IET Software

Review Article

IET Journals
The Institution of Engineering and Technology

MBT in agile/lightweight processes: a process-centred review

ISSN 1751-8806
Received on 8th May 2018
Revised 22nd February 2019
Accepted on 2nd April 2019
doi: 10.1049/iet-sen.2018.5164
www.ietdl.org

Masoumeh Taromirad[1], Raman Ramsin[1] ✉

[1]*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

✉ *E-mail: ramsin@sharif.edu*

Abstract: This study presents a process-driven view on the use of model-based testing (MBT) in agile/lightweight processes. It argues that process-related aspects of MBT and agile processes should be explicitly considered in any practical MBT approach intended for use in agile development (AD). It demonstrates that an effective MBT approach for lightweight processes has to specify how MBT activities are integrated into a development process, how and when MBT artefacts are generated in relation to other development artefacts, and who would carry out MBT activities. Accordingly, a set of integration criteria is introduced for complete incorporation of MBT into agile/lightweight processes. The integration criteria demonstrate the specific characteristics of an appropriate MBT process for AD processes, and help identify the benefits and shortcomings of existing methods on the use of MBT in such processes. Evaluation of existing works based on the proposed integration criteria shows that they have all focused on *minimal modelling*, and only one method has considered the 'evolution' of test models and the 'reuse' of test cases, whereas 'evolution' and 'reuse' are essential characteristics of agile processes, which have to be addressed in any MBT approach intended for such processes.

1 Introduction

In modern software development, testing plays an invaluable role as a quality assurance measure. In particular, agile/lightweight methodologies recommend automated testing as the main method for quality assurance [1]. By bringing testing into the main development cycle and focusing on automated testing, these methods aim to achieve low defect rates while remaining faithful to the principles of agility. However, they are plagued with several deficiencies in this regard including complex and difficult-to-maintain test scripts [2].

A promising solution to these issues lies in model-based testing (MBT) processes, which provide a structured approach to testing based on high-level behavioural models [3]. The benefit of MBT lies primarily in automated test case generation and automated analysis of the test results; it has, therefore, received significant attention in the testing of complex software systems. MBT also supports requirements validation early in the development process; defects in requirements such as ambiguity and incompleteness can be detected during the development of the test models, which are usually produced during the earlier phases of the development process. However, many MBT methods are not directly suitable for application to agile/lightweight methodologies, as the practises prescribed by MBT are not particularly compatible with those prescribed by agile methods: typically, agile/lightweight methodologies tend to simplify and speed up the development process by discouraging the production, and subsequent updating, of models [2, 4].

As mentioned at the Dagstuhl Seminar in 2004, for MBT to be successful, it must be integrated into the development process, and hence, it has to support process-specific characteristics that are specifically related to testing. In our view, an effective MBT approach for agile/lightweight processes needs to explicitly consider and address the characteristics of such processes. Many studies have argued that a major obstacle in the effective application of MBT to agile processes is that the models required for MBT are typically different from those usually developed during general software analysis and design or need additional details that are not typically included in ordinary models (e.g. temporal-logic descriptions for properties [5]). Accordingly, the use of MBT in agile/lightweight processes has mainly been studied by focusing on the models that could be used for MBT in agile/lightweight processes (e.g. [2, 4, 6–8]).

We aim to demonstrate that agile/lightweight processes involve certain other characteristics and features that should be considered in an effective MBT approach for agile methods, as addressed by a small number of studies (e.g. [7, 8]). For example, an inherent characteristic of agile processes is the evolution of development artefacts including models. Therefore, the test models need to be kept up-to-date, and hence, an MBT approach has to explicitly provide mechanisms to effectively support model evolution. Moreover, the (test) models become larger and more complex over time; hence, the number of test cases grows continuously and rapidly, which if not properly managed, will increase the testing effort required. This makes it impossible to get frequent and quick feedback on the whole system, which is a mandatory requirement in agile/lightweight processes. These characteristics are addressed when process-related issues and challenges are considered in the application of MBT in agile/lightweight processes. Whereas if the focus is limited to the issues related to artefacts, all that is observed is the misalignment or difference in abstraction level between the models required by MBT techniques and those used in agile methods.

