbaubank with project no. 2007 FE 9041.

When a software system is modified, repeating the entire testing activity is a very costly task. A large system may have a huge number of test cases and test-execution requirements. Executing all these test cases is generally not a economically feasible option. Hence, it is necessary for regression testing to select a subset of the test cases corresponding to modifications. This is known as the selective strategy for regression testing and is a more feasible solution in terms of cost and time (Binder, 1999).

Another important issue during regression testing is scalability. Conventional code-based regression testing approaches fail to deal with the huge size of modern software systems. Model-based regression testing is a potential solution to this problem, because it offers several advantages compared to the conventional code-based regression testing approaches. This includes *better scalability*, *better complexity management* and *better comprehension* of the system, the relevant test suites and test cases. In model-based testing, the testing activity can be started in early phases of software development allowing *early regression planning and estimation* (Briand, Labiche, & He, 2009). This results in *effort reduction* in terms of time and labour. Furthermore, traceability *maintenance* between test cases and models is relatively easier to accomplish as compared to the code-based approaches (Briand, Labiche, & He, 2009). Due to the use of models as primary artefact in the MBRT, static and dynamic *interactions are more visible* in design models and no static and dynamic analysis is required to determine the dynamic bindings as in code based approaches. Finally, *portability* and *platform independence* is a major benefit for evolving systems to adopt the rapid changes in technology and operational environment.

Besides all these benefits, there are some limitations of model-based regression testing as well. One of the major limitations is the potential impact of incomplete and outdated design models on the creation of effective regression test suites and plans. Moreover, since the test cases are generated from the design models, they are more abstract than test cases generated from code. This abstraction, sometimes make the test execution more difficult as the test cases should be adapted according to the implementation environment. However, considering the benefits of model-based regression testing, these limitations can be somehow compromised.

In this chapter, we try to not only give a broad overview of the area but also discuss all the main steps and key challenges involved in model-based regression testing. We discuss the role and place of MBRT in the classical software development lifecycle and identify the major steps involved in the MBRT phase. We also identify and discuss the challenges associated with model-driven regression testing approaches which are relatively novice approaches and are influenced by the concepts of model-driven architecture (MDA, 2011). We present our state-based regression testing approach with a demonstrating example to apply the regression testing steps identified previously and give a detailed insight of how practical regression testing can be performed to the reader. As the major contribution of the chapter, we provide a comparative analysis of the existing MBRT approaches. For this analysis we develop comprehensive evaluation criteria and evaluate the approaches based on the criteria. Our analysis not only shows the strengths and weaknesses of the MBRT approaches but also highlights the areas which still require attention of the researchers and developers. By reading this chapter, researchers and practitioners can get a thorough picture of the area and state of the approaches available in the

area. They will also get an insight into how practical MBRT can be performed and what are the major challenges in the field. The rest of the chapter is organized as follows.

Section II provides an overview of model-based regression testing. It discusses the MBRT in the context of the traditional software development lifecycle and also elaborates the major steps involved in the regression testing phase.

Section III presents a state-based regression testing approach developed by the authors as an example, together with a demonstrating case study to explain the approach. The approach and the discussed example provide an insight how the regression testing steps can be performed in practical scenarios.

Section IV discusses the challenges involved in MBRT in detail. These challenges include change identification, the notion of change propagation, the difficulties associated with baseline test suite generation, the risk of invalid test cases after regression test selection and the challenge of test automation. The challenges guide the analysis and classification of the state of the art presented in the section VI.

Section V extends the discussion of challenges of the previous section by those of model-*driven* regression testing approaches.

Section VI contains a comprehensive analysis and classification of existing MBRT approaches. It discusses the research questions identified, the comparison criteria and later the detailed analysis based on those criteria.

Section VII finally concludes the chapter and sums up its contents and findings.

II. MODEL-BASED REGRESSION TESTING: THE BIG PICTURE

In the traditional software development life cycle (SDLC), MBRT is the part of the testing activity in the maintenance phase. If we consider the simple waterfall development model, the regression testing will be performed in the maintenance phase when a change request is triggered. The following figure depicts the place of regression testing in the classical waterfall style SDLC. However, this figure is just for understanding the concept, in reality MBRT is applicable to all major SDLC,s for example RUP[i], SCRUM[ii] etc.

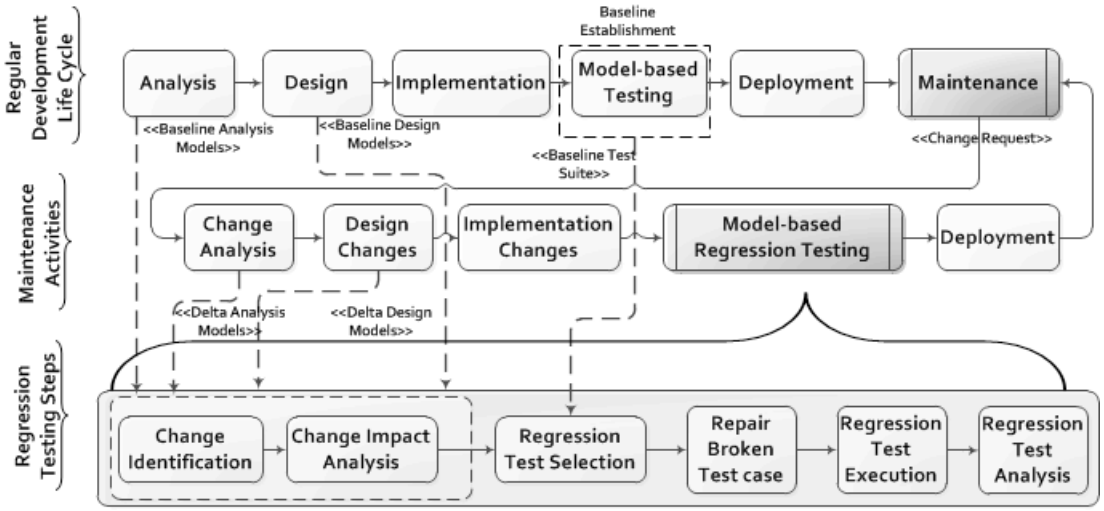

Figure 1: MBRT the big picture

According to Figure 1, a software system is constructed using the steps of any regular software development life cycle. When a change request is triggered probably due to a changed requirement, the maintenance activities are performed to entertain it. Once this new requirement is implemented, the system should be tested to detect the faults introduced by the changes. Model-based regression testing is used in this phase to test the changed software system.
As depicted in Figure 1, model-based regression testing constitutes of several steps shown in the last row. We discuss these steps in detail in the following sub sections.

A. Steps Involved in Model-based Regression Testing Phase

As depicted in Figure 1, we identified 6 major steps, which constitute the MBRT phase. Before discussing those steps in detail, we first discuss a pre-requisite of MBRT in the next section i.e. establishment of baseline test suite. The sections afterward provide a discussion of the steps of the MBRT phase.

1. Baseline Test Suite Establishment

Before performing the actual regression testing, we need to have an existing test suite of the stable version of the system (the so-called baseline system). This test suite is often referred to as a baseline test suite. The test selection during regression testing is then performed by using this test suite. The establishment of a baseline test suite is a necessary activity because if there is no formal baseline test suite available, no regression testing can be performed.

Baseline test suites are often constructed using model-based testing approaches as shown in Figure 1. However, an important aspect while establishing the baseline test suite is preserving the relationships between the constituents of the test suites and the systems models. Otherwise, it will be hard to identify the affected test cases corresponding to affected elements of the models.

This concept is a core concept in the field of model-based regression testing and is discussed in detail in section IV.B. In the next sections, we discuss the regression testing steps depicted in the lower part of Figure 1.

2. Change Identification

Change identification is the first step of regression testing performed after establishing the baseline test suite. This step aims at the identification of the delta–the changes introduced in the new system. In MBRT this delta identification is often performed by comparing various design and architecture models. A more detailed discussion on change identification is available in section IV.A.

3. Change Impact Analysis

After obtaining the delta, the next step in MBRT is *Change Impact Analysis*. Several relationships and dependencies exist between different elements of the system. An impact analysis is necessary to identify the elements affected due to these relationships and dependencies. In context of model-based development several models of a system represent different views of the system and hence, are related to each other to give a complete picture of the system. The aim of change impact analysis in MBRT is to determine the impact of change in one model on other models of the system. It helps to identify the parts of the models/system which are indirectly affected by the changes. This topic is discussed in detail in section IV.B

4. Regression Test Selection

Regression test selection is performed after identifying the changed and impacted elements in the system. In this step, the changes and their impact (already determined in the previous steps) is used to select a subset of the test suite for regression testing. Relationships between elements in the model and test cases are established for performing test selection. As discussed earlier, these relationships are either established at the time of baseline test suite establishment or they need to be discovered later by applying heuristics to discover such relations. Test cases are classified against the added, deleted and changed parts of the system. A very famous test suite classification often adopted by several regression testing techniques in the literature by Leung et al. divides the regression test suite into obsolete, reusable and re-testable test cases (Leung & White, 1989). Test cases may also be prioritized based on cost and risk factors (Chen & Probert, 2003).

5. Repair Broken Test cases

A lot of test cases become inapplicable due to the changes introduced to the system. After performing the test selection, it is necessary to identify such test and repair them for further use.

6. Regression Test Execution

Once all the test cases have been selected and broken test cases are repaired, the next step is execution of these test cases. This step does not require any special tools and techniques, as existing test execution methodologies, environments and engines used for testing the base line can be adapted during the regression test execution as well.

7. Regression Test Analysis

The last step is to analyze the test results and evaluate the test verdicts to determine the regression defects. Existing test analysis approaches for baseline test analysis can also be used in this step. If some defects are uncovered during the test analysis, some rework is often required to correct the system. In the proceeding section we discuss an example state-based regression testing approach to demonstrate the applicability of the steps discussed above.

III. A PRACTICAL STATE-BASED MBRT APPROACH: A DEMONSTRATING EXAMPLE

In this section, we present a state-based approach for model-based regression testing developed by the authors, together with an illustrating example. This approach provides a practical demonstration of all the MBRT steps discussed in Section II.A. It is also included later in our analysis of the MBRT approaches.

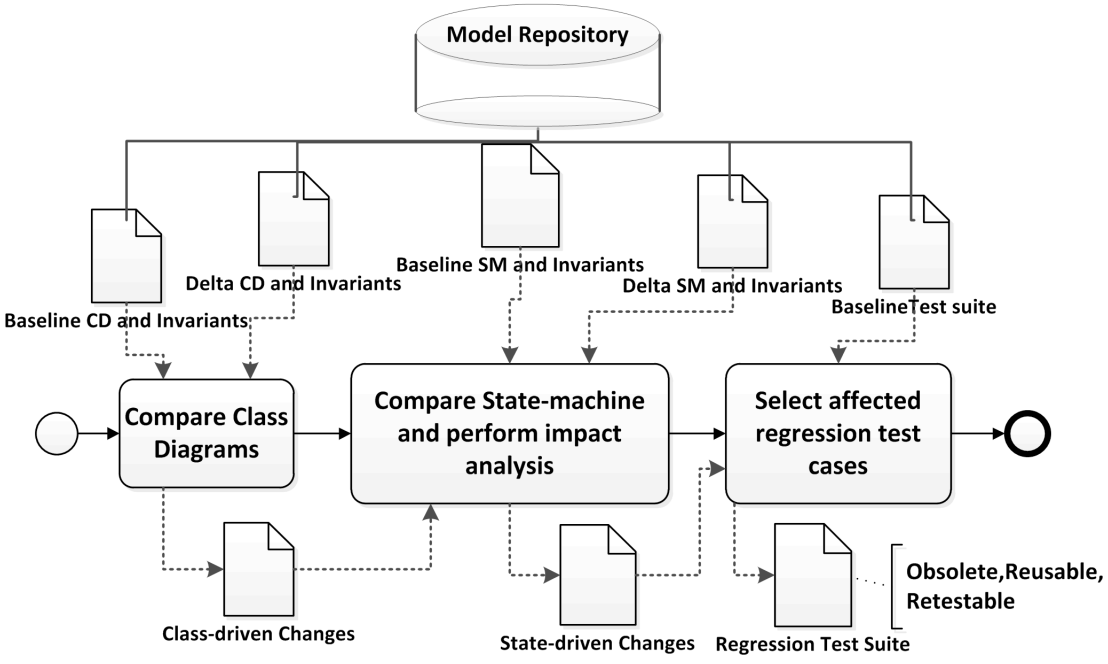

Figure 2: State-based Regression Testing Process

Our approach uses UML class diagrams and state machines as input. The dependencies between both types of artefacts are discovered and change impact analysis is performed based on these dependencies.
Figure 2 presents the overview of the approach. According to
Figure 2, first of all the baseline version of both state machine and class diagram are compared for change identification and impact analysis. These artefacts are stored in a model repository for the version control. The set of changes obtained after the comparison is used to select the regression test cases from the baseline test suite. Before we discuss these activities in detail in following sections, we explain how the baseline test suite was established in the next section.

A. Baseline Test Suite Establishment in the State-based MBRT Approach

As mentioned earlier baseline test suite construction is a prerequisite for regression testing. We constructed our state-based base line test suite using the transition tree methodology (Binder, 1999). Figure 3, depicts the partial transition tree of the state machine corresponding to the Student class presented in Figure 5. The dashed lines in the figure represent an ongoing path which is not shown in due to the huge size of the tree.

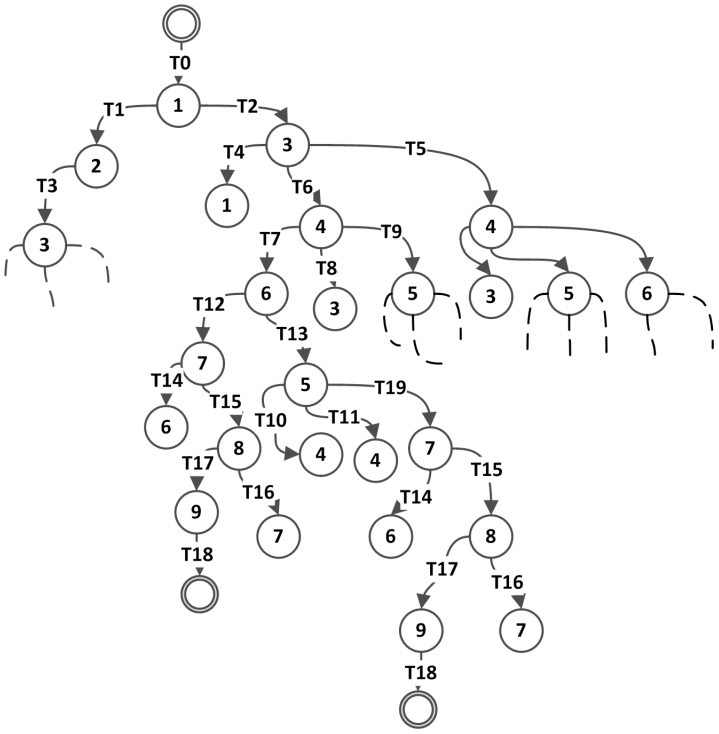

Figure 3: Partial Transition Tree for Baseline Student state machine

The concrete test representation is in a XML format, where each block marked as Test case in the XML document represents a distinct path in the transition tree. Listing 1 depicts an excerpt of the test suite contacting two cases.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:uml="http://schema.omg.org/spec/UML/2.0"
xmlns:xmi="http://schema.omg.org/spec/XMI/2.1">
<Test case name="TC1" TestContext=”StudentClass”>
<TestComponents>
<TestComponent name=”RegistrarTestComponent”/>
</TestComponents>
                    <TestTransition     name= "T0" source="Initial" target="RegisteredStudent" trigger=""/>
                    <TestTransition     name= "T2" source="RegisteredStudent" target="BeingEnrolled"
trigger="registrar.enrollInCourse"/>
                    <TestTransition     name= "T4" source="BeingEnrolled" target="RegisteredStudent"
trigger="self.status==unsuccessful"/>
</Test case>
<Test case name="TC2">
<TestComponents>
<TestComponent name=”RegistrationTestComponent”/>
</TestComponents>
                    <TestTransition     name= "T0" source="Initial" target="RegisteredStudent" trigger=""/>
                    <TestTransition     name= "T2" source="RegisteredStudent" target="BeingEnrolled"
trigger="registrar.enrollInCourse"/>
                    <TestTransition     name= "T6" source="BeingEnrolled" target="Enrolled"
trigger="self.status==unsuccessful"/>
                    <TestTransition     name= "T7" source="Enrolled" target="Enrolled | Studying"
trigger="registration.currentSemester.status==Study"/>
                    <TestTransition     name= "T12" source="Enrolled | Studying" target="Enrolled |
GivingExams" trigger="registration.currentSemester.status==midTerm||finalExam"/>
                    <TestTransition     name= "T14" source="Enrolled | GivingExams" target="Enrolled |
Studying" trigger="registration.currentSemester.status==Study"/>
</Test case>
</xmi:XMI>
```

Listing 1: The Concrete Test Representation in XML Format

B. Traceability between System Models and Test cases

We use the concept of implicit traceability to discover the relationship between class diagrams and state machines and then the test cases. In implicit traceability, the traceability information is not already available and made explicit. To discover the relationship between class diagrams and state-machines, we apply heuristics based on similarity of the names and the ID’s of model elements.

Discovering the relationship between state machines and test cases is simpler in our approach. During the test suite generation every test case contains the id of the corresponding transition in the state machine. These ID’s are used later to trace the test cases corresponding to the affected transitions.

C. Performing Change Identification and Impact analysis on the state-based MBRT Approach

The first step of our MBRT approach is change identification and impact analysis. According to Figure 2, at first the changes in the class diagram are obtained by comparing the baseline and the delta versions of class diagrams along with class invariants and operation contracts. A change set

is obtained after this comparison and is referred to as "Class-driven Changes". Change computation is performed by comparing the properties of all the elements of the class model such as classes, operations and attributes, after parsing both class diagrams.

After computing the class-driven changes the next activity is state machine comparison. Changes in both versions are detected and class-driven changes are used to obtain the affected elements of the state machine. For example, a state transition will be marked as affected if it uses any changed attribute or operation of the corresponding class in its guards, events or actions. The set of these changes is referred to as state-driven changes.

To elaborate the methodology, let us consider Student Enrolment System as an example. Figure 4 represents an excerpt of the class diagram of the Student Enrolment System.

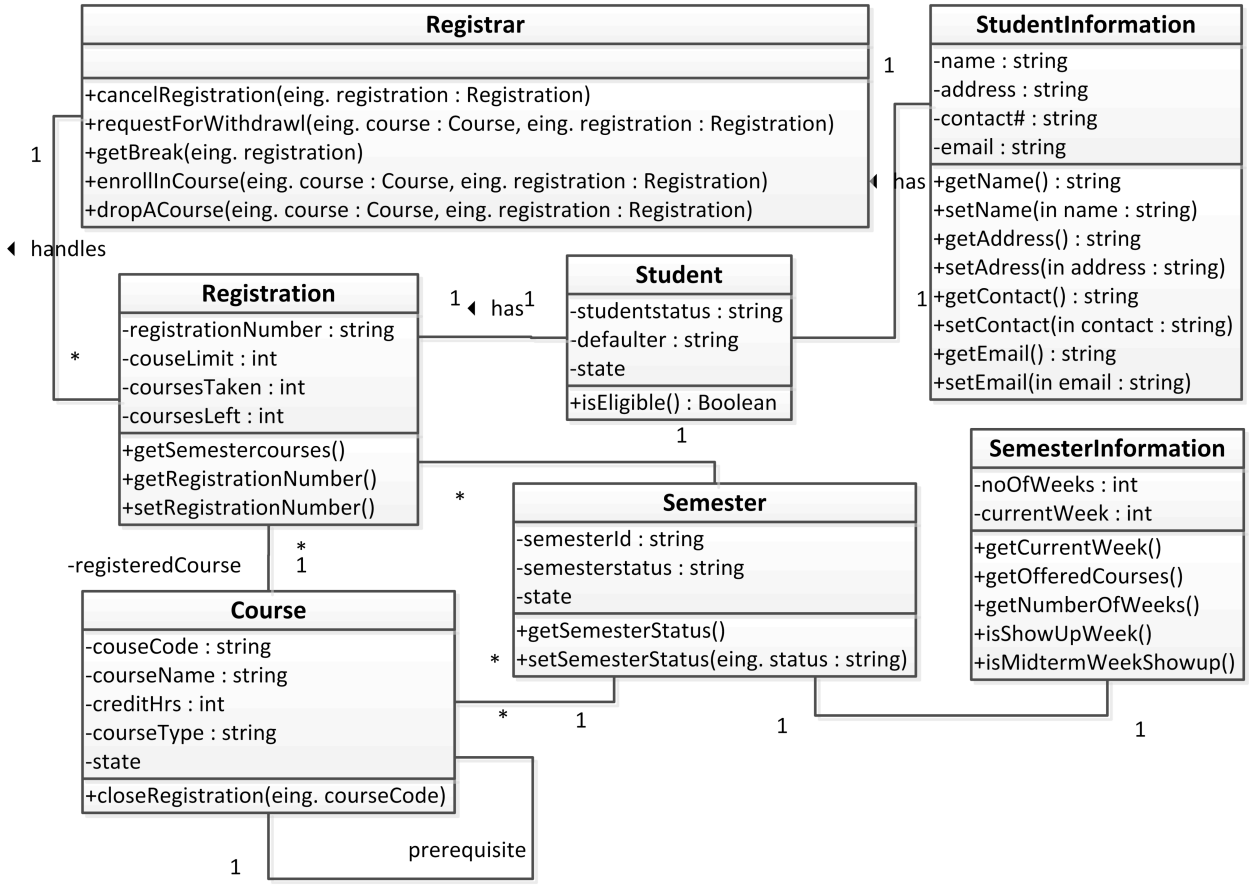

Figure 4: Baseline version class diagram of the example student enrolment system

In the class diagram shown in Figure 4, we have state machine corresponding to two state-full classes; the Student class and the Course class. Figure 5 depicts the state machine of the baseline version of the student class.

Following corrective changes are introduced in the new version of the system.

1. The defaulter attribute of student class is modified. Its type is changed from *String* to *Boolean*.
2. State 2 and transition T1 and T3 are no more in the state machine of student class.

During the class diagram comparison process, the defaulter attribute is identified as modified and inserted in the set of class-driven changes. In the state machine comparison process, the state 2 and transition T1 and T3 are marked as deleted. However, an additional transition is also marked as modified. The transition T5 is using the defaulter attribute of the student class in its guard condition "[self.defaulter="false"]", hence T5 is also marked as a modified transition
Listing 2 contains two sample change impact rules for the modified transition. It is important to note that such change models and change impact rules are defined for every model element in the state machine meta-model.

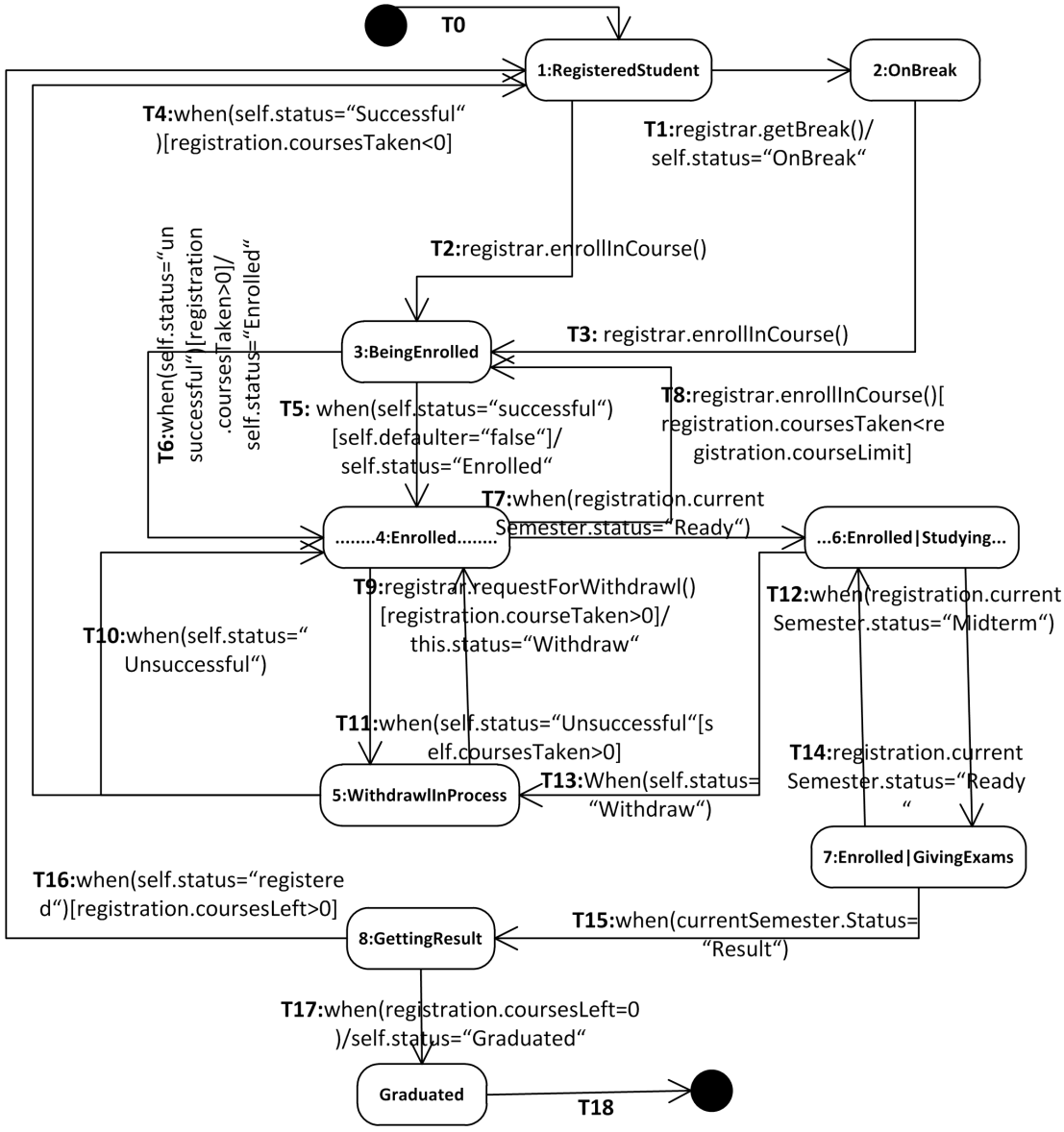

Figure 5: Base line Version state machine of the of the Student Class

A. Regression Test Selection

Finally, the set of affected test cases from the baseline test suite are selected by tracing the state-driven changes to the corresponding test cases. As discussed earlier, the test cases contain the information about the ID's of the transitions they correspond. The ID,s of the affected transitions are matched with the test cases to identify the affected test cases. Our test suite is classified into three types of test cases; obsolete, reusable and re-testable (Leung & White, 1989).

1. A transition is modified if the event associated with the transition is modified.
    a. A call event is modified if its corresponding operation/Operation Contract defined in the class diagram is modified
    b. A signal event is modified if its corresponding operation /Operation Contract defined in the class diagram is modified
    c. A Change Event or a Time Event is modified if it uses a variable defined as class attribute in the class diagram and that variable is modified.
2. A transition is modified if its guard conditions are modified
    a. A guard condition is modified if it uses a variable or operation defined in the corresponding class and that variable or operation is modified.

Listing 2: Example Change Impact Rules for ModifiedTransition

This classification is adopted by several regression testing techniques in the literature. Obsolete test cases are no more valid for the delta version. They usually correspond to elements in the system that are deleted and are not accessible in the delta version. Re-testable test cases need to be executed for regression testing as they correspond to modified parts of the system.

Table 1 summarizes the results of the case study. According to the table, the baseline test suite of the Course and Student class consists of 723 and 58 test cases respectively.

Table 1: Results of The Student Enrolment Case Study

Test Classes	Total base-line test cases	Reusable	Re-testable	Obsolete
Course	723	276	447	0
Student	58	14	15	29

After performing the test suite classification for regression test selection, the total number of re-testable test cases that are required to execute during regression testing for the Course and Student class are 447 and 15 respectively. 29 test cases of the student class are also marked as obsolete and cannot be executed to test the delta version of the system.

IV. CHALLENGES IN MODEL-BASED REGRESSION TESTING

In this section gives a general introduction to the challenges in research and industrial application of model-based regression testing. These challenges guide the analysis and classification in section VI.

A. Change Identification

Change identification is a crucial part of regression testing. As discussed earlier, change identification in MBRT is the process of calculating the change given one of more models of baseline and delta version of the system. Model comparison is a key concept while dealing with change identification in MBRT. Unfortunately, most of the existing approaches in MBRT do not place much focus on this aspect (see section VI.D.3).

However, in a few approaches [(Briand, Labiche, & Soccar, 2002), (Farooq, Z., Malik, & Nadeem, 2007)] change identification is explicitly covered. These approaches focus on elementary change types; *addition* of a model element, *deletion* of a model element and *modification* (*changed/added/deleted* property) of a model element. However, there can be more complex and fine grained changes. The so-called modular operators discussed by Baldwin and Clarke (Baldwin & Clarke, 1999) can be used as a foundation for understanding complex change types. These modular operators are *splitting*, *substituting*, *augmenting*, *excluding*, *inverting* and *porting*. Mäder et al. (Mäder, Gotel and Phillipow, 2009) also presented an interesting taxonomy of complex change types for traceability maintenance. These change types are *add*, *delete*, *move*, *merge*, *split*, and *replace*.

In Figure 6, we present a taxonomy of elementary and complex model changes for comparison of models for change identification during regression testing. The figure contains the change types from Baldwin and Clark, and Mäder et al. and some other additional change types relevant to model comparison. All the change types/operators are presented with a demonstrating example of a structural model (class diagram) and a behavioural model (activity or process diagram). In Figure 6, left hand side of the containers separated with a dashed line show the original models and right hand side shows the modified models. The name of the container depicts the name of the change type/operator applied. We discuss these change types/operators in the following.

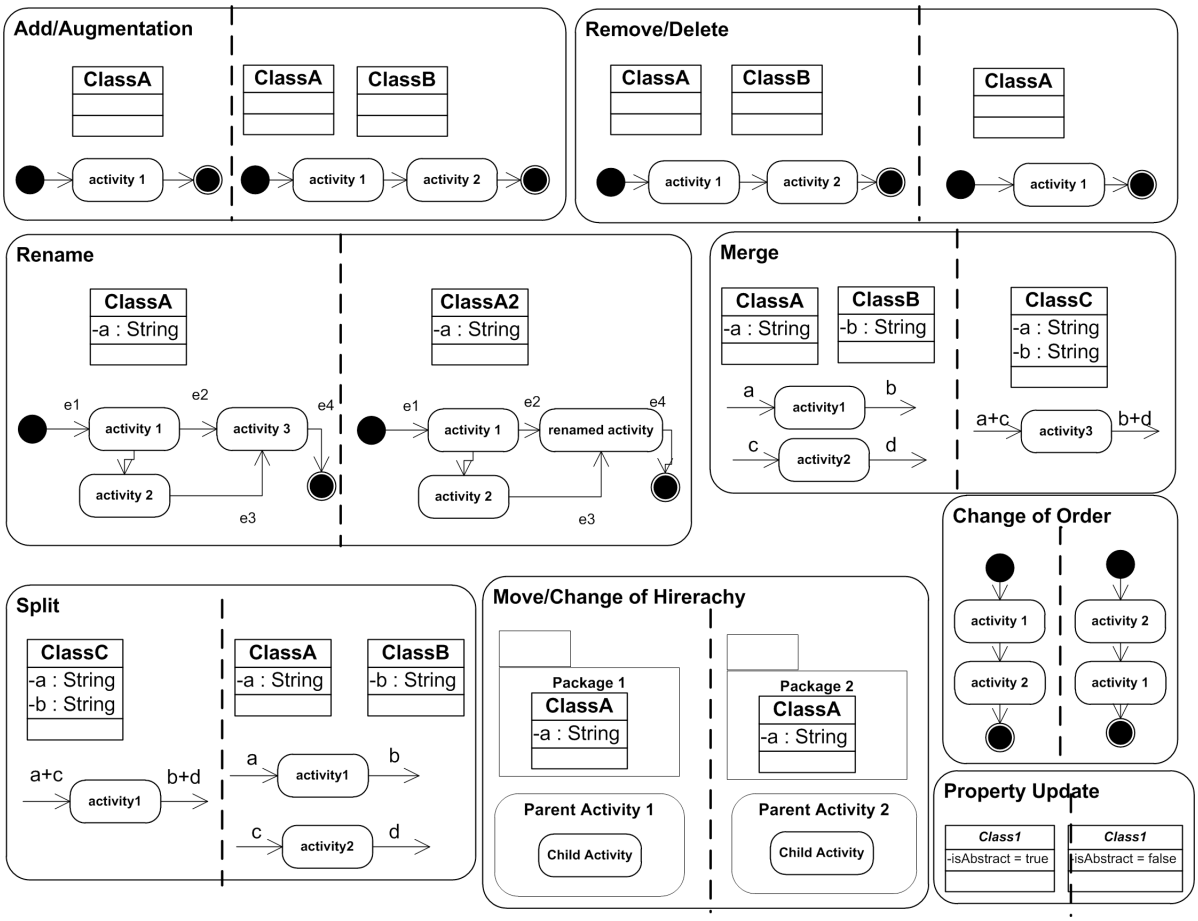

Figure 6: A Taxonomy of Changes for Change Identification between models

Add/Augmentation, Delete/Remove

The first two change types in Figure 6 are *add* and *delete* or according to Baldwin *augmentation* and *remove*. This change type/or operator adds a new model element or removes an existing model element from a model. In the corresponding example in Figure 6 a new class and activity is added in the class diagram and activity/process diagram.

Rename

Rename is a change type/operator which changes the name of a model element in a model. *Rename* is quite similar to the property *update* change type; however, we separated it because name is often a unique and most significant property of a model element and the effect of this change type will normally different in regression testing as compared to the update property change type.

Split and Merge

The change-types *split* splits a model element into n number of model elements; whereas, *merge* merges n number of model elements into 1 respectively. In Figure 6, two classes A and B are merged into 1 class C. It is to notice that after merge properties of both classes are part of the merged class.

Similarly in case of an activity/process diagram example in Figure 6, input and output of the merged process are a combination of both processes it merged. Normally, changes like *merge* and *split* are hard to detect and require very strong heuristics and change detection rules.

Move/Change of Hierarchy
We combined the *move* and *change of hierarchy* change types/operators because, if we move any model element from one place to another, it will be placed from one container to another, which is a form of change in hierarchy as well. However, in some cases change of hierarchy can also be separated from move. For example, in a class diagram, if a new parent class is added or an existing parent class of a class is deleted then it will also cause change of hierarchy without application of move change type/operator.

Change of Order
Change of order occurs in the behavioural models. In the corresponding example in Figure 6, one the order of activities in the activity/process diagram is changed. Such changes are also hard to detect and also require strong heuristics and change detection rules.

Property Update

Property update is a change in which any attribute/property of a model element is changed. This is also a form of elementary change. Figure 6 contains an example of a *property update*, where a property value of the attribute isAbstract of a class is changed from true to false.

We think that the above mentioned change types are very interesting for change identification in MBRT and the investigation of the impact of such change type during regression test selection is a very interesting research question.

In the following sections we will discuss the types of model comparison approaches and the available tools and technologies for model comparison

1. Type of Model Comparison Approaches

There are two basic types of model comparison techniques. We discuss them in the following subsections.

a) Offline/State-based Change Detection

In offline/state-based change detection, two states of the models are compared. The state before change (baseline models) and the state after change (delta model). Models are often compared to detect the added and deleted elements and changed properties of these elements. All the regression testing techniques discussed in the literature so far are using offline change detection.

b) Online/ Event-based Change Detection

In contrast to the offline change detection, each change operation performed by a modeller is recorded in online/event-based change detection. This event capturing mechanism is built-in in the modelling tool and produces chains of recorded events. These event chains are then

processed to extract the complex change type based on modeller's intentions by applying heuristics. Online change detection is used in many fields such as traceability maintenance and conflict resolution [(Mäder, Gotel and Phillipow, 2008); (Gerth et al., 2010)]. However, none of the regression testing techniques discussed in the literature so far uses online change detection. It is an interesting research question how these change operation chains can be processed to compute complex changes and then how they can affect the regression testing. At present, using online change detection for MBRT to answer these questions is a work in progress and the authors of this chapter are working on it.

2. Recent Tools and Technologies for Model Comparison

In this section, we discuss some recent tools and technologies developed in the field of model comparison.

EMF Compare (Generic Model Comparison Tool)
EMFCompare is a generic tool for comparing EMF based models (EMFCompare, 2011); hence, the tool is able to compare any model expressed in EMF. It focuses on the elementary change types add delete and update/change during model comparison. A very interesting feature of EMFCompare is its ability of change visualisation in form of a model.

ECL (Epsilon Comparison Language)
Another interesting concept in model comparison introduced recently is of ECL (ECL, 2011). ECL is a rule-based language to specify the comparison logic; hence, complex change types can be specified. However; more case studies are required to evaluate the comparison strengths and weaknesses and scalability of ECL.

Model Comparison using Model Transformations

In such comparison approaches, comparison logic is specified using model transformation languages. The examples available at the website of ATL (Atlas Transformation Language) for model comparison and model merging are examples of such comparison (ATL, 2011).

The approaches discussed in this section can be used in the change identification phase of MBRT because most of the MBRT approaches do not focus on change identification, as mentioned earlier. It is an interesting research question how regression testing techniques can use these new approaches for change identification, and consider the changes mentioned above.

B. Change Propagation

After the identification of changes, their consequences on tests have to be determined. A change in one artefact affects other related artefacts, for example test cases. This phenomenon is often referred to as *Change Propagation*. In model-based development, several artefacts are covered by different views, which represent different aspects of the system. These views are inter-related as they represent the whole system. It is necessary to consider the relationships and dependencies

between the artefacts and model elements, to analyze the change propagation phenomenon for effective regression testing. Two concepts are important when dealing with change propagation; "*Traceability*" for the establishment and maintenance of the relationships, and "*Change Impact*" for the determination of the affected artefacts. In the following sub sections, we will discuss both concepts in detail.

1. Traceability

According to a definition by Gotel and Finkelstein (Gotel & Finkelstein, 1995), traceability refers to the ability to describe and follow those aspects that are of interest. Traceability deals with the relationships among entities of interest. A traceability link expresses a dependency between two or more entities, which has been passed during a development activity. By definition, a dependency constitutes a relation between two artefacts of which the one has to be adapted if the related artefact changes. For MBRT, relations between artefacts from all development phases such as requirements, design, implementation, deployment, and test are of relevant.

Different artefacts relate with each other in a different context. This context determines the type of the relationship, the type of the artefacts, which can be related, and rules for completeness and consistency. The context, however, is determined by the problem to be solved by the modelling, and thus the goal of the modelling. There is no need to analyse other aspects such as the actual development activity, the design methodology applied, and the problem domain are related to the context in a minor degree, because the goal of the modelling is influenced by them. For example for a modelling of time behaviour, artefacts with a regard to time such as events, tasks and semaphores are relevant. The relevant types of relationships between them comprise all relationships with influence on the time behaviour such as *after*, *before*, *waits-for* and *similar*.

If the analysis of change propagation constitutes the goal of the modelling, all those artefacts are relevant that are influenced by a change, for example use cases and conceptual models as part of requirements models; systems, components, and interfaces as part of the structural view of architectural and design models; the various elements from behavioural models, and the elements of implementation and deployment models. The set of relevant types of relationships comprise all dependencies, for example the relationships *use*, *implements*, *part-of*, *is-a*, *instance-of*, and many more.

The relationship type of dependencies and traceability links is important for its evaluation and utilization. Using the relationship type, rules for consistency checks can be established, and methods for impact analysis can be developed. Unfortunately, there is no standard classification of types of dependencies. Moreover, Antequi et al. argue that it is not possible to foresee all relationship types for dependencies (Anquetil, et al., 2008). However, it is helpful for impact analysis to classify relationship types according to their nature in order to establish rules for impact evaluation for theses classes. Since the goal of modelling determines the relevant relationships, a classification should be made according to the purpose they serve. For testing, the issue analysed by a test determines the relevant dependency types. For example, if functional or structural properties are validated by a test, then component relations such as *part-of*, *kind-of*,

and *instance-of* have to be evaluated to determine the need for a retest of a component after a change of a related component.

In literature, different characteristics for relationships are mentioned. Bachmann outlines basic dependency characteristics (Bachmann, Bass, & Klein, 2002):
Symmetricity explains the existence of names for both directions (e.g. *verify – isTestedBy*).
Semantic dependencies cannot be broken with intermediaries, only weakened by abstractions.
Dependencies are independent of specific changes to a model – dependencies remain despite modifications of modules. Pornpit et al. 2008 present a categorization of relationship types in an ontology, which we can transfer to the determination of impact relevant for tests (Wongthongtham, Chang, Dillon, & Sommerville, 2009): *generalisation*, *association*, *include* relationship, and *extend* relationship.

Furthermore, there is a category of dependencies between problem description and solution, or even between model elements towards a solution. Dependencies of the types *implements* and *realizes* fall into this category. Since, executable software is the subject, dependencies such as *uses*, *defined-by*, and *asserts* can help to evaluate the fact that on artefact has impact on another one. For tests regarding quality issues however, other types of dependencies are relevant according to the type of quality issue to be evaluated. For tests regarding security, safety, and dependability, all dependencies of the types *caused-by*, *Agent-Actor*, *harms*, and *failure-prevented-by* have to be analyzed. Summarizing we have to state that the relevance of a type of dependencies is related to the aspect one wants to test.

2. Change Impact Analysis

Determining the effect of a change to a software system is commonly referred to as Impact Analysis (Bohner & Arnold, 1996). Caused by dependencies between different software artefacts (e.g. classes), the effect of a change is able to spread across the entire software system, causing new changes and resulting in unwanted side effects.

Many different techniques to uncover such ripple-effects have been proposed in literature within the last years of research. The overall goal is to assist developers who are responsible for planning and implementing changes, allowing them to evaluate the effects of proposed changes before actually performing the change.

Impact Analysis can as well be used to [(Kabaili, Keller, & Lustman, 2001); (Orso, Apiwattanapong, & Harrold, 2003)] identify those test cases which must be executed after implementing a change and therefore assist regression testing. As MBRT is concerned with abstract system representations such as a systems architecture, Impact Analysis techniques developed for abstract models, such as (Briand L. , Labiche, O'Sullivan, & Sowka, 2006), can be used to identify possible candidates for retesting.

Apart from dependencies between different artefacts, change couplings offer an additional source for conducting Impact Analysis. Change couplings can be inferred by observing historical

change data, i.e. examining version control systems for patterns or clusters of co-changing artefacts (Xing & Stroulia, 2004), as a change to one artefact of a cluster is very likely to cause changes to the entire cluster of artefacts.

C. **Baseline Test Suite Generation**

As mentioned earlier, it is a prerequisite for regression testing approaches to have an existing baseline test suite used to test the stable version of the system. However, baseline test suite generation is very crucial, since model-based test generation approaches are still not very mature. Most of the test suite generation activities are often manual and only a few approaches for automated test generation in the domain of model-based testing are available.
Moreover, most of the approaches use ad-hoc specification languages for test specification. There are a very few approaches using test specific languages and even these approaches are not mature enough to be applied in different domains and in larger development contexts (Baker et al., 2008).
Moreover, maintenance of traceability and preserving the relationship between models and test cases is also often overlooked during baseline test suite generation. This makes the regression testing activity more difficult.

D. **Validity of Test Cases**

As mentioned earlier, due to introduction of changes, many test cases often become invalid and they therefore, must be identified and repaired prior to regression test execution. How these test cases can be automatically repaired is a very interesting research question which is completely neglected by existing MBRT approaches.

E. **Test Automation**

Like other model-based testing approaches, test automation is also a big challenge for model-based regression testing approaches. The standard conformance of these tools is another major issue, caused by rapidly changing modelling standards. Moreover, integration of model-based regression testing tools with other tools, especially baseline test generation tools and test execution environments is also a necessity which is often overlooked. Since MBRT relies heavily on models precise definitions of models (meta-models) and their implementation is required for tool implementation. Such meta-models are often not available for many domain specific languages. However, they are increasingly made available for example the UML and BPMN meta-model implementation for eclipse platform which is a positive sign for MBRT tool developers [(UML2Eclipse, 2011); (BPMNEclipse, 2011)].

In the next section, we discuss a practical approach for state-based MBRT. We are demonstrating the approach with the help of an example for the sake of brevity. The approach is discussed in context of the MBRT steps and challenges discussed above.

V. MODEL-DRIVEN REGRESSION TESTING – CHALLENGES AND EMERGING APPROACHES

Due to the increasing adaptation of model-driven development in the industry, researchers are investigating the possibilities of introducing MDA practices during regression testing as a next step after MBRT. The core of MDA lies in raising the level of abstraction by introducing models in all the development stages, compared to model-based techniques. Model transformations are the key concept in MDA which allows transformations of models between different or same abstractions. Recently some researchers proposed ideas in the dimension of Model-driven regression testing (referred in this chapter as MDRT).

A. Challenges in Model-driven Regression Testing

1. Maturity of Model-driven testing (MDT) approaches

For the application of regression testing approaches, well defined model-driven test generation approaches are required. Due to the fact that MDT itself is a quite new research area, the number of sophisticated MDT approaches is very limited. MDRT approaches depend on the test-suites generated by MDT approaches; hence, it is difficult to perform the regression testing without existing test generation methodologies.

2. Maturity of Transformation languages

For the application of MDT and MDRT approaches, transformation languages area core requirement. However, a lot of available transformation languages are not mature enough. They lack sophisticated development environments, sometimes they do not support the required modelling languages and sometimes they lack the important development facilities such as debugging etc.

3. Maturity of Test modelling languages

MDRT should use test modelling languages to conform to the MDA ideology. However, the support for test modelling is also very limited. Very few test modelling languages are available for example AGEDIS (AGEDIS, 2002), Tela (Pickin, 2001) and U2TP (U2TP1.0, 2005). However, AGEDIS and Tela are abandoned project and no more support is available for them. Although U2TP is a standard by OMG, it still lakes proper semantics and tool support for U2TP is also very limited.

4. Platform Independence

By definition of model-based and model-driven approaches, they should provide platform independence. However, its a big challenge to provide platform independence at every level

during regression testing. For example, as discussed earlier, test modelling is another way to introduce the platform independence for test specification but due to immaturity of test modelling language it is hard to make the test specification platform independent.

5. Various Dimensions of Evolution

Due to the fact that in MDA models are available at different levels of abstractions, there are CIM (Computation Independent Models), PIM (Platform Independent Models) and PSM (Platform Specific Models). Hence, evolution is performed both vertically and horizontally as compared to the traditional model-based development where evolution is normally horizontal (Briand, Labiche, & Yue, 2009).

Another important aspect is meta-model evolution. In MDA every model should conform to some meta-model. In case the meta-model is changed or extended, it can create version compatibility issues. Another type of evolution is platform and technical infrastructure evolution where chains of code generators, runtime environments, dependency tools are changed. The effect of such evolution can be same as meta-model evolution (Visser, Warmer, & Deursen, 2007).

6. Models as Code

B. Emerging Approaches in the field of Model-driven Regression Testing

As discussed earlier, MDRT is about introducing MDA practices such as model transformations, platform independent models for test generation and platform independent models for test specifications. The approaches mentioned here are recent and most of them only present research ideas. The application of these ideas is still a work to be done.

Naslavsky et al's approach is to use traceability and model transformations for regression test selection (Naslavsky, Ziv, & Richardson, 2010). They used sequence diagrams for baseline test suite generation. The idea was in a preliminary phase and no results are reported for the success of the approach. Pilskalns et al. discuss another interesting approach for regression testing the designs models directly instead of testing the implementation (Pilskalns, Uyan, & Andrews, 2006). This means that the test cases corresponds directly to the model and will be executed on the design models instead of the code.

Silva et al. present a concern-based approach for model-driven system; however, they do not discuss the use of model transformation languages or model-based test specification in their approach (Silva, Budnik, Hasling, McKenna, & Subramanyan, 2010). We are evaluating the above mentioned approaches in our analysis in Section VI.

Another recent approach is by one of the authors of this chapter and it is a work in progress. The idea consists of using the MDA transformations for the baseline test suite generation with the integrated traceability. However, instead of ad-hoc test suite representations as adopted by other

approaches, we are using U2TP[2] (U2TP1.0, 2005), a test modelling language, for the test specification. Use of a test modelling language not only increases the portability of the test suites but also the traceability maintenance is easy between design models and test models rather than test code. Another important aspect is that dedicated test specification languages cover several aspects of test specification such as test architecture, test time and test data modelling; hence, the impact analysis is fine grained and covers various aspects of test suites.

VI. EVALUATION OF MODEL-BASED REGRESSION TESTING APPROACHES

In this section, we present a comprehensive evaluation of the existing MBRT approaches. Considering the challenges identified in the previous sections, it is very important to evaluate the ability of the existing MBRT approaches to deal with all these challenges. This will help the researches working in the field of MBRT to identify the weaknesses of the existing approaches and to further continue the research in those areas to improve those weaknesses. For the practitioners, however, this evaluation can help to select the approach that suites their particular needs and project's requirement. For the tool developers, the analysis provides a guideline to identify the state of automation in the field and it provides the motivation to build new tools to address the need of testers during software maintenance.

Before performing the analysis, we identified some research questions and formulated detailed evaluation criteria to address these research questions. We discuss them in the subsequent sections. The criteria presented here is equally applicable to MDRT approaches as well; hence, we include the MDRT approaches discussed in V.B in our analysis as well.

There are some existing surveys on model-based regression testing (Fahad & Nadeem, 2008), (Mahdian, Andrews, & Pilskalns, 2009), (Engstroem, Runeson, & Skoglund, 2010). However, the major difference of the survey presented in this chapter is the level of detail. We identified a set of research questions for each criterion to have a better understanding of weakness and strengths of approaches in certain areas. The criteria presented in this chapter are very comprehensive as compared to any other criteria developed to evaluate model-based regression testing techniques in the literature. It contains ***9 criteria*** and ***27 inquiries*** corresponding to them. Our criteria are discussed in detail in Section VI.C. Moreover, before the comparison, we divided the approaches into 6 different sets for better understanding. These sets are explained in the start of VI.D.

A. Research Questions

According to the challenges discussed in section IV, we identified the following important research questions.

1. How much support is available for each regression testing step discussed in section II.A?

[2] U2TP (UML 2 Testing Profile) is a standard test specification language by Object Management Group (OMG). The preliminary building blocks of U2TP are "Test Architecture", "Test Behaviour", "Test Data" and "Test Time". The current available version of U2TP is 1.0; however, the next revisions are also in progress.

2. Whether the techniques provide strong support for change identification and impact analysis to deal with change propagation or not?
3. How much platform independence the existing techniques provide?
4. Do the techniques provide adequate test suite classification and how much reduction they promise?
5. How mature are the approaches in the field and how much support for the users they provide?
6. How much automation is supported by the model-based regression testing techniques?

Besides the research questions related to the challenges discussed earlier, we added two more research questions for better understand the techniques.

1. How many approaches exist for each testing level, i.e. unit, integration and system level?
2. What is the tendency of coverage of structural and behavioural models by the available model-based regression testing techniques?

In the next section we explain how we selected the studies for our analysis. We eliminate the irrelevant studies by establishing study selection criteria.

B. Study Selection

To eliminate the irrelevant studies before performing the analysis we used the following two filters.

1. All the approaches we considered are from the year 2000 and onwards. The reason is that the studies before that do not use mature modelling languages and are not applicable to the present scenarios.
2. We consider only those approaches that use models as input; hence, ignoring the approaches that use source code or specification and design artefacts other then models.

In the following sub sections, we discuss the study selection process and the eliminated studies.

1. Selected Studies

Initially the research papers were selected on the basis of title and abstract relevance. To search for the relevant papers we relied mostly on the existing knowledge of the authors in the field; hence, the initial papers were mostly already known by the authors. To make our search more reliable, we also thoroughly searched the references of all available papers and finally to get confidence on a complete coverage we searched the most popular databases IEEE digital Library, ACM Digital Library, SpringerLink and ScienceDirect using different combinations of following keywords.

“Regression Testing, Models, UML, Design Models, Model-driven, Model-based, specification-based, evolution”

After applying the initial selection filters and filtering them based on abstract and title relevance we obtained total 17 studies corresponding to 31 research papers. The list of these selected studies and their corresponding references are given in Appendix A.

2. Excluded Studies

We excluded all those approaches from our analysis which are published before year 2000. Interested readers can have a look on these studies in additional readings section in Appendix XI. Some other studies are excluded because the input used by these approaches was not models, they were either textual requirements or version data modelled using OCL. The list of these studies is also included in the additional reading section.

C. Analysis Criteria

In this section, we discuss the analysis criteria we developed to answer the above mentioned research questions. The criteria contain a set of further questions/inquiries satisfying the criteria. Table 2 presents each criterion and the inquiries related to the criterion.

Table 2: Analysis Criteria for MBRT Approaches

1. Criteria name: *Testing Level*
Inquiries **Inq-1:** What is the testing level addressed by the approach?
2. Criteria Name: *Model Coverage*
Inquiries **Inq-2:** The approach covers structural modelling diagrams, behavioural modelling diagrams or both? Structural only Behavioural only Both structural and behavioural **Inq-3:** What are the input models used by the approach? **Inq-4:** What is the test specification language used by the approach?
3. Criteria Name: Support for Regression Testing Steps (RTS)
Inquiries **Inq-5:** Baseline test suite establishment **Inq-6:** Change identification (see Criteria 4) **Inq-7:** Change impact analysis (see Criteria 5) **Inq-8:** Regression test selection (see Criteria 7) **Inq-9:** Repair broken test cases **Inq-10:** Test result analysis
4. Criteria Name: *Change Identification*

Inquiries

Inq-11: Does the approach provide sound change definitions for modifications in the system?

Inq-12: How many change types were considered by the approach?

Inq-13: Does the approach discusses the change detection mechanism and rules for change detection between different versions of the system?

5. Criteria Name: *Impact Analysis*

Inquiries

Inq-14: How the traceability between several design and test artefacts was established? (Traceability Support)

Explicit Traceability
Implicit Traceability
No traceability

Inq-15: Does the approach perform impact analysis?

(Inter-model) Impact Analysis within several models
(Intra-model) Impact analysis within 1 model
No Impact Analysis

Inq-16: Does the approach consider the dependency types?

6. Criteria Name*: Platform Independence*

Inquiries

Inq-17: Does the approach use platform independent specification and design models (input models) or they are polluted with implementation specific concerns.

Inq-18: Does the approach support platform independent test modelling?

Inq-19: Does the Tool support/implementation provided by the approach is specific to some platform?

7. Criteria Name: *Efficiency*

Inquiries

Inq-20: how much reduction is achieved? (We are collecting the data provided by the authors and are not measuring the reduction ourselves)

Inq-21: Does the approach provide some effective classification for the regression test selection?

Modified test cases are identified
Obsolete test cases are identified
The elements for which new test cases are required are identified

8. Criteria Name: *Maturity and Support*

Inquiries

Inq-22: Is the approach evaluated on any case study or does any experimental evaluation was present?

No: 0
Just an example (only some example diagrams are used)
Small case study (Less than 100 Test cases OR less than 10 components)
Medium case study (100-500 test cases OR 5-20 Components)
Large case study(More than 500 test cases OR more than 20 Components)

Inq-23: Is the approach compliant to the standards for input models?

Complete compliance with a standard
Partial Compliance (Notations and extensions applicable to standards)
No standard compliance

Inq-24: Is the approach compliant to the standards for test specifications?
Complete compliance with a standard
Partial Compliance (Notations and extensions applicable to standards)
No standard compliance

Inq-25: What is the degree of documentation and support?

Just a paper
Detailed Method description
Plus Tutorials, templates and examples

9. Criteria Name: *Automation and Tool Support*

Inquiries

Inq-26: Were the ideas defined by some algorithmic details or not?

Inq-27: Does the approach provide tool support or not?
Full tool support
Prototype tool
No tool support

A very important criterion to evaluate the regression testing techniques was presented by Rothermel & Harrold. This criterion includes 5 parameters, Inclusiveness, Precision, Efficiency, Generality, and Accountability (Rothermel & Harrold, 1994).

According to the criteria, inclusiveness is the extent to which modification revealing test cases are added in the regression test suite and Precision is the extent to which non-modification revealing test cases are omitted from the regression test suite. To determine inclusiveness is not possible without an experimental evaluation. For us experimental evaluation of 18 approaches

was not possible due to time, resources and non-availability of detailed information for all the approaches. However, inclusiveness and precision in our case can be deduced by considering criterion 4 and 5 in Table 2.

Efficiency, according to Rothermel & Harrold, is determined by considering space and time requirements of the regression testing techniques. This criterion also cannot be determined without experimental evaluations. We defined efficiency as the reduction capability of regression testing techniques and ability to classify the original test suite for regression testing effectively. The criterion 7, in Table 2 presents this criterion. Rothermel & Harrold defined generality as the ability to function in a wide and practical range of solutions. In our case, generality can be deduced by considering the inquiries in criterion 1.

Harrold et al defined Accountability as the extent to which regression testing approaches promote structural coverage criteria. To us, application of a structural coverage is concerned mostly with test prioritization approaches and most of the regression testing approaches do not deal with application of structural coverage criteria; hence, we do not consider this in our analysis. Another very important issue is scalability of the approaches. Scalability can be deduced by considering criterion 1 and 2 in Table 2.

D. Detailed Analysis of MBRT Approaches

In this section, we present the detailed analysis of the selected studies for each criterion. We classified the approaches into 5 different categories according to the models they are using. Following is the classification of the approaches.

Specification Level Activity-based Approaches: These are approaches with use specification models like use cases and activity diagrams as input for regression testing.

Approaches involving Both Specification and Design Artefacts: These approaches either perform system level testing or involve multiple testing levels such as unit, integration and system levels. They take both specification and design models as input.

Design Level State-based Approaches: These approaches take event-based models as input for regression testing. Most of these approaches take variants of finite state machines as input.

Design Level Component-based Approaches: These approaches are specific for component-based regression testing.

Design Level Approaches using Sequence and Class diagrams: These approaches takes sequence and class diagrams as input for regression testing.

Design Level Miscellaneous Approaches: These approaches are the design level approaches which do not fall into any above mentioned category.

In the following, we present the analysis of each criterion using for each classification of approaches mentioned above. Each criterion contains a corresponding analysis table and a "critical Issues" section highlighting the major findings of the analysis of the criterion.

1. Analysis of Criterion 1 & 2–Testing Level & Model Coverage

The criteria states which input models and test specification language the approaches use, and what levels of testing the approaches address? The inquiries other then inquiry 4 are more elaborative then analytical. These are used to understand the nature of the approaches better. Table 3 presents the analysis of the selected approaches for these criteria. According to Table 3, from the total 17 approaches selected for the analysis, 7 of the approaches deal with system level testing, 2 approaches are about component-based testing, 4 approaches are integration level approaches and 7 approaches can be applied at unit level.

Most of the approaches do not use any particular test specification language for the test representation. Most of the approaches specify test in their custom styles. However, one approach uses JUnit for test specification and one use a XML-based representation of the test cases. The corresponding critical issues section contains the critical points extracted in the light of analysis.

Table 3: Analysis of Model-based Regression Testing Approaches for the Criterion "*Testing Level*" & "*Model Coverage*"

Approaches▼	**Study ID▼**	**Inq-1:** What is the testing level addressed by the approach?	**Inq-2:** The approach covers structural modelling diagrams, behavioural modelling diagrams or both	**Inq-3:** What are the input models used by the approach?	**Inq-4:** What is the test specification language used by the approach?
		Testing Level	**Model Coverage**		
Specification Level Approaches (Activity-based)	**Study-1: (Gorthi et al., 2008)**	System Level	Behavioural Only	Structured Activity diagram	None
	Study-2: (Chen et al., (a), 2002,2003)	System Level	Behavioural only	Activity Diagram	None
	Study-3: (Silva et al., 2010)	Functional Testing using Category partition	Both structural and behavioural	Main artefact(Activity diagram) others (Class Diagram and Sequence Diagram)	None
Approaches involving Both Specification and Design Artefacts	**Study-4: (Mansour et al., 2007, 2011)**	Unit, Integration and System Level	Behavioural and Structural both	Interaction Overview Diagram, Sequence Diagram, Class Diagram	None
	Study-5: (Briand et al., 2002, 2003, 2009)	System Level	Behavioural and Structural both	Use case Diagram, Sequence Diagram, Class Diagram	None

	Study-6: (Deng et al., 2004)	Black-box system testing	Behavioural and Structural both	Use case Diagram, Class Diagram, Sequence Diagram, Activity Diagram, State Chart	None
Design Level Approaches (State-based)	**Study-7: (Chen et al., (b), 2007, 2009)**	Unit Level	Behavioural Only	EFSM (Extended Finite State Machine), SDL	None
	Study-8: (Korel et al., 2002)	Unit Level	Behavioural Only	EFSM	None
	Study-9: (Beydeda et al., 2000)	Unit Level	Behavioural Only	Class State Machine and	None
	Study-10: (Farooq et al., 2007, 2010)	Unit and Integration Level	Behavioural and Structural both	Class Diagram, State Machine	XML
Design Level Approaches (Sequence diagram and Class diagram)	**Study-11: (Ali et al., 2007)**	System Level	Behavioural and Structural both	Class Diagram, Sequence Diagram	None
	Study-12: (Pilskalns et al., 2006)	System Level Testing of UML designs	Behavioural and Structural both	Class Diagram, Sequence Diagram,	None
	Study-13: (Naslavsky et al., 2007, 2009, 2010)	Unit and Integration Level	Behavioural and Structural both	Class Diagram, Sequence Diagram	JUnit for concrete test representation.
	Study-14: (Jeron et al., 1999, 2000)	Integration Testing	Structural	Class Diagram	None
Design Level (Component-based)	**Study-15: (Muccini et al., (b), 2005, 2006, 2007)**	Component-based Testing	Behavioural and Structural both	Sequence Diagram, Component Diagram, State Machine (FSP Algebra)	None
	Study-16: (Wu & Offet, 2003)	Component Level	Behavioural and Structural both	Class Diagram, State Chart, Collaboration Diagram	None
Design Level Approaches (Miscellaneous)	**Study17: (Martins et al., 2005)**	U nit Testing	Behavioural Only	Activity Diagram for a class implementation logic	None

Critical Issues–Testing Level and Model Coverage:

The existing MBRT approaches use no standard test specification language such as TTCN or U2TP. They only use ad-hoc test representations and in most of the cases test specifications are very abstract and how these abstract test cases will be later mapped to the concrete test cases is unclear.

2. Analysis of Criterion 3–Support for Regression Testing Steps

This criterion shows how much support for the regression testing steps discussed in section II.A is provided by the existing MBRT approaches. Table 4 presents the analysis of the selected MBRT approaches for the inquiries of the criterion. For the discussion on the critical findings of the analysis please refer to the corresponding critical issues section.

Table 4: Analysis of MBRT Approaches for the criterion "*Support for Regression Testing Steps*"

Approaches▼	Study ID▼	**Inq-5:** Baseline test suite establishment	**Inq-6:** Change identification (see Criteria 4)	**Inq-7:** Change impact analysis (see Criteria 5)	**Inq-8:** Regression test selection (see Criteria 7)	**Inq-9:** Repair broken test cases	**Inq-10:** test result analysis
Specification Level Approaches (Activity-based)	**Study-1: (Gorthi et al., 2008)**	Not discussed	Not discussed	No Impact Analysis	Yes (risk and cost based)	No	No
	Study-2: (Chen et al., (a), 2002,2003)	Not discussed	Not discussed	No Impact Analysis	Yes (risk and cost based)	No	No
	Study-3: (Silva et al., 2010)	Functional test cases generated by TDE-UML using category partitioning method	Yes (online change identification)	Yes	Yes (obsolete, reusable, re-testable)	No	No
Approaches involving Both Specification and Design Artefacts	**Study-4: (Mansour et al., 2007, 2011)**	Not discussed	Partial(Change Definitions not provided)	Yes	Partial (only affected)	No	No
	Study-5: (Briand et al., 2002, 2003, 2009)	Yes it is referred to a previous approach	Yes	Yes	Yes	No	No
	Study-6: (Deng et al., 2004)	Rules for All branch, boundary and Faulty testing are discussed	No	Very Limited	No	No	No
Design Level Approaches (state-based)	**Study-7: (Chen et al., (b), 2007, 2009)**	Yes (As the work is a continuation of Korel et al. technique)	No	Yes(Partial, only intra model)	Yes (Only affected test cases are identified)	No	No
	Study-8: (Korel et al., 2002)	Yes	No	Yes (Partial only intra-model)	Yes (Only affected test cases are identified)	No	No
	Study-9: (Beydeda et al., 2000)	Yes	No	Only between specification and source code	Yes (Only affected test cases are identified)	No	No
	Study-10: (Farooq et al., 2007, 2010)	Yes, but manual test generation using transition tree method	Yes	Yes	Yes (obsolete, reusable, re-testable)	No	No

Approaches▼	Study ID▼	**Inq-5:** Baseline test suite establishment	**Inq-6:** Change identification (see Criteria 4)	**Inq-7:** Change impact analysis (see Criteria 5)	**Inq-8:** Regression test selection (see Criteria 7)	**Inq-9:** Repair broken test cases	**Inq-10:** test result analysis
Design Level Approaches (Sequence diagram and Class diagram)	**Study-11: (Ali et al., 2007)**	Not discussed	Partial	Yes	Yes	No	No
	Study-12: (Pilskalns et al., 2006)	Yes, UML design testing producing UML test cases	yes	Yes	Yes (obsolete, reusable, new)	No	No
	Study-13: (Naslavsky et al., 2007, 2009, 2010)	Sequence diagram based test generation.	yes	Yes	yes	No	No
	Study-14: (Jeron et al., 1999, 2000)	Integration testing using test dependency graph	No	No	Limited Discussion	No	No
Design Level Approaches (Component-based)	**Study-15: (Muccini et al, 2005, 2006, 2007)**						
	Study-16: (Wu & Offet, 2003)	Not discussed	No	Partial (Intra-model only)	New, retestable	No	No
Design Level Approaches (Miscellaneous)	**Study17: (Martins et al., 2005)**	Paths of a Behavioral Control Flow Graph (BCFG)	yes	No	Yes	No	No

Critical Issues–Support for RTS:

1. The existing regression testing approaches do not consider two important steps of the regression testing; how the selected test should be repaired and analyzed.
2. A lot of the approaches provide limited support for change identification and impact analysis as well (see criterion change identificationVI.D.3 and impact analysis VI.D.4 for further details.)

3. Analysis of Criterion 4–Change Identification

Change identification is an important activity in regression testing. This section provides the analysis of the inquiries corresponding to the change identification for the selected MBRT approaches. The analysis is presented in Table 5. For the critical issues refer to the corresponding critical issues section.

Table 5: Analysis of the MBRT Approaches for the criterion "*Change Identification"*

Approaches▼	Study ID▼	**Inq-11:** Does the approach provide sound change definitions for modifications in the system?	**Inq-12:** How many change types were considered by the approach?	**Inq-13:** Does the approach discusses the change detection mechanism and rules for change detection between different versions of the system?
Specification Level Approaches (Activity-based)	**Study-1: (Gorthi et al., 2008)**	No	Add, delete, modify	No
	Study-2: (Chen et al., (a), 2002,2003)	No	Modify	No
	Study-3: (Silva et al., 2010)	No	Add, delete, modify	Yes (using time stamps and edit time monitoring)
Approaches involving Both Specification and Design Artefacts	**Study-4: (Mansour et al., 2007, 2011)**	No	Modify	Yes
	Study-5: (Briand et al., 2002, 2003, 2009)	Yes	Addition of elements, Deletion of elements, Modifications of element properties	
	Study-6: (Deng et al., 2004)	No	Modify	No
Design Level Approaches (state-based)	**Study-7: (Chen et al., (b), 2007, 2009)**	No	Add ,delete, modify	No
	Study-8: (Korel et al., 2002)	No	Add ,delete	No
	Study-9: (Beydeda et al., 2000)	No	Modify	No
	Study-10: (Farooq et al., 2007, 2010)	Yes	Add ,delete, modify	Yes
Design Level Approaches (Sequence diagram and Class diagram)	**Study-11: (Ali et al., 2007)**	Yes (Only a limited set)	Modify	No
	Study-12: (Pilskalns et al., 2006)	Yes	Add, delete, modify	Yes
	Study-13: (Naslavsky et al., 2007, 2009, 2010)	Yes	Add, delete, modify (using EMFCompare)	Yes
	Study-14: (Jeron et al., 1999, 2000)	No	No	No
Design Level Approaches (Component-based)	**Study-15: (Muccini et al, 2005, 2006,2007)**	Yes	Add, delete, modify	
	Study-16: (Wu & Offet, 2003)	No	Add, delete, modify	No
Design Level Approaches (Miscellaneous)	**Study17: (Martins et al., 2005)**	No	Add, Remove	Yes

Critical Issues–Change Identification:

1. Most of the approaches do not provide the sound change definitions to detect the changes in the models. If a change is not defined it cannot be detected later.
2. The existing MBRT approaches only consider the primary change types (Add, Delete, and Property Modification) and the effect of other complex change types discussed in section II.A.2 is not considered by any of the approaches.
3. A lot of approaches also do not discuss the rules for change identification between two versions of the system.

4. Analysis of Criterion 5–Impact Analysis

As discussed earlier, impact analysis is one of the most important activities in the regression testing. This section presents the analysis of the selected approaches for their capabilities to support impact analysis. Table 6 presents the analysis of the MBRT approaches for different inquiries corresponding to the impact analysis. The corresponding critical issues section discusses the critical findings of the analysis.

Table 6: Analysis of the MBRT Approaches for the criterion "*Impact Analysis*"

Approaches▼	Study ID▼	**Inq-14:** How the traceability between several design and test artefacts was established	**Inq-15:** Does the approach perform impact analysis?	**Inq-16:** Does the approach consider the dependency types?
Specification Level Approaches (Activity-based)	**Study-1: (Gorthi et al., 2008)**	No Traceability	No Impact Analysis	None
	Study-2: (Chen et al., (a), 2002,2003)	Explicit (Traceability Matrix)	No Impact Analysis	None
	Study-3: (Silva et al., 2010)	Explicit traceability links are established	Supported using traceability links (between artefacts, models and test cases)	Not discussed
Approaches involving Both Specification and Design Artefacts	**Study-4: (Mansour et al., 2007, 2011)**	Implicit Traceability	Between class diagram, IOD and SD	
	Study-5: (Briand et al., 2002, 2003, 2009)	Implicit, (using sequence matching)	Between CD, SD and UC	
	Study-6: (Deng et al., 2004)	No Traceability	Very few and abstract rules for impact analysis	None
Design Level Approaches (state-based)	**Study-7: (Chen et al., (b), 2007, 2009)**	No Traceability	Intra-model	
	Study-8: (Korel et al., 2002)	No Traceability	Intra-model	

	Study-9: (Beydeda et al., 2000)	No Traceability	Intra-model	
	Study-10: (Farooq et al., 2007, 2010)	Implicit Traceability	Inter-model (Between CD and SM)	
Design Level Approaches (Sequence diagram and Class diagram)	**Study-11: (Ali et al., 2007)**	Implicit Traceability	Between CD and SD	
	Study-12: (Pilskalns et al., 2006)	Implicit traceability	Inter model (CD, SD OMDG, test cases)	Use dependency
	Study-13: (Naslavsky et al., 2007, 2009, 2010)	Explicit Traceability (in form of a traceability model).	Between SD and CD	Not discussed
	Study-14: (Jeron et al., 1999, 2000)	None	No Impact Analysis	Contractual and Implementation Dependencies
Design Level Approaches (Component-based)	**Study-15: (Muccini et al, 2005, 2006, 2007)**	Implicit		
	Study-16: (Wu & Offet, 2003)	No Traceability	Intra Model	Control and data dependencies (within same model)
Design Level Approaches (Miscellaneous)	**Study17: (Martins et al., 2005)**	Implicit traceability	No Impact Analysis	None

Critical Issues–Impact Analysis:

1. Most of the regression testing approaches do not support the concept of explicit traceability, i.e., traceability is not maintained at the time of test generation so that it could be used to perform impact analysis later during regression testing.
2. Some approaches provide the traceability, most of them use ID and name comparison to find the relations which is a week approach and might miss many relations.
3. A lot of MBRT approaches either do not support impact analysis or support impact analysis within one diagram. The relations between several diagrams are considered only in a few approaches.
4. Most of the approaches do not consider different types of dependencies. The type of dependency can affect the way selected test should be treated later. Only a few approaches consider control and data dependencies for intra-model impact analysis.

5. Analysis of Criterion 6–Platform Independence

This section provides the analysis of selected approaches for the criterion Platform independence. The criterion considers the platform independence of input models and test specification models and the implementation platform. Table 7 presents the analysis of the criterion for the selected approaches. Further issues are discussed the corresponding critical issues section.

Table 7: Analysis of the MBRT Approaches for the criterion "*Platform Independence*"

Approaches▼	Study ID▼	**Inq-17:** Does the approach use platform independent specification and design models?	**Inq-18:** Does the approach support platform independent test modelling?	**Inq-19:** Does the Tool support/implementation provided by the approach is specific to some platform?
Specification Level Approaches (Activity-based)	**Study-1: (Gorthi et al., 2008)**	Yes (Extended Activity Diagram)	Test Modelling not discussed (Only test paths which are platform Independent)	No implementation
	Study-2: (Chen et al., (a), 2002,2003)	Yes (Activity Diagram)	Test Modelling not discussed (Only test paths which are platform Independent)	No implementation
	Study-3: (Silva et al., 2010)	Yes (Activity, sequence and Class diagram)	Test Modelling not supported (Custom test procedures, containing test steps and data bindings)	TDE/UML developed in Java, also available as in-house eclipse plug-in by SIEMENS corporation
Approaches involving Both Specification and Design Artefacts	**Study-4: (Mansour et al., 2007, 2011)**	Yes (UML)	Test Modelling is not supported (test paths depicting sequence of methods)	No Implementation
	Study-5: (Briand et al., 2002, 2003, 2009)	Yes (UML)	Test Modelling not supported (Tests are in form of action sequence triplets)	Java 2 Platform, POET Object Server Suite, However UML meta model implementation is developed internally
	Study-6: (Deng et al., 2004)	Yes (UML)	Test Modelling not supported and form of the test cases is not discussed	No Implementation
Design Level Approaches (state-based)	**Study-7: (Chen et al., (b), 2007, 2009)**	Yes, SDL is platform Independent	Test Modelling is not discussed (Probably in form of SDL sequences)	No Implementation
	Study-8: (Korel et al., 2002)	Yes	Test Modelling is not discussed (Probably in form of sequences)	No Implementation
	Study-9: (Beydeda et al., 2000)	No (CSC is not a standard artefact and CSIG is constructed using both source code and specification)	Tests are not platform independent, contain source code information	No Implementation
	Study-10: (Farooq et al., 2007, 2010)	Yes (UML)	Yes XML representation of state test sequences	Java based Implementation in Eclipse platform, UML2 plug-in for Eclipse

Design Level Approaches (Sequence diagram and Class diagram)	**Study-11: (Ali et al., 2007)**	Yes (UML)	Test Modelling is not supported (test paths depicting paths of CCFG)	No Implementation
	Study-12: (Pilskalns et al., 2006)	Yes (UML)	Test Modelling not supported (test cases are in form of graph tuples)	No Implementation
	Study-13: (Naslavsky et al., 2007, 2009, 2010)	Yes (UML)	Test Modelling is not supported. Abstract test cases are sequences of sequence diagram.	Java-based implementation using Eclipse plugins. (EMFCompare, ATL and UML2 Plugins for Eclispe)
	Study-14: (Jeron et al., 1999, 2000)	Yes UML	Test Modelling not supported	No Implementaion
Design Level Approaches (Component-based)	**Study-15: (Muccini et al,2005, 2006, 2007)**			
	Study-16: (Wu & Offet, 2003)	Yes (UML)	Test Modelling not supported	No Implementation
Design Level Approaches (Miscellaneous)	**Study17: (Martins et al., 2005)**	Yes(UML)	Test Modelling not supported	No Implementation

Critical Issues–Platform Independence:

1. The concept of test modelling which supports platform independent test suites is not supported by the existing model-based regression testing techniques.
2. Almost all the prototype implementations provided by the approaches are compliant to the Java platform and support for other platforms is not provided by the approaches

6. Analysis of Criterion 7–Efficiency

This criterion analyzes the efficiency of the approaches by analyzing the reduction capabilities and by considering the ability of test suite classification of the selected approaches. Table 8 presents the analysis of the selected approaches for the corresponding inquiries. The corresponding critical issues section highlights the general issues considering the evaluation of the approaches for efficiency.

Table 8: Analysis of the MBRT Approaches for the criterion "*Efficiency*"

Approaches▼	**Study ID▼**	**Inq-20:** how much reduction is achieved?	**Inq-21:** Does the approach provide some effective classification for the regression test selection?
Specification Level Approaches	**Study-1: (Gorthi et al.,**	Not Discussed	(Added ,Affected) test

(Activity-based)	**2008)**		cases
	Study-2: (Chen et al., (a), 2002,2003)	Approx 70%	(Added, affected and Prioritized safety tests)
	Study-3: (Silva et al., 2010)	Not discussed	Obsolete, Reusable, Re-testable &New
Approaches involving Both Specification and Design Artefacts	**Study-4: (Mansour et al., 2007, 2011)**	92-100%	Affected
	Study-5: (Briand et al., 2002, 2003, 2009)		Obsolete, reusable and re-testable
	Study-6: (Deng et al., 2004)	Not Discussed	Not Discussed
Design Level Approaches (state-based)	**Study-7: (Chen et al., (b), 2007, 2009)**	83-99.09 %	Affected
	Study-8: (Korel et al., 2002)	83-99 %	Affected
	Study-9: (Beydeda et al., 2000)	Not Discussed	Affected
	Study-10: (Farooq et al., 2007, 2010)	Up to 63%	Obsolete, reusable and re-testable
Design Level Approaches (Sequence diagram and Class diagram)	**Study-11: (Ali et al., 2007)**	Not Discussed	
	Study-12: (Pilskalns et al., 2006)	93 %	New, Reusable, Obsolete
	Study-13: (Naslavsky et al., 2007, 2009, 2010)	Not discussed	Obsolete, Reusable and Retestable
	Study-14: (Jeron et al., 1999, 2000)	Not discussed	Affected
Design Level Approaches (Component-based)	**Study-15: (Muccini et al, (a), 2005, 2006)**		Retestable, New
	Study-16: (Wu & Offet, 2003)	Not Discussed	Modified and New
Design Level Approaches (Miscellaneous)	**Study17: (Martins et al., 2005)**	Varies version to version	Reusable, Retestable, Obsolete

Critical Issues–Efficiency:

Although some MBRT approaches report the reduction achieved by applying their approaches on the case studies. However this reduction depends on how much modifications they considered and how complex were the case studies. To evaluate the reduction capabilities of the approaches, the approaches should be empirically analysed with the same set of the approaches.

7. Analysis of Criterion 8–Maturity and Support

This criterion evaluates the maturity and support provided by the MBRT approaches by focusing on three main issues; case studies or evaluations, standard compliance and available documentation and support. Table 9 shows the results of the analysis of the selected approaches. We discuss the critical issues related to this criterion in the corresponding critical issues section.

Table 9: Analysis of the MBRT Approaches for the criterion "*Maturity and Support*"

Approaches▼	Study ID▼	**Inq-22:** Is the approach evaluated on any case study or does any experimental evaluation was present?	**Inq-23:** Is the approach compliant to the standards for input models?	Inq-24: Is the approach compliant to the standards for test specifications?	**Inq-25:** What is the degree of documentation and support?
Specification Level Approaches (Activity-based)	**Study-1: (Gorthi et al., 2008)**	retail system case study, 342 Test cases MEDIUM	Partial Compliance (Activity like notation with extensions)	No Standard Compliance	One Conference Paper
	Study-2: (Chen et al., (a), 2002,2003)	3 IBM WEB SPHERE Components, 306 test cases): MEDIUM	Partial Compliance (Activity like notation with extensions)	No Standard Compliance	Two Conference Papers One Master's Thesis
	Study-3: (Silva et al., 2010)	**None**	Activity diagram with extended properties	No Standard Compliance	One Conference Paper
Approaches involving Both Specification and Design Artefacts	**Study-4: (Mansour et al., 2007, 2011)**	(Evaluation on three different case studies. Max Test suite size is 90 MEDIUM)	UML 2.0 (full Compliance)	No Standard Compliance	A conference paper A journal paper
	Study-5: (Briand et al., 2002, 2003, 2009)	LARGE(596 test cases)	UML(full Compliance)	No Standard Compliance	A Conference Paper A journal Paper A technical Report
	Study-6: (Deng et al., 2004)	No case study and evalutaion	UML (Version unknown)	No standard compliance	A conference paper
Design Level Approaches (state-based)	**Study-7: (Chen et al., (b), 2007, 2009)**	LARGE (Models: 6 SDL models max No of test case: 1691)	SDL (Full Compliance)	No Standard Compliance	One Conference Paper One Journal Paper
	Study-8: (Korel et al., 2002)	Just an example	Partial Compliance (State machine)	No Standard Compliance	Conference Paper
	Study-9: (Beydeda et al., 2000)	Just an example (Class Account)	Partial Compliance (State machine)	No Standard Compliance	Conference Paper
	Study-10: (Farooq et al., 2007, 2010)	LARGE (723 test cases Enrolment system Case study)	UML(full Compliance)	No Standard Compliance	One Conference Paper One Workshop Paper Masters Thesis Tool source code
Design Level Approaches (Sequence diagram and Class	**Study-11: (Ali et al., 2007)**	(Just an example of ATM system No TC: 6)	UML (Full Compliance)	No Standard Compliance	A conference paper

diagram)	**Study-12: (Pilskalns et al., 2006)**	Transcoder Component of Battik toolkit 32 Classes and sequence diagrams and 52 test cases	UML (Version Unknown)	No Standard Compliance	A conference Paper
	Study-13: (Naslavsky et al., 2007, 2009, 2010)	PIMS), and the Aqualush case studies. Number of component, diagrams or test cases are not discussed	UML (Version 2)	Non standard test specifications (Abstract test cases are written in Custom format. For concrete test cases JUnit is used.)	3 Conference Papers
	Study-14: (Jeron et al., 1999, 2000)	SMDS Server case study in Telecommunication domain containing 38 classes	UML (Applicable to any version of class diagram)	No Standard Compliance	1 journal paper and one conference
Design Level Approaches (Component-based)	**Study-15: (Muccini et al, 2005, 2006, 2007)**	(15 Components, number of test cases not specified MEDIUM)	Charmy Language		
	Study-16: (Wu & Offet, 2003)	An Example of ATM system.	UML	No Standard Compliance	A conference paper
Design Level Approaches (Miscellaneous)	**Study17: (Martins et al., 2005)**	Common C++ Library casestudy (2 classes having 16 and 8 methods) 6 versions of each class are considered	UML (Version Unknown)	No Standard Compliance	A conference Paper

Critical Issues–Maturity and Support:

1. Although some cases studies are available for evaluating MBRT approaches for their applicability; however, most of the studies do not evaluate the approaches for efficiency and reduction. Unavailability of comparative evaluations is also a major issue in MBRT.
2. Standard compliance, especially for test specification is a major deficiency in the existing MBRT approaches.
3. The degree of documentation and support is very limited in the existing MBRT approaches. The only support material available for most of the approaches is a conference or a workshop paper. Very few approaches also publish their results in a journal paper which contains relatively more detailed information. However, none of the approaches provide tutorials or other artefacts to support their approach. Most of the approaches also do not provide any other kind of documentation and tutorials for their tools and methodology.

8. Analysis of Criterion 9–Automation and Tool Support

This criterion evaluates the degree of automation and tool support by the existing MBRT approaches. Table 10 presents the analysis of the selected approaches for two further inquiries. The approach provides the algorithmic details of the ideas or not and the ideas or implemented in a tool or not. The corresponding critical issues section discussed the critical findings of the analysis.

Table 10: Analysis of the MBRT Approaches for the criterion "*Automation and Tool Support*"

Approaches▼	Study ID▼	**Inq-26:** Were the ideas defined by some algorithmic details or not?	**Inq-27:** Does the approach provide tool support or not?
Specification level MBRT Approaches	**Study-1: (Gorthi et al., 2008)**	Yes	No
	Study-2: (Chen et al., (a), 2002,2003)	Yes	No
	Study-3: (Silva et al., 2010)	Yes	Yes (TDE/UML) by Siemens Inc.
Approaches involving Both Specification and Design Artefacts	**Study-4: (Mansour et al., 2007, 2011)**	Yes	No
	Study-5: (Briand et al., 2002, 2003, 2009)	Yes	Prototype (RTS Tool)
	Study-6: (Deng et al., 2004)	Yes	No
Design Level Approaches (state-based)	**Study-7: (Chen et al., (b), 2007, 2009)**	Yes	No
	Study-8: (Korel et al., 2002)	Yes	No
	Study-9: (Beydeda et al., 2000)	Yes	No
	Study-10: (Farooq et al., 2007, 2010)	Yes	Prototype tool (START)
Design Level Approaches (Sequence diagram and Class diagram)	**Study-11: (Ali et al., 2007)**	Yes	No
	Study-12: (Pilskalns et al., 2006)	Yes	No
	Study-13: (Naslavsky et al., 2007, 2009, 2010)	yes	Eclipse-based prototype tool, For model comparison EMFCompare is used.
	Study-14: (Jeron et al., 1999, 2000)	Yes	No
Design Level Approaches (Component-based)	**Study-15: (Muccini et al, 2005, 2006, 2007)**		A plugin inside Charmy environment
	Study-16: (Wu & Offet, 2003)	No	No
Design Level Approaches (Miscellaneous)	**Study17: (Martins et al., 2005)**	Yes	No

Critical Issues–Automation and Tool Support:
Existing model-based regression testing approaches provide very limited tool support. Most of the tools are not mature and also not available online.

E. Discussions

In the above section, we presented a detailed analysis of the existing MBRT approaches. The critical issues corresponding to each analysis criterion are identified and discussed. The critical issues highlight the areas within MBRT which require further attention from the researchers. In general, the analysis shows that there is a need of better support of change identification and impact analysis. Moreover, test automation, standard conformance repair of broken test cases and test result analysis are the areas where further research is required.

MBRT approaches need to be mature by providing support in form of tutorials and more help materials to perform the approaches practically. Further, there is a strong need of comparative empirical evaluations of different categories of MBRT approaches to determine their comparative reduction capabilities and other factors discussed in section VI.C.

In the next section, we discuss an example of our state-based regression testing approach to demonstrate how MBRT can be practically applied. Although, we are not resolving the above mentioned issues in the discussed example but we believe that this example can be useful to understand the basic concepts of MBRT.

VII. CONCLUSION

In this chapter, we discussed the model-based regression testing (MBRT), its core concepts, challenges, the state of the art and the emerging trends. We also demonstrated how MBRT fits in the software development life cycle and we demonstrated the steps involved in MBRT phase. We discussed the challenges, which MBRT has to overcome to reach a more widespread application in the industrial practice. Moreover, we give an overview over the challenges of the emerging model-driven regression testing approaches, which are still in an early stage of development.

The main contribution of the chapter is provision of a comparative analysis and classification of the existing MBRT approaches. First we classified the approaches based on the artefacts they use and whether they are specification-based or design-based MBRT approaches. For this analysis, we developed comprehensive analysis criteria which contain 9 major evaluation criterions consisting of 27 inquiries (research questions). The criteria is based on the challenges in MBRT, we identified earlier in this chapter. We selected 17 studies from the MBRT literature consisting of 31 research papers. We applied the analysis criteria to compare the selected studies in detail. We identified the critical issues in existing MBRT approaches for each criterion after

our analysis. In total, we identified 16 major issues that need to be improved by the MBRT approaches. This analysis can provide the researchers the reasons to do further research in the area of MBRT and to choose the issues they should address in their research. For the practitioners as well, the analysis provides a thorough insight of the strengths and weaknesses for different classes of the approaches.

Furthermore, we presented our own approach for state-based regression testing as a concrete example on how MBRT works in practical scenarios. To conclude we suggest, based on the analysis presented in this chapter, that the lack of tool support, standard conformance especially for test specification, limited focus on the impact analysis and change identification, and the lack of documentation are the major hindrances in the practical application of MBRT and the utilization of its full potential. Research should be conducted in the above mentioned areas to improve the quality and applicability of model-based regression testing approaches.

VIII. REFERENCES

Abramson, D., Sosic, R., & Brisbane, K. R. (1996). A Debugging and Testing Tool for Supporting Software Evolution. *Journal of Automated Software Engineering , 3*, 369-390.

AGEDIS (2002), Last accessed (April 2011), *Automated Generation and Execution of Test Suites for DIstributed Component-based Software,* Available at: http://www.agedis.de/downloads.shtml

Ali, A., Nadeem, A., Iqbal, M. Z.Z., & Usman, M. (2007). Regression Testing Based on UML Design Models. *Pacific Rim International Symposium on Dependable Computing, IEEE , 0*, 85-88.

Anquetil, N., Grammel, B., da, G. L., R., J. A., Khan, S., & Arboleda, H. (2008, June). Traceability for model driven, software product line engineering. *In Proceedingsof ECMDA Traceability Workshop*, 77-86.

Arnold, R., & Bohner, S. (1996). *Software Change Impact Analysis* (1st ed.). Wiley-IEEE Computer Society Press.

ATL, Model Comparison (Last accessed:April 2011)*, Meta-model comparsion and model migration*, Available at: http://www.eclipse.org/gmt/amw/usecases/compare/

Bachmann, F., Bass, L., & Klein, M. (2002). *Illuminating the fundamental contributors to Software Architecture Quality.* Technical Report, Carnegie Mellon Institute, Pittsburgh.

Baker, P., Dai, Z.R., Grabowski, J., Schieferdecker, I. & Williams, (2008) *Model-driven Testing using UML Testing Profile*. ISBN 978-3-540-72562-6, Springer Verelag.

Baldwin,C.Y & Clark. K.B. 1999. *Design Rules: The Power of Modularity Volume 1*. MA, USA, MIT Press, Cambridge.

Beydeda, S., & Gruhn, V. (2001). Integrating White- and Black-Box Techniques for Class-Level Regression Testing. *In Proceedings of 25th Annual International Computer Software and Applications Conference*, Chicago, Illinois, 357.

Binder, R. V. (1999). *Testing Object-Oriented Systems: Models, Patterns, and Tools.* Addison-Wesley Professional.

Bohner, S.A. & Arnold, R.S. (1996), "*Software change impact analysis*", Wiley-IEEE

BPMNEclipse (Last accessed: April 2011), *UML meta-model implementation for eclipse*,Available at: http://www.eclipse.org/modeling/mdt/?project=bpmn2

Briand, L. C., Labiche, Y., & He, S. (2009). Automating regression test selection based on UML designs. *Information and Software Technology. , 51* (1), 16-30

Briand, L. C., Labiche, Y., & Yue, T. (2009). Automated traceability analysis for UML model refinements. *Information and Software Technology , 51*, 512-527.

Briand, L.C., Labiche, L., Buist, K., Soccar, G. (2003). Automating Impact Analysis and Regression Test Selection Based on UML Designs. *Software Quality Engineering Laboratory, Carleton University*, Technical Report, TR SCE-02-04.

Briand, L., Labiche, Y., & Soccar, G. (2002). Automating Impact Analysis and Regression Test Selection Based on UML Designs. *In Proceedings of the International Conference on Software Maintenance*, IEEE Computer Society, 252.

Briand, L., Labiche, Y., O'Sullivan, L. & Sowka, M. (2006). Automated impact analysis of UML models. *Journal of Systems and Software*, 79, 339-352

Chen, Y. (2002). Specification-based Regression Testing Measurement with Risk Analysis. *School of Graduate Studies and Research, Carleton University,* PhD Thesis.

Chen, Y., & Probert, R. (November 2003). A Risk-based Regression Test Selection Strategy. *In Proceeding of the 14th IEEE International Symposium on Software Reliability Engineering*, 305-306.

Chen, Y., Probert, R. L., & Sims, D. P. (2002). Specification-based regression test selection with risk analysis. *In Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*, IBM Press, 1.

Chen, Y., Probert, R. L., & Ural, H. (2007). Regression test suite reduction using extended dependence analysis. *In Proceedings of Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting,* ACM, 62-69.

Chen, Y., Probert, R. L., & Ural, H. (2009). Regression test suite reduction based on SDL models of system requirements. *Journal of Software Maintenance and Evolution: Research and Practice , 21* (6), 379-405.

Chen,Y., Probert,R.L., & Ural, H. (2007). Regression test suite reduction using extended dependence analysis. *In* Fourth *international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting (SOQUA '07)*. ACM, New York, NY, USA, 62-69.

Chittimalli, P. K., & Harrold, M. J. (2008). Regression test selection on system requirements. *In Proceedings of the 1st India software engineering conference*, ACM Computer Society Press, ISBN: 0818673842., 87-96.

Deng, D., Sheu, P.-Y., & Wang, T. (2004). Model-based testing and maintenance. *In Proceedings of the IEEE Sixth International Symposium on Multimedia Software Engineering,* pp. 278-285.

ECL (Last Accessed April, 2011), *Epsilon Comparison Language*, Available at: http://www.eclipse.org/gmt/epsilon/doc/ecl/

EMFCompare, (Last Accessed: April 2011), *Model Comparsion Tool*, Available at: http://www.eclipse.org/emf/compare/

Engstroem, E., Runeson, P., & Skoglund, M. (2010). A systematic review on regression test selection techniques. *Information and Software Technology , 52* (1), 14-30.

Fahad, M., & Nadeem, A. (2008). A Survey of UML Based Regression Testing. In *Intelligent Information Processing, IV*, 200-210.

Farooq, Q. u.-a. (2010). A Model Driven Approach to Test Evolving Business Process based Systems. *In Proceedings of Doctoral Symposium in MODELS 2010*, 19-24.

Farooq, Q. u.-a., Iqbal, M.Z.Z., Malik, Z.I., & Riebisch, M. (2010). A Model-Based Regression Testing Approach for Evolving Software Systems with Flexible Tool Support. *In Proceedings of the 2010 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, 41-49.

Farooq, Q.-u.-a., Iqbal, M. Z.Z., Malik, Z. I., & Nadeem, A. (2007). An approach for selective state machine based regression testing. *In Proceedings of the 3rd international workshop on Advances in model-based testing*, ACM, 44-52.

Farooq, Q.-u.-a. (November 2007). An Approach for Selective State-machine based Regression Testing. *Mohammad Ali Jinnah University, Islamabad, Pakistan,* Available at: http://www.theoinf.tu-ilmenau.de/~qurat/publications.htm. Masters Thesis.

Filho, R.S.S., Budnik, C.J., Hasling, W.M., McKenna, M., & Subramanyan, R.(2010). Supporting Concern-Based Regression Testing and Prioritization in a Model-Driven Environment. *IEEE 34th Annual Computer Software and Applications Conference Workshops (COMPSACW),* 323-328.

Gerth, C., Küster, J.M., Luckey, M., & Engels, G.(2010). Precise detection of conflicting change operations using process model terms. *In* Proceedings of the 13th international conference on Model driven engineering languages and systems: Part II *(MODELS'10),* Springer-Verlag, Berlin, Heidelberg, 93-107.

Gorthi, R. P., Pasala, A., Chanduka, K. K., & Leong, B. (2008). Specification-Based Approach to Select Regression Test Suite to Validate Changed Software. *In Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference*,*0*, IEEE Computer Society, 153-160.

Gotel, O., Finkelstein, A. (1995). Contribution structures Requirements artifacts. *, Proceedings of the Second IEEE International Symposium on Requirements Engineering*, 100- 107.

Rothermel, G, & Harrold, M.J. (1994). A framework for evaluating regression test selection techniques. In *Proceedings of the 16th international conference on Software engineering* (ICSE '94). IEEE Computer Society Press, Los Alamitos, CA, USA, 201-210.

Kabaili, H., Keller, R.K., & Lustman, F. (2001). A Change Impact Model Encompassing Ripple Effect and Regression Testing. *In Proceedings of the Fifth International Workshop on Quantitative Approaches in Object-Oriented Software Engineering,* Budapest, Hungary, 25-33

Kolovos,D.S., Paige,R.F., & Polack, F.A.C. (2006). Model comparison: a foundation for model composition and model transformation testing. *In Proceedings of the 2006 international workshop on Global integrated model* management *(GaMMa '06)*, ACM, New York, NY, USA, 13-20.

Laski, J., & Szermer, W.(1992). Identification of program modifications and its applications in software maintenance, *In Proceedings of Conference on Software Maintenance,* 282-290.

Leung, H.K.N., & White, L. (1989). Insights into regression testing-software testing. *In Proceedings of Conference on Software Maintenance,* 60-69.

Mader,P., Gotel,O., & Philippow,I. (2008). Enabling Automated Traceability Maintenance by Recognizing Development Activities Applied to Models. *In Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE '08)*. IEEE Computer Society, Washington, DC, USA, 49-58

Mäder,P., Gotel, O., & Philippow, I. (2009). Enabling Automated Traceability Maintenance through the Upkeep of Traceability Relations. *In Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA '09)*, Springer-Verlag, Berlin, Heidelberg, 174-189.

Mahdian, A., Andrews, A.A., & Pilskalns, O.J. (2009). Regression testing with UML software designs: A survey. *Journal of Software Maintenance and Evolution.* 21(4), 253-286.

Mansour, N., & Takkoush, H. (2007). UML based regression testing for OO software. *In Proceedings of the 11th IASTED International Conference on Software Engineering and* Applications *(SEA '07)*, Jeffrey E. Smith (Ed.). ACTA Press, Anaheim, CA, USA, 96-101

Mansour, N., Takkoush, H. and Nehme, A. (2011), UML-based regression testing for OO software. Journal of Software Maintenance and Evolution: Research and Practice, 23: 51–68.

MDA (Last accessed: April 2011), *Model Driven Architecture*,Available at: http://www.omg.org/mda/

Muccini, H. (2007). Using Model Differencing for Architecture-level Regression Testing. *In Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO '07),* IEEE Computer Society, Washington, DC, USA, 59-66.

Muccini, H., Dias, M., & Richardson, J.D. (2006). Software architecture-based regression testing. *Journal of Systems and Software*, 79 (10), 1379-1396.

Muccini, H., Dias, M., &Richardson, D.J. (2005). Reasoning about software architecture-based regression testing through a case study. *In Proceedings of the 29th annual international conference on Computer software and applications conference (COMPSAC-W'05),* IEEE Computer Society, Washington, DC, USA, 189-195.

Muccini, H., Dias,M.S., & Richardson,D.J.(2005). Towards software architecture-based regression testing. *SIGSOFT Software Engineering Notes,* 30(4), 1-7.

Naslavsky, L., & Richardson, D.J. (2007). Using traceability to support model-based regression testing. *In Proceedings of the twenty-second IEEE/ACM international conference on Automated software* engineering *(ASE '07).* ACM, New York, NY, USA, 567-570.

Naslavsky, L., Ziv, H., Richardson, D.J. (2009). A model-based regression test selection technique. *IEEE International Conference on Software Maintenance ICSM* , 515-518.

Naslavsky,L., Ziv, H., & Richardson, D.J. (2010). MbSRT2: Model-Based Selective Regression Testing with Traceability. *In Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation (ICST '10)*. IEEE Computer Society, Washington, DC, USA, 89-98.

Orso, A., Apiwattanapong, T., & Harrold, M. J. (2003). Leveraging Field Data for Impact Analysis and Regression Testing. *In Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering (ESEC/FSE'03)*. Helsinki, Finland, 128-137.

Pickin,S., Jard, C., Heuillard,T., Jézéquel, J.M., & Desfray, P. (2001). A UML-integrated Test Description Language for Component Testing. *In Workshop of the pUML-Group held together with the «UML»2001 on Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists*, 208-223.

Pilskalns, O., Uyan, G., Andrews, A. (2006). Regression Testing UML Designs. *22nd IEEE International Conference on Software Maintenance ICSM '06,* 254-264.

Pretschner, A., Philipps, L.J. (2001). Model Based Testing in Evolutionary Software Development. *In Proceedings of the 12th International Workshop on Rapid System Prototyping (RSP '01),* IEEE Computer Society, Washington, DC, USA, 155.

Stallbaum,H., Metzger,A., & Pohl, K. (2008). An automated technique for risk-based test case generation and prioritization. *In Proceedings of the 3rd international workshop on Automation of software test (AST '08)*. ACM, New York, NY, USA, 67-70.

Traon, Y.L., Jeron, T., Jezequel, J.M., Morel, P.(2000). Efficient object-oriented integration and regression testing, *IEEE Transactions on Reliability*, 49(1), 12-25.

U2TP1.0, (July 2005), *UML2 Testing Profile*, Availeble at: http://www.omg.org/spec/UTP/

UMLEclipse (Last accessed: April 2011*), BPMN meta-model implementation for eclipse*,Available at: http://www.eclipse.org/modeling/mdt/?project=uml2

Visser, E., Warmer, J., Deursen, A. V. (2007). Model-driven software evolution: A research agenda. *In Proceedings of International workshop on Model-driven Sofware Evolution held with ECSMR 2007* .

Wu, Y., &Offutt, J. (2003). Maintaining Evolving Component-Based Software with UML. *In Proceedings of the Seventh European Conference on Software Maintenance and Reengineering (CSMR '03),* IEEE Computer Society, Washington, DC, USA, 133.

Xing, Z., & Stroulia, E. (2004). Understanding Class Evolution in Object-Oriented Software. *In Proceedings of the 12th IEEE International Workshop on Program Comprehension (IWPC'04)*, 34-43

Zhang, J. (2004). Supporting software evolution through model-driven program transformation. *In Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and* applications *(OOPSLA '04),* ACM, New York, NY, USA, 310-311.

IX. ADDITIONAL READING SECTION

Recommended Readings on MBRT Techniques before year 2000

Harrold, M. J. (1998). Architecture-Based Regression Testing of Evolving Systems. *Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis (ROSATEA 1998)* , 73-77.

Hsia, P., Li, X., Kung, D. C., Hsu, C.-T., Li, L., Toyoshima, Y.(1997). A technique for the selective revalidation of OO software. *Journal of Software Maintenance , 9*, 217-233.

Kung, D., Gao, J., Hsia, P., Toyoshima, Y., Chen, C., Kim, Y.-S., et al. (1995). Developing an object-oriented software testing and maintenance environment. *Commun. ACM , 38* (10), 75-87.

Kung, D., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., & Chen, C. (1994). Change impact identification in object oriented software maintenance. *In Proceedings of the International Conference on Software Maintenance*, 202-211.

Onoma, A., Tsai, W., Poonawala, M., & Suganuma, H. (1998). Regression testing in an industrial environment. *Communications of ACM , 41* (5), 81-86.

Winter, M. (1998). Managing Object-Oriented Integration and Regression Testing (without becoming drowned). *In Eurostar (Ed.)*.

Studies Excluded from the Analysis of MBRT Approaches

Biswas, S., Mall, R., Satpathy, M., & Sukumaran, S. (2009). A model-based regression test selection approach for embedded applications. *SIGSOFT Software Engineering Notes , 34*, 1-9

Lindvall, M., & Runesson, M. (1998). The visibility of maintenance in object models: an empirical study. *In Proceedings of the International Conference on Software Maintenance*, 54-62.

M., A. S., & Wibowo, B. (2003). Regression Test Selection Based on Version Changes of Components. *In Proceedings of the Tenth Asia-Pacific Software Engineering Conference Software Engineering Conference* , IEEE Computer Society, 78.

Memon, A., Nagarajan, A., & Xie, Q. (2005). Automating regression testing for evolving GUI software: Research Articles. *Journal of Software Maintenance and Evolution,* 17 (1), 27-64.

Memon, A.M., Banerjee, I., Hashmi, & N., Nagarajan, A. (2003). DART: A Framework for Regression Testing "Nightly/daily Builds" of GUI Applications. *In Proceedings of 19th IEEE International Conference on Software Maintenance*, 410-419

Mayrhauser, A. v., & Olender, K. (1993). Efficient testing of software modifications. *In Proceedings of the IEEE International Test Conference on Designing, Testing, and Diagnostics*, 859-864.

Sajeev, A., & Wibowo, B. (2003). UML Modeling for Regression Testing of Component Based Systems. *Electronic Notes in Theoretical Computer Science , 82* (6), 190-198

X. KEY TERMS & DEFINITIONS (SUBHEAD 1 STYLE)

Baseline and Delta Versions: A baseline is a stable and tested version of the system. The test suite which was used to test the baseline is often referred to as baseline test suite. A delta version

of the system is one in which new changes are introduced. It has to be tested using the regression testing approaches.

Regression Testing: Regression testing is a testing activity which is performed after a change is introduced into the system. The aim of the regression testing is to reveal the defects introduced after the changes. The changes introduced in the system are the result of the software evolution. Changes are often identified by comparing the baseline and delta versions of the system. After the change identification, the test cases corresponding to the changes are identified from the baseline test suite to retest the system.

Model-based and Model-driven Testing:
Model-based testing uses analysis and design models of a system as input to identify the changes between different versions of a system. Model-driven testing is a type of Model-based testing which uses MDA principles such as Platform independent models and platform specific models as input and model transformations for test generation. Additionally Model-driven regression testing approaches should also use platform independent test suites and should support the concept of test modelling.

MBRT: Model-based Regression Testing (MBRT) is a type of regression testing that uses analysis and design models of baseline and delta versions of the software system for the change identification. The analysis and design models of the baseline and delta versions are compared to identify the changes between different versions of the systems. The changes are used later to select the regression test cases.

Traceability: Traceability is ability to specify and preserve the relationship between two entities of interest. Traceability is often categorized as implicit and explicit traceability. Implicit traceability is the traceability which exists between two model elements but is not made explicit. Explicit traceability is the traceability which is discovered and stored/persisted for further reuse.

Change Impact Analysis: Change impact analysis is the process to identify the impact of change in one artefact on the other related artefacts. The impact analysis is performed by considering the various dependencies that exists between artefacts in a system.

Abstract test cases and concrete test cases: Abstract test cases are often extracted from the specification of the system. They cannot be executed often due to the fact that they are derived from a representation which is at a higher level of abstraction then the actual system code. They need to be translated to the executable from (concrete test cases) for execution on the system under test.

XI. APPENDIX A

Table 11: The list of selected studies for the analysis

Selected Studies	Corresponding Research Papers
Study-1: (Gorthi et al., 2008)	(Gorthi R. P., Pasala, Chanduka, & Leong, 2008)
Study-2: (Chen et al., (a), 2002,2003)	(Chen, Probert, & Sims, 2002), (Chen Y. , 2002) (Chen & Probert, 2003)
Study-3: (Silva et al., 2010)	(Silva, Budnik, Hasling, McKenna, & Subramanyan, 2010)

Study-4: (Mansour et al., 2007, 2011)	(Mansour & Takkoush, 2007) (Mansour, Takkoush, & Nehme, 2011)
Study-5: (Briand et al., 2002, 2003, 2009)	(Briand, Labiche, & He, 2009) (Briand L. , 2003) (Briand, Labiche, & Soccar, 2002)
Study-6: (Deng et al., 2004)	(Deng, Sheu, & Wang, 2004)
Study-7: (Chen et al., (b), 2007, 2009)	(Chen, Probert, & Ural, 2007), (Chen, Probert, & Ural, 2009)
Study-8: (Korel et al., 2002)	(Korel, Tahat, & Vaysburg, 2002)
Study-9: (Beydeda et al., 2000)	(Beydeda & Gruhn, 2000)
Study-10: (Farooq et al., 2007, 2010)	(Farooq, Q., 2007) (Farooq, Z., Malik, & Nadeem, 2007), (Farooq, Iqbal, Malik, & Riebisch, 2010) Tool Source code: (http://www.theoinf.tu-ilmenau.de/~qurat/projects.htm)
Study-11: (Ali et al., 2007)	(Ali, Nadeem, Iqbal, & Usman, 2007)
Study-12: (Pilskalns et al., 2006)	(Pilskalns, Uyan, & Andrews, 2006)[iii]
Study-13: (Naslavsky et al., 2007, 2009, 2010)	(Naslavsky, Ziv, & Richardson, 2010) (Naslavsky, Ziv, & Richardson, 2009) (Naslavsky & Richardson, 2007)
Study-14: (Jeron et al., 1999, 2000)	(Traon, Jeron, Jezequel, & Morel, 2000), (Jaeron, Jaezaequel, Traon, & Morel, 1999)
Study-15: (Muccini et al., 2005, 2006, 2007)	(Muccini(b), Dias, & Richardson, 2005) (Muccini(a), Dias, & Richardson, 2005) , (Muccini, Dias, & Richardson, 2006) (Muccini, 2007)
Study-16: (Wu & Offet, 2003)	(Wu & Offutt, 2003)
Study17: (Martins et al., 2005)	(Martins & Vieira, 2005)

[i] http://www-01.ibm.com/software/awdtools/rup/#
[ii] http://scrummethodology.com/
[iii] Although this paper consider testing of UML designs itself not the source code but we are not considering any particular representation of SUT. It can be either source code or executable model; hence, we are including this study in our analysis as well