D-MBTDD: An Approach for Reusing Test Artefacts in Evolving Systems

Thaís Harumi Ussami and Eliane Martins
Institute of Computing
University of Campinas
Campinas, S.P., Brazil
Email: thais.ussami@gmail.com, eliane@ic.unicamp.br

Leonardo Montecchi
Dipartimento di Matematica e Informatica
University of Florence
Florence, FI., Italy
Email: lmontecchi@unifi.it

***Abstract*—Agile software development methodologies use an iterative and incremental development in order to handle evolving systems. Consolidated techniques in the field of testing have been applied to these techniques with the main purpose of aiding in the test creation stage. An example is Model-Based Test Driven Development (MBTDD) which joins the concepts of Model-Based Testing (MBT) and Test Driven Development (TDD). However, when iterative and incremental processes are used, problems appear as the consequence of the evolution of the system, such as: how to reuse the test artefacts, and how to select the relevant tests for implementing the new version of the system. In this context, this work proposes a process called D-MBTDD in which the agile development of a system is guided by model-based tests, focusing on helping with the reuse of test artefacts and on the process of identifying tests relevant to development. The information about the modifications between two versions of the test model are used in this approach, which was compared to the Regenerate-All approach, which regenerates test cases along the iterations and does not reuse any of them.**

***Keywords*—Model-Based Test Driven Development, Evolving System, Agile Development, Incremental Tests, Test Reuse, Model-Based Regression Tests**

I. Introduction

Since the Manifesto for Agile Software Development [1], also known as the Agile Manifesto, the adoption of agile approaches has been growing. With these approaches, the focus is on the client's needs, and the development phases (Planning, Analysis, Project, Implementation, Test and Deliver) are performed iterative and incrementally [2]. Agile software development methodologies use an iterative and incremental development in order to handle evolving systems and consequently requirement changes. In each iteration, a subset of the system requirements is analysed, designed, implemented and tested incrementally.

Agile development contributes to improve the client satisfaction with the final version of the system by means of performing continuous tests. Following this idea, some agile approaches were created emphasizing tests. One of these techniques, proposed by Kent Beck, is Test Driven Development (TDD) [3]. TDD proposes that unit tests are iteratively created before the Implementation phase in order to anticipate the validation and verification (V&V) of the system. These tests guide the development of the features during the Implementation phase following a cycle composed of three steps. It starts with a test case creation, which is executed before the implementation of the software, and therefore initially it will fail. In the second step, the code that will make the test be executed successfully is written. And finally the code is refactored in order to make it easier to be maintained [3].

Based on TDD, Dan North proposed Behaviour Driven Development (BDD) [4]. However, differently from TDD which focuses on unit tests, BDD focuses on acceptance tests. BDD focuses on system behaviour and proposes the collaboration between business-oriented and technology-oriented people in order to anticipate the validation of the system.

Tests are often created manually in a non-systematic way, and that is what usually happens with the use of TDD and BDD. However, this practice is subject to human errors because of the repetitive process and the lack of guarantees that the system has a good test coverage, a measure that represents how complete the test suite is. Furthermore, test artefacts created in this way are difficult to be reused, which is of particular importance in a context of test driven development with constant evolution and modification.

A more systematic way to generate test cases consists of deriving them from test models that represent the expected system's behaviour. This idea comes from Model-Based Testing (MBT) [5], in which formal test models that represent the system's behaviour are created and validated in order to automatically generate test cases from them. Following this idea, Model-Based Test Driven Development (MBTDD) [6] joins the concepts of MBT and TDD. Therefore, MBTDD proposes that test models are iteratively created in order to represent enough information for the current iteration, and from these models and using concepts and techniques of MBT, test cases are automatically generated. These tests guide the development of the system by using concepts and techniques of TDD.

In every iteration of MBTDD, the test model evolves to specify new behaviour and consequently a new version of the test model is created. From this new version, test cases are derived in order to guide the development of the new version of the system. During this life cycle, two problems emerge: how to reuse previous generated test cases and how to identify those that will support the development of new features.

In order to minimize these problems, this work proposes D-

MBTDD, a solution combining MBTDD with some concepts from model-based regression testing, to support and facilitate the reuse of test cases. In particular we use concepts inspired in a Software Product Line (SPL) approach, Delta-Oriented Model-Based SPL Regression Testing [7, 8, 9]. It proposes an approach in which delta modelling concepts are used to express the variability between the product variants, and the revalidation of the previous test suite is based on the modifications of the test model. It aims for test artefact reusability and, with the support of deltas that explicitly represent differences among variants, it determines which existing test cases are valid for a product variant and which ones have to be created.

In our solution we adapt the idea of Delta-Oriented Model-Based SPL Regression Testing in order to include it in a context of a model-based test driven development and to support the development of new features by means of test case reusability. Finite State Machines (FSM) are used to represent the system's behaviour, and test cases are derived from them. When the system evolves, the reuse of the test model and the test cases occurs. Therefore, test cases from the previous version are analysed in order to identify which ones are still valid and consequently can be reused. After that, new test cases are created from the new test model version in order to update the new test suite. These new test cases support the development of new features, while the reusable test cases are used as regression tests.

To understand the benefits of the proposed D-MBTDD approach, we compared it with the Regenerate-All approach, in which only the test model is reused, and the test suite is regenerated any time the test model evolves, discarding previously generated test cases.

This paper is organized as follows: Section II presents related work, Section III introduces the D-MBTDD approach, Section IV reports the evaluations and results of the approach, and Section V presents the conclusions.

II. Related Work

The use of model-based testing in an agile context has increased due to the benefits that MBT has in an iterative and incremental environment. MBT supports the automation of test case generation and is adaptable to modifications. Therefore some researches propose the use of MBT in an agile context. Wieczorek et al. [10] propose an approach that uses TDD on the component level, and MBT on the system level. MBT is used to support the generation of integration tests. Hametner et al. [11] propose a method which adapts TDD for the context of the development of an automation system, in which models are used to generate the test cases. However, they only describe the design of the test models. Entin et al. [12] propose to use MBT to support the generation of regression tests. Even though these works deal with MBT in an agile context, test cases are not generated with the aim of guiding the development.

Sadeghi and Hosseinabadi [6] proposed a technique that combines MBT and TDD techniques so that model-based tests guide the development, called Model-Based Test Driven Development (MBTDD). In the MBTDD process, the TDD cycle is extended with MBT steps. Two steps are included before the TDD cycle: the modelling step and the model-based testing step. In the first step the test model is created in order to represent enough information for an iteration, and in the second step the test cases that guide the development are derived from the test model. Along the iterations the test model evolves, so that in each iteration test cases to guide the development are derived. Even though MBTDD deals with a development guided by model-based tests, it does not deal with the reuse of test cases along the iterations. Moreover, it does not support the identification of which test cases guide the development of new features.

In order to reduce these problems, techniques that use Model-Based Regression Testing can be used. The use of model-based regression testing is addressed with different focus in literature, either by focusing on the revalidation of the previous test cases, i.e. their classification [13, 14, 15], or the reuse of test models and test cases [16, 17].

The approach described in this paper is based on the model-based regression testing approach described in [7, 8, 9]. Delta-Oriented Model-Based SPL Regression Testing is an approach for incremental model-based testing of Software Product Line (SPL) which aims to reuse test artefacts. It proposes the use of delta modelling to support the specification of modifications between variants and the incremental creation of test artefacts. In delta modelling, similar products are represented by a designated *core* product and a set of *deltas* that specify changes with respect to the core product. Each product has the following test artefacts: a *test model* which describes the behaviour of the system, a *test suite* containing test cases that cover a set of *test goals* which have to be covered, and a *test plan* containing the test cases that have to be (re-)tested during regression testing. This technique was proposed in the context of SPL, and consequently does not deal with a model-based test driven development and with the identification of which test cases guide the development of new features.

Delta-Oriented Model-Based SPL Regression Testing was adapted to be used in an iterative and incremental development, in which the first version of the system can be seen as a *core* product, and the next version as its *variant*. When the second version evolves into a third version, it is considered as the core product and the new version as its variant. The differences between the versions are represented as *deltas*. To simplify the description when referencing a long chain of versions, all versions after the first can be called variants of the first version, considered the initial core version.

The same collection of test artefacts was used, but only finite state machines were used as test models, and following the idea of Korel et al. [18], the deltas contained only additions and deletions of transitions. That is because an addition or a deletion of a state is associated with an addition or a deletion of at least one transition, respectively.

Differently from related works, D-MBTDD focuses on supporting the reuse of test artefacts along the iterations of a model-based test driven development, and the development of new features by identifying the test cases that will guide

it. D-MBTDD proposes a new feature development cycle in which new test cases guide the development of new features and reusable test cases are used as regression tests. Moreover, D-MBTDD reuses not only the test model, but also the test cases.

III. D-MBTDD

Our approach, D-MBTDD, is inspired by MBTDD [6]. However, in addition to a model-based test driven development, we support the incremental test creation and maintenance, and the identification of test cases that guide the development of new features. D-MBTDD also aims to reduce the effort required for the transformation of abstract test cases into executable test cases by reusing those from previous versions. In order to support this, we use some concepts from Delta-Oriented Model-Based SPL Regression Testing [7, 8, 9], adapting it to the context of an evolving system. Differently from Delta-Oriented Model-Based SPL Regression Testing, D-MBTDD assumes a sequential evolution of the test models in which in each iteration the deltas are applied to the previous version of the test model. We consider the same collection of test artefacts, containing: a *test model* to represent the system's behaviour, a *test suite* which contains test cases, a set of *test goals* that the test suite must cover, and a *test plan* which contains the valid test cases.

In this work, finite state machines (FSM) are used as test models to represent the system's behaviour and the deltas which contain the differences between two versions. Even though during the D-MBTDD process there are steps to create and evolve the test models, it is not in the scope of the approach how they are created. It is assumed that the test models were already created and validated by specialists.

A. Process for the first iteration

In the first iteration the first version of the test artefacts is created, followed by the first version of the system. There are no artefacts to be reused, therefore the following steps are performed:

1. Create and Validate the Test Model: A *test model (TM)* is created and validated by specialists in order to represent the expected system's behaviour of the first iteration.

To illustrate the D-MBTDD process, we have a first version of a FSM in Figure 1.

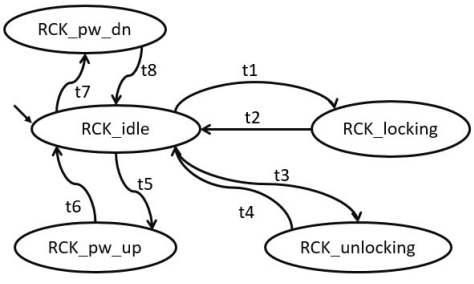

Fig. 1. Core test model example (adapted from [19])

2. Test Goals Definition: To support the generation of model-based tests, it is necessary to define a *coverage criterion* for the test suite. It is defined together with the client, the developers and the testers. Based on this criterion, the *test goals* that the test suite has to cover are derived.

For example, considering a coverage criterion of 100% transitions coverage, the test goals for the FSM illustrated in Figure 1 are: $tg = \{t1, ..., t8\}$.

3. Generate Test Cases: The *test cases* that cover all test goals are generated with the aid of an MBT tool.

For example, using the StateMutest tool [20], 5 test cases were generated in order to cover all test goals for the FSM of Figure 1.

4. Test Suite and Test Plan Definition: The test suite is composed of all test cases, while the test plan is composed of only the valid test cases. In the first iteration, all test cases were generated, therefore the test suite and the plan are equal.

In our example, the test suite and the test plan are composed of 5 test cases.

5. Development Cycle: The *test plan* guides the development, during the *Development Cycle* which follows the TDD concepts.

B. Process for the subsequent iterations

D-MBTDD is used in the context of an evolving system, therefore in the subsequent iterations there will be new requirements and modifications to do, and test cases to be reused. The process follows the steps below, considering:

- $TSreu$ and $TSobs$: the set of reusable and obsolete test cases, respectively, from the previous version.
- $TSreu'$, $TSobs'$ and $TSnew'$: the set of reusable, obsolete and new test cases, respectively, from the new version.

1. New Test Model Creation and Validation: The modifications necessary to obtain a new version of the system are discussed with the client and are represented as a *delta*. This delta is applied to the previous test model version (TM) in order to obtain the *new test model version (TM')*, which is *validated with a specialist*.

For example, we assume that in order to obtain the new version of the system it was necessary to remove transitions $t3$ and $t4$, and add transitions $t9, ..., t15$. This information is represented in the delta illustrated in Figure 2, in which removed transitions are represented with dashed lines and added transitions with double lines. When the delta is applied to the previous FSM illustrated in Figure 1, the new version illustrated in Figure 3 is obtained.

2. Update Test Goals: The test goals for the new test model are updated based on a *coverage criterion*. It can be the same from the previous iteration, but it can also be a different one.

For example, for the new test model of Figure 3 and using the same coverage criterion of the first iteration, the test goals are updated to: $tg = \{t1, t2, t5, ..., t15\}$.

3. Revalidation of the Test Suite: The test suite from the previous version is revalidated in order to identify which test cases are still valid or not. Based on the delta information, the set of reusable test cases used in the previous iteration ($TSreu$), ie. those that were reused in the previous iteration,

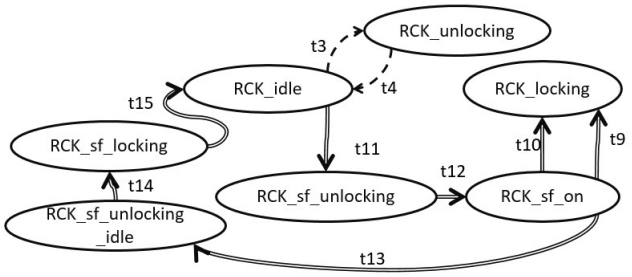

Fig. 2. Delta between model of Fig. 1 and model of Fig. 3. It contains modifications that need to be applied to Fig. 1 in order to obtain Fig. 3 (adapted from [19])

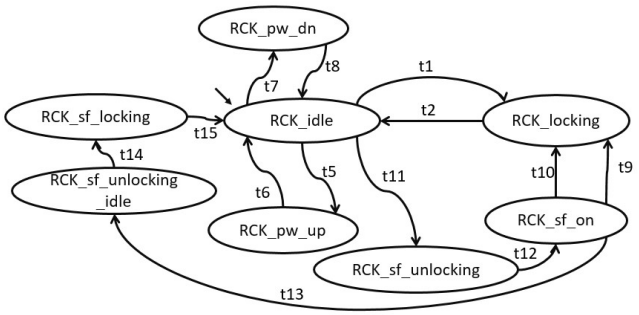

Fig. 3. New test model version example (adapted from [19])

can be classified in the new iteration as *reusable* if they remain valid, or *obsolete* if they do not. A test case may become invalid if it traverses removed transitions. Finally, the set of reusable ($TSreu'$) and obsolete ($TSobs'$) test cases of the new version are created.

In our example, all test cases from the old test model (Figure 1) are present in the set of reusable test cases ($TSreu$) of that version, because it is the first version. $TSreu$ has the 5 test cases and when analysed, 3 remained valid in the new version and 2 became obsolete. Therefore after the revalidation of the test suite, $TSreu'$ had 3 test cases and $TSobs'$ 2 test cases.

4. Update the Test Suite: Using the sets of classified test cases ($TSreu'$ and $TSobs'$) and the set of obsolete test cases from the previous test suite ($TSobs$), the new test suite is updated in order to cover all the test goals. If there are missing test goals to be covered by the test suite, new test cases are created and included in the set of new test cases ($TSnew'$). D-MBTDD does not discard the obsolete test cases in the next test suite because they might be reused in subsequent versions.

In our example, no obsolete test case from the previous version becomes valid for the new version. Therefore the new set of obsolete test cases ($TSobs'$) was updated with the same 2 test cases of the previous set ($TSobs$). Consequently the new set of reusable test cases ($TSreu'$) was updated with the 3 test cases from the previous set ($TSreu$). Because some transitions were added by the delta (Figure 2), there were missing test goals to be covered ($t9, ..., t15$). Therefore, 2 new test cases were created, which make up the set of new test cases ($TSnew'$).

5. New Test Plan: The new test plan (TP') is created with only the valid test cases. Therefore, TP' includes the sets of reusable ($TSreus'$) and new test cases ($TSnew'$) of the new version.

In our example, TP' was created with 5 test cases: 3 from $TSreus'$ and 2 from $TSnew'$.

6. New Feature Development Cycle: The *test plan* guides the development. Because now only the new features have to be developed, a new development cycle is proposed based on the TDD concepts. The set of new test cases guide the development of new features, while the set of reusable test cases are used as regression tests. Therefore, the reusable test cases are applied in order to guarantee confidence in the modified version of the system.

IV. Evaluation and Results

In this Section we discuss the experiments that have been performed and the obtained results. During the evaluation, D-MBTDD was compared to the Regenerate-All approach. In every iteration, differently from D-MBTDD, Regenerate-All updates the test model and generates all test cases from it without reusing any previously generated test cases. It discards all test cases generated in the previous iterations and reuses only the test model. Binder [21, 22] justifies this approach by affirming that updating a test model requires less effort than maintaining a test suite, since the size and complexity of test models grows more slowly than the test suite.

This evaluation tries to identify if Binder's claim is always valid, or if reusing test cases can be a good alternative in some cases. In a development guided by model-based tests there is the necessity to identify those test cases that will support the development of new features. Note that when test cases are automatically generated from models the correspondence between test cases and features might not be obvious. There is also the effort to transform the abstract test cases into executable test cases. Therefore, this evaluation aims to answer the following research questions when D-MBTDD is compared to Regenerate-All: a) Does D-MBTDD require less effort for test case creation? b) Does D-MBTDD require less effort for the identification of which test cases guide the development of new features? c) Does D-MBTDD require a smaller total effort?

The objects used during the experiments were FSMs created in a case study performed by the authors of Delta-Oriented Model-Based SPL Regression Testing. The FSMs describe functionalities of a simplified Body Comfort Sytem (BCS) [19]. The delta modelling concept was used during the case study, therefore core test models, delta models and different variants were created. To adapt it to an evolving system context, each FSM was used as if it were a test model of an iteration. The core represents the test model of the first iteration, while the variants represent the test models for the iterations that follow. From all the core models present in the article, only those which had at least one delta model were selected, resulting in 8 selected core models. Table I summarizes the number of delta models for each selected core test model. The name of each core model was extracted from

the original source [19]. All the models are fully detailed in [19].

TABLE I
CORE TEST MODELS INFORMATION

ID	Model	Deltas
M1	*Manual Power Window*	1
M2	*Automatic Power Window*	1
M3	*Remote Control Key*	3
M4	*Central Locking System*	2
M5	*Human Interface Component*	3
M6	*LED Automatic Power Window*	1
M7	*Alarm System*	2
M8	*Exterior Mirror*	2

The experiments were conducted following a process consisting of two steps: 1) for each core FSM the test cases were created following a coverage criterion of 100% transition coverage; 2) for each version Regenerate-All and D-MBTDD were performed following the same coverage criterion. When Regenerate-All was used all test cases were generated in order to cover all test goals of the FSM. When D-MBTDD was used the analysis steps described in Section III were performed in order to identify the reusable test cases, and only the test cases necessary to cover missing test goals were generated. To perform the test case generation, the StateMutest tool [20] was used during all the experiments.

A. Metrics

The evaluation aims to compare D-MBTDD with Regenerate-All. Therefore, we defined some metrics to be applied to each version of a core test model, in order to compare the effectiveness of the two approaches.

Considering TSO and TSN the size of the old and the new test suite, respectively, and N the number of new test cases generated with D-MBTDD:

- **Generated Test Cases (GenTC):** represents the number of generated test cases per iteration.
  When Regenerate-All is used all test cases are generated from scratch. The number of generated test cases equals to the size of the new test suite, therefore:

$$GenTC(RA) = TSN \tag{1}$$

  When D-MBTDD is used only new test cases necessary to cover missing test goals are generated, therefore:

$$GenTC(DMBTDD) = N \tag{2}$$

- The generated test cases were classified as **Not Modification Traversal Test Cases (NotModTC)** or **Modification Traversal Test Cases (ModTC)**. $NotModTC$ represents the number of test cases that traverse only the unmodified parts, i.e., that do not traverse any modified element of the FSM. $ModTC$ represents the number of test cases that traverse at least one modified element of the state machine. For both approaches, the following relationship between these metrics hold:

$$GenTC = NotModTC + ModTC \tag{3}$$

- **Focus:** measures the effectiveness in generating more modification traversal test cases ($ModTC$) than not modification traversal test cases ($NotModTC$) per iteration. The $ModTC$ test cases support the development of new features, so having a larger portion of them results in more generated test cases used during the development of new features. For both approaches, focus measures the relationship between $ModTC$ and all generated test cases ($GenTC$), therefore:

$$Focus = \frac{ModTC}{GenTC} \tag{4}$$

B. Results

For each test model version, the values of Generated Test Cases ($GenTC$), Not Modification Traversal Test Cases ($NotModTC$), Modification Traversal Test Cases ($ModTC$) and *Focus* are shown in Table II.

Each version of a core test model is identified with an ID in Table II. This ID was defined using the following pattern: the ID of the core test model, presented in Table I, followed by a sequence of IDs (letter D followed by a number) representing the deltas applied to the core model.

TABLE II
VALUE OF METRICS WHEN APPLYING THE TWO APPROACHES TO DIFFERENT VERSIONS

Model ID	Approach	GenTC	NotModTC	ModTC	Focus (%)
M1_D1	Regenerate-All	13	2	11	84.62
M1_D1	D-MBTDD	7	0	7	100
M2_D1	Regenerate-All	17	0	17	100
M2_D1	D-MBTDD	7	0	7	100
M3_D2	Regenerate-All	5	0	5	100
M3_D2	D-MBTDD	3	0	3	100
M3_D2_D1	Regenerate-All	4	0	4	100
M3_D2_D1	D-MBTDD	2	0	2	100
M3_D2_D1_D3	Regenerate-All	8	0	8	100
M3_D2_D1_D3	D-MBTDD	6	1	5	83.33
M4_D1	Regenerate-All	5	0	5	100
M4_D1	D-MBTDD	1	0	1	100
M4_D1_D2	Regenerate-All	6	1	5	83.33
M4_D1_D2	D-MBTDD	2	0	2	100
M5_D1	Regenerate-All	7	0	7	100
M5_D1	D-MBTDD	2	0	2	100
M5_D1_D2	Regenerate-All	7	0	7	100
M5_D1_D2	D-MBTDD	2	0	2	100
M5_D1_D2_D3	Regenerate-All	10	0	10	100
M5_D1_D2_D3	D-MBTDD	2	0	2	100
M6_D1	Regenerate-All	12	3	9	75
M6_D1	D-MBTDD	7	1	6	85.71
M7_D1	Regenerate-All	11	2	9	81.82
M7_D1	D-MBTDD	3	0	3	100
M7_D1_D2	Regenerate-All	11	6	5	45.45
M7_D1_D2	D-MBTDD	10	3	7	70
M8_D1	Regenerate-All	10	2	8	80
M8_D1	D-MBTDD	8	1	7	87.50
M8_D1_D2	Regenerate-All	10	3	7	70
M8_D1_D2	D-MBTDD	9	2	7	77.78

To help with the discussion, the metrics have been plotted using bar charts, and are displayed in Figures 4. For each version of a core FSM two approaches were simulated, and therefore we have two bars for each experiment: the left bar contains the results obtained with Regenerate-All, and the right those obtained with D-MBTDD. The total size of each bar represents the value of $GenTC$. Each bar was split into two parts: a blue bar filled with squares, and an orange bar filled

with circles. The blue bar represents the value of $ModTC$, and the orange bar represents the value of $NotModTC$.

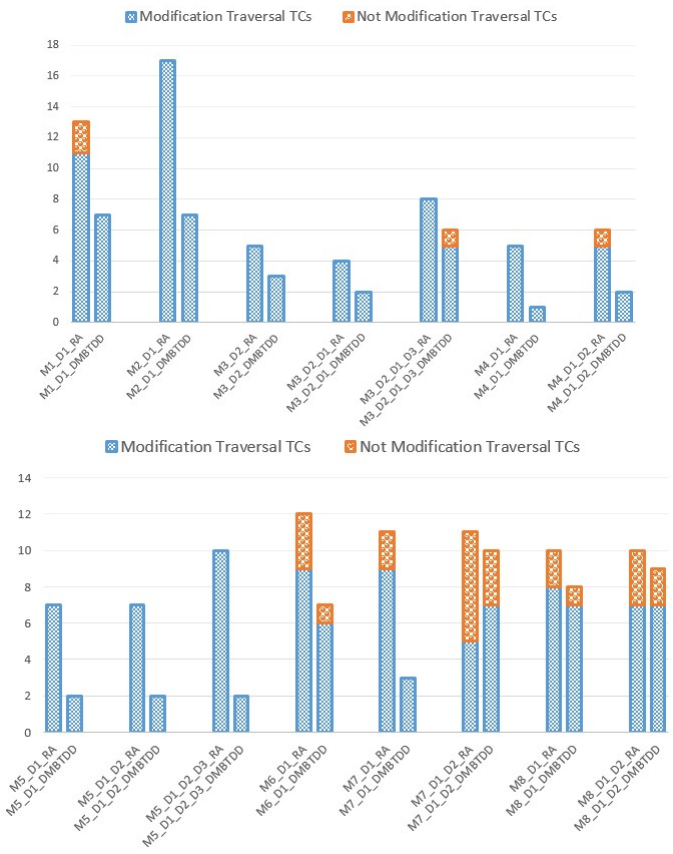

Fig. 4. Results of the experiments. The bar graphs show the number of generated test cases for different test models.

When analysing the values of Table II, and the Figures 4 we noticed that:

1) In all experiments, the value of $GenTC$ when D-MBTDD was used was equal to or smaller than the value when Regenerate-All was used. Therefore, D-MBTDD generated less test cases per execution than the Regenerate-All approach.
2) In 93.33% of the experiments, the value of $ModTC$ when D-MBTDD was used was equal to or smaller than the value when Regenerate-All was used. The only exception was the `M7_D1_D2` example. Therefore, in the majority of the examples, D-MBTDD generated an equal or smaller value of modification traversal test cases than Regenerate-All.
3) In 93.33% of the experiments, the value of $NotModTC$ when D-MBTDD was used was equal to or smaller than the value when Regenerate-All was used. The only exception was the `M3_D2_D1_D3` example. Therefore, in the majority of the examples, D-MBTDD generated an equal or smaller value of not modification traversal test cases than Regenerate-All.
4) In 93.33% of the experiments, the value of *Focus* when D-MBTDD was used was equal to or greater than the value when Regenerate-All was used. The only exception was the `M3_D2_D1_D3` example. Therefore, in the majority of the examples, D-MBTDD had a higher efficiency in generating test cases that guide the development of new features than Regenerate-All.

C. Cost comparison between D-MBTDD and Regenerate-All

Based on the results and on the experiments we can discuss the answers for the research questions defined in the beginning of this Section. For each question we summarized the cost of each step involved when using Regenerate-All and when using D-MBTDD considering:

- $c(ge)$: cost to *generate* the test cases;
- $c(re)$: cost to *revalidate* the old test suite;
- $c(id)$: cost to *identify* the test cases that guide the development of new features;
- $c(tr)$: cost to *transform* the abstract test cases into executable ones

Total Cost of generating test cases (E_Cr)

In Regenerate-All, after the test model evolves the test cases are simply generated from it, therefore:

$$E\_Cr(RA) = GenTC*c(ge)$$

Considering equation 1:

$$E\_Cr(RA) = TSN*c(ge) \quad (5)$$

In D-MBTDD, before generating the test cases it is necessary to revalidate the test cases from the old test suite, therefore:

$$E\_Cr(DMBTDD) = TSO*c(re) + GenTC*c(ge)$$

Considering equation 2:

$$E\_Cr(DMBTDD) = TSO*c(re) + N*c(ge) \quad (6)$$

When comparing equation 5 and 6, we can see that D-MBTDD has an additional cost to revalidate the previous test suite. Therefore, in general D-MBTDD does not require less effort for test case creation per iteration when compared to Regenerate-All. However, this is not true if $(TSO + N) < TSN$ and $c(re) < c(ge)$.

Cost of identifying test cases that guide the development of new features (E_Id)

The test cases that guide the development of new features are those classified as modification traversal. In Regenerate-All, after the generation of all test cases it is necessary to identify those that traverse modified elements, therefore:

$$E\_Id(RA) = GenTC*c(id)$$

Considering equation 1:

$$E\_Id(RA) = TSN*c(id) \quad (7)$$

In D-MBTDD the generation of test cases is focused on modified elements. Consequently the modification traversal

test cases are already identified in the set of new test cases, therefore no effort is required for this task:

$$E\_Id(DMBTDD) = 0 \quad (8)$$

When comparing equation 7 and 8 we can conclude that D-MBTDD requires less effort for the identification of which test cases guide the development of new features when compared to Regenerate-All.

Total Effort (E_Total)

The total effort to use each approach is composed of the effort to generate the test cases, to identify the modification traversal test cases, and to transform abstract test cases into executable test cases.

Considering equations 5 and 7, and that all generated test cases have to be transformed into executable test cases in Regenerate-All, the total effort is:

$$E\_Total(RA) = TSN*c(ge) + TSN*c(id) + GenTC*c(tr)$$

Considering equation 1:

$$E\_Total(RA) = TSN*c(ge) + TSN*c(id) + TSN*c(tr) \quad (9)$$

Considering equations 6 and 8, and that only the new test cases have to be transformed into executable test cases in D-MBTDD, the total effort is:

$$E\_Total(DMBTDD) = TSO*c(re) + N*c(ge) + N*c(tr) \quad (10)$$

When analysing equations 5 and 6, we noticed that the total effort for using Regenerate-All is impacted by the value of TSN, while D-MBTDD is impacted by the values of TSO and N. According to the evolution of the test model, in order to cover a specific criterion, the number of generated test cases when Regenerate-All is used ($Gen(RA)$) tends to increase, and consequently the size of the new test suite (TSN). With D-MBTDD, in order to cover the same criterion, there is a tendency to generate less test cases because part of the test goals will be already covered by the reusable test cases. Therefore, the value of the generated test cases ($Gen(DMBTDD)$), and consequently the value of N, tends to be smaller than $Gen(RA) = TSN$.

If we consider:

- $TSO \approx TSN$, i.e., the size of the old and new test suites are approximately the same, and
- $N < TSN$, the number of generated test cases with D-MBTDD is smaller than with Regenerate-All as discussed above.

Then two conditions improve the effectiveness of D-MBTDD over Regenerate-All:

- $c(tr) > c(re)$, i.e. the cost of transforming abstract test cases into executable test cases is greater than the cost of revalidating the old test suite; and
- $c(id) > c(re)$, i.e. the cost of identifying the test cases that guide the development of new features is greater than the cost of revalidating the old test suite.

D. Threats to the Validity

When performing an experiment, there are some threats to the validity to be concerned about. In this evaluation the models and the number of deltas used during the experiments represent threats to external validity since their complexity, their size and their domain can affect the results. The results are also impacted by the MBT tool used during the experiments. The metrics measured during the controlled experiments may not be able to completely show the trade-offs between the two alternatives, representing threats to the construction of the experiments.

V. Conclusions and Future Work

A model-based test driven development has some challenges along the iterations, such as the reuse of test artefacts and the identification of those that should guide the development of new features. This work proposed D-MBTDD, an approach which tries to minimize these problems by joining MBTDD and Delta-Oriented Model-Based SPL Regression Testing. This results in a process which contains steps of Delta-Oriented Model-Based SPL Regression Testing to support test case creation, maintenance and reusability, and steps of MBTDD to support the development guided by model-based testing. D-MBTDD reuses not only the test model, but also the test cases. During new increments, the test cases are generated focusing on the modified elements of the test model. With that, only these test cases have to be transformed into executable test cases per iteration in order to support the development of new features, allowing the effort needed for this transformation to be reduced.

During this work D-MBTDD was evaluated against the Regenerate-All approach, in which only the test model is reused along the iterations, and the test suite is regenerated any time the test model evolves. D-MBTDD performed better according to the set of defined metrics. In 93.33% of the experiments, D-MBTDD had an equal or greater value of effectiveness (*Focus*) in generating modification traversal test cases which are used to guide the development of new features. Even though D-MBTDD requires a revalidation of the previous test suite, it requires less effort to support the development of new features. This is because D-MBTDD generates test cases based on the modifications and consequently focused on the new elements.

This work defined some formulas to quantify the total effort to use D-MBTDD and Regenerate-All. While D-MBTDD has a cost to revalidate the previous test suite before generating test cases, Regenerate-All has the cost to identify the modification traversal test cases after the test case generation. As a future work we plan to develop experimental case studies in order to quantify the costs involved in each step of the approaches. Moreover, we plan to to perform more extensive experimental analyses, and properly apply statistical techniques like hypothesis testing to verify the obtained results. The obtained experimental data would allow us to quantify and analyse the total effort to use the approaches, and better understand in which occasions one can be recommended over the other.

ACKNOWLEDGMENT

This work has been partially supported by the project DEVASSES - DEsign, Verification and VAlidation of large-scale, dynamic Service SystEmS, funded by the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no PIRSES-GA-2013-612569. The authors also would like to thank the grant 151647/2013-5, CNPq, for the financial support.

REFERENCES

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Granning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001, http://www.agilemanifesto.org Acccessed March 21, 2016.

[2] I. Sommerville, *Software Engineering (original title in Portuguese: Engenharia de software)*, 8th ed. Addison Wesley, 2007, ch. Agile Software Development.

[3] K. Beck, *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] D. North, "Introducing BDD," *Better Software Magazine*, Mar. 2006. [Online]. Available: http://dannorth.net/introducing-bdd/

[5] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Softw. Test. Verif. Reliab.*, vol. 22, no. 5, pp. 297–312, Aug. 2012.

[6] A. Sadeghi and S.-H. Mirian-Hosseinabadi, "Mbtdd: Model based test driven development," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 08, pp. 1085–1102, 2012.

[7] S. Lity, M. Lochau, I. Schaefer, and U. Goltz, "Delta-oriented model-based spl regression testing," in *Proceedings of the Third International Workshop on Product LinE Approaches in Software Engineering*, ser. PLEASE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 53–56.

[8] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity, "Incremental model-based testing of delta-oriented software product lines," in *Proceedings of the 6th International Conference on Tests and Proofs*, ser. TAP'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 67–82.

[9] M. Lochau, S. Lity, R. Lachmann, I. Schaefer, and U. Goltz, "Delta-oriented model-based integration testing of large-scale systems," *J. Syst. Softw.*, vol. 91, pp. 63–84, May 2014.

[10] S. Wieczorek, A. Stefanescu, M. Fritzsche, and J. Schnitter, "Enhancing test driven development with model based testing and performance analysis," in *Practice and Research Techniques, 2008. TAIC PART '08. Testing: Academic Industrial Conference*, Aug 2008, pp. 82–86.

[11] R. Hametner, D. Winkler, T. Ostreicher, S. Biffl, and A. Zoitl, "The adaptation of test-driven software processes to industrial automation engineering," in *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, July 2010, pp. 921–927.

[12] V. Entin, M. Winder, B. Zhang, and S. Christmann, "Introducing model-based testing in an industrial scrum project," in *Automation of Software Test (AST), 2012 7th International Workshop on*, June 2012, pp. 43–49.

[13] Q.-u.-a. Farooq, M. Z. Z. Iqbal, Z. I. Malik, and A. Nadeem, "An approach for selective state machine based regression testing," in *Proceedings of the 3rd International Workshop on Advances in Model-based Testing*, ser. A-MOST '07. New York, NY, USA: ACM, 2007, pp. 44–52.

[14] L. Naslavsky, H. Ziv, and D. J. Richardson, "Mbsrt2: Model-based selective regression testing with traceability," in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*, April 2010, pp. 89–98.

[15] J. O. Blech, D. Mou, and D. Ratiu, "Reusing test-cases on different levels of abstraction in a model based development tool," in *Proceedings 7th Workshop on Model-Based Testing, MBT 2012, Tallinn, Estonia, 25 March 2012.*, 2012, pp. 13–27.

[16] S. Weileder, D. Sokenou, and H. Schlingloff, "Reusing state machines for automatic test generation in product lines," in *MoTiP 08: Model-Based Testing in Practice*, Thomas Bauer, Hajo Eichler, Axel Rennoch, Ed. Fraunhofer IRB Verlag, 2008.

[17] D. Dranidis, A. Metzger, and D. Kourtesis, *Towards a Service-Based Internet: Third European Conference, ServiceWave 2010, Ghent, Belgium, December 13-15, 2010. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. Enabling Proactive Adaptation through Just-in-Time Testing of Conversational Services, pp. 63–75.

[18] B. Korel, L. H. Tahat, and B. Vaysburg, "Model based regression test reduction using dependence analysis," in *Software Maintenance, 2002. Proceedings. International Conference on*, 2002, pp. 214–223.

[19] S. Lity, R. Lachmann, M. Lochau, and I. Schaefer, "Delta-oriented software product line test models - the body comfort system case study," TU Braunschweig, Tech. Rep. 2012-07, 2013.

[20] W. F. F. Cardoso, "Statemutest: an extended state model based test support tool (original title in portuguese: Statemutest: uma ferramenta de apoio ao teste baseado em modelos de estado estendidos)," Master's thesis, Unicamp, SP, 2015.

[21] R. V. Binder, "How to ice the testing backblog," 2013, http://robertvbinder.com/how-to-ice-the-testing-backblob/ Accessed March 21, 2016.

[22] R. Binder, "Model-based testing: Taking bdd/atdd to the next level," 2014, slides from presentation at the Chicago Quality Assurance Association, February 25, 2014, http://pt.slideshare.net/robertvbinder/taking-bddtothenext level Acccessed March 21, 2016.