This paper presents a process-driven view on the use of MBT in agile/lightweight processes, highlighting the process-related issues. It introduces a set of criteria to be considered in a complete incorporation of MBT into agile/lightweight processes, to yield an appropriate MBT approach for agile/lightweight processes. In the process-driven approach to the application of MBT in a development process, we mainly focus on (i) how MBT activities are integrated into a development process, (ii) how MBT artefacts (e.g. test models and test cases) are generated, in relation to other development artefacts, and (iii) who would carry out MBT activities. Note that we are not interested in classifying MBT techniques based on, for example, their underlying (formal) semantics and test generation algorithms (i.e. based on MBT taxonomies such as that proposed in [9]). In fact, such classifications and studies are helpful if a concrete MBT technique needs to be selected (e.g. [10, 11]). We look at such properties, only when required, in relation to other development artefacts.

Accordingly, we introduce a set of *integration criteria* for complete and practical incorporation of MBT into agile/lightweight processes. The criteria are defined based on (i) the general MBT

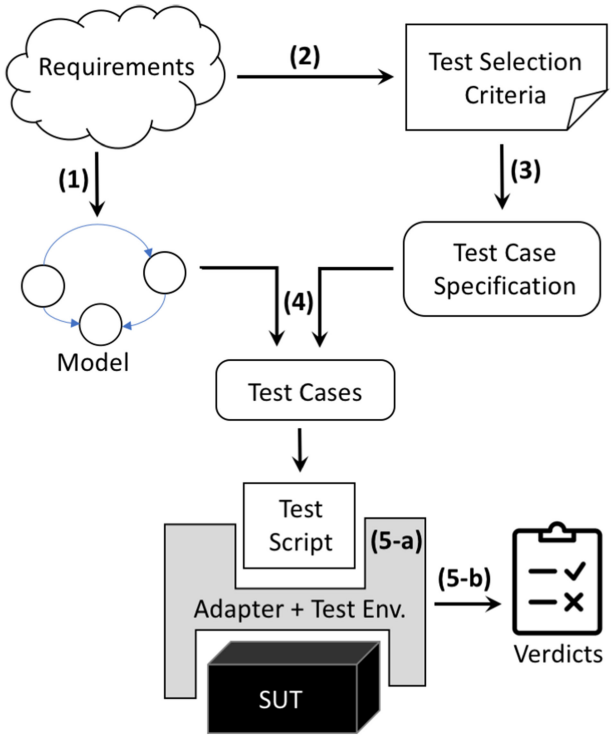

Fig. 1 *Generic process of MBT, proposed by Utting et al. [12]*

process and its requirements and (ii) the specific challenges of the target development process (which is an agile/lightweight methodology). In this paper, we focus on those characteristics/requirements that are related to MBT; i.e. characteristics/requirements that affect or are affected by the integration.

The integration criteria define the specific characteristics of an appropriate MBT process for agile processes, thus allowing us to engineer an appropriate MBT process for a target development process. Additionally, they enable us to assess the extent to which a testing approach complies with or fulfils the requirements of a target development process. Moreover, the set of criteria helps identify the benefits and shortcomings of the methods already used for MBT in agile/lightweight processes.

The main contributions of this paper are as follows:

- A comprehensive set of criteria for incorporation of MBT (activities and artefacts) into agile/lightweight processes.
- An evaluation of existing studies on the use of MBT in agile/lightweight processes, identifying the shortcomings of existing approaches.

The rest of this paper is structured as follows: Section 2 briefly introduces MBT processes including their activities and artefacts. In Sections 3 and 4, considering an ideal and complete incorporation of MBT into agile/lightweight processes, a comprehensive set of integration criteria is proposed. Section 5 introduces the existing works that consider process-related issues in the application of MBT in agile processes. Then, in Sections 6 and 7, these works are evaluated and discussed with respect to the proposed integration criteria. Finally, Section 8 concludes this paper and outlines the future work.

2 MBT: fundamentals

MBT is a variant of testing that relies on explicit behavioural models that encode the intended behaviour of a system under test (SUT) and/or the behaviour of its environment [12]. Test cases are generated from one of these models or a combination thereof and are then executed on the SUT. The use of explicit models is motivated by the observation that, traditionally, test derivation tends to be unstructured, irreproducible, undocumented, and dependent on the ingenuity of the individual performing it. The idea is that the artefacts that explicitly encode the intended SUT, and possibly the environment's behaviour, can help mitigate these problems. However, some aspects remain unclear such as the models that can be used by MBT techniques, and the abstraction level of these models [13].

2.1 MBT process and activities

MBT encompasses various processes and techniques for the automatic derivation of abstract test cases from abstract models, the generation of concrete tests from abstract tests, and the manual or automated execution of the resulting concrete test cases.

Utting *et al.* [12] define a generic process for MBT, as depicted in Fig. 1, which involves the following major activities:

1. building the (abstract) test model,
2. defining test selection criteria,
3. transforming test selection criteria into operational test case specifications,
4. generating tests, and
5. (a) developing and setting up the adaptor component and (b) executing the tests on the SUT.

2.2 MBT artefacts

The general MBT process involves a specific set of artefacts required for automatic test generation and execution. We will have a brief look at the main artefacts.

Test model: A model of the SUT is built from informal requirements or existing specification documents, which is often called a test model. There might be a set of models of the SUT. Test models are used as the basis for test generation, and thus they must be sufficiently precise to serve as a basis for the generation of 'meaningful' test cases. The tests generated from the models should be complete enough in terms of actions, input parameters and expected results, to provide real added value. If not, a part of the test design job has to be performed manually.

It is usually either to develop a test-specific model directly from the informal requirements or to reuse just a few aspects of the development models as the basis for the test model, which is then validated against the informal requirements. Nevertheless, it is important to have a certain degree of independence between the models used for test generation and the development model, so that errors in the development model are not propagated to the generated tests [14].

Test case specifications: Test case specifications formalise the notion of test selection criteria and render them operational so that an automatic test case generator can generate/derive a test suite from the test models, satisfying the test case selection criteria. Test selection criteria guide the automatic test generation to produce a 'good' test suite – one that fulfils the test policy defined for the SUT or a testing experiment. Test selection criteria can address different aspects of a system. For example, they may relate to a given functionality of the system (requirements based test selection criteria), to the structure of the test models (state coverage and transition coverage), to data coverage heuristics (pairwise, boundary value), to properties of the environment, or to a well-defined set of faults. Test selection criteria are transformed into test case specifications.

Test suite and test case: A test suite is a finite set of test cases. A test case is a finite structure of the input and expected output; it can be a pair of input and output in the case of deterministic systems, a sequence of input and output in the case of deterministic reactive systems, and a tree or a graph in the case of non-deterministic reactive systems.

Adaptor: As the test model and SUT reside at different levels of abstractions, these different levels must be bridged. Execution of a test case starts by concretising the test inputs and sending that concrete data to the SUT. The resulting concrete output of the SUT must be captured and abstracted to obtain the high-level expected result, which is then compared against the expected (abstract) result. The component that performs the concretisation of test inputs and abstraction of test outputs is called the adaptor, as it

Table 1 Evaluation values for 'building the model'

Case	Description and values
explicit	activity is considered *explicitly*
	specify **when** the activity is carried out (if possible)
	specify **by whom** the activity is carried out (if possible)
	specify the required **inputs** (if existent)
implicit	activity is *implied*
	specify **how** it is implied (if applicable)
no	activity is neither explicitly considered nor implied

Italic words show the general meaning of the corresponding case for the criterion. Bold words define the aspects that are considered for a criterion.

adapts the abstract test data to the concrete SUT interface. The adaptor is a concept and not necessarily a separate software component – it may be integrated within the test scripts.

3 MBT in agile development (AD) processes

The most essential feature for considering a testing process as an MBT process is to support the generation and execution of test cases based on some (preferably formal) abstract test model. In this context, the very first requirement for applying an MBT approach is to have test models that are appropriate for the desired level(s) of testing and to then generate the required test cases using these test models. This MBT approach is called *artefact-driven*.

The *artefact-driven* approach has been commonly considered in the context of using MBT in development processes. Most previous studies have focused on the different types of test models that can be used for MBT in a development process or in a specific context or domain such as software product lines or cyber-physical systems. Similarly, in the context of agile/lightweight processes, previous studies have largely focused on test models, and have hence argued that the main challenge in the use of MBT in such processes is the misalignment or difference in abstraction level between the artefacts required by MBT and those used in agile methods.

Apart from the overall approach, a practical incorporation of MBT into a development process has to specify how MBT activities are integrated and carried out, and how test artefacts (e.g. test models and test cases) relate to other development artefacts; however, very few studies have addressed this (e.g. [8]).

We propose an alternative, *process-driven* approach to the use of MBT in AD processes, by highlighting the process-related issues and characteristics. In this approach, an MBT technique is studied in the context of a complete development process, considering all the testing and related development activities and artefacts, as well as the related process-specific characteristics and requirements of agile/lightweight methods. We are mainly interested in the following aspects, considering general MBT activities and artefacts:

- how MBT activities/tasks are integrated into an AD process;
- how MBT artefacts, e.g. test models and test cases, are generated and used in relation to other development artefacts;
- who would carry out each MBT activity; and
- how each (related) specific requirement of agile methods is fulfilled.

Accordingly, a set of criteria for complete incorporation of an MBT approach into agile/lightweight processes is identified and introduced. Basically, an integration is considered from two main perspectives: (i) the MBT process and (ii) the type or context of the target development process. Each perspective focuses on specific aspects and the needs corresponding to that perspective.

Additionally, the application of MBT in a development process is affected by the type or level of testing for which MBT is intended. An MBT technique for unit testing (UT) requires different types of artefacts from an MBT technique for system testing (ST). We have also come to realise that an MBT process proposed in the context of agile/lightweight processes can aim for different levels of detail, from abstract to very detailed; for example, one may intentionally provide a high-level, and hence more flexible, MBT process. Accordingly, the intended level of detail has been defined as an integration criterion that can be high/abstract, medium, or low/detailed.

4 Integration criteria

In this section, we first define the process-related characteristics of MBT that should be considered in complete and fully detailed integration. Then, considering agile processes, the agility characteristics that have to be fulfilled in a practical MBT approach are introduced and described.

4.1 MBT process characteristics

As explained in Section 2.1, an MBT process consists of a number of major activities along with their associated artefacts. In a practical application of MBT in a development process, it is also important to specify who does what; i.e. the roles involved. Note that there are classifications and taxonomies for MBT such as [12, 15, 16] that consider the characteristics of a standalone MBT technique, regardless of the development process in which it is used. Such taxonomies would help practitioners or researchers in identifying specific MBT techniques, and would hence improve the adoption of MBT technologies. However, they do not consider the process-related issues and requirements, which are the focus of our study. In the context of our study, such taxonomies can provide guidelines for identifying the different aspects in the process-driven view on the application of MBT in agile processes; we have also used them, in a few cases, to identify the detailed evaluation values (e.g. for determining the test execution mode).

Through a number of iterations and refinements, a number of criteria have been defined along with their possible values. These criteria, which assess an integration with respect to the characteristics of MBT processes, are described throughout the rest of this section.

4.1.1 Criteria related to MBT activities: Each MBT activity should be covered, particularly with respect to other development activities. However, some of the activities are more important, and are, therefore, considered explicitly and in more detail; others, which are less essential for integration, are typically implied by the proposed technique or the testing tool. In this section, we will discuss each MBT activity in the context of detailed integration into an agile/lightweight process, and introduce its relevant criterion.

Building the model: This is the most important activity in an integration, as pointed out in the literature. It is typically signified by its main associated (output) artefact: test models. Depending on the intended levels (types) of testing, a complete integration should specify and discuss *when* and *by whom* the test models are built. A practical integration has to specify the *input* for this activity as well. Test models may be (i) an already available model such as design models (this would be an instance of complete reuse), (ii) automatically or manually generated based on other models (partial reuse), and (iii) completely constructed from scratch (no reuse). In some cases, this activity is implied by assuming that the test models are already available, which is sometimes a valid assumption (e.g. when the test models are provided by a third party); otherwise, it would result in an incomplete integration. For the evaluation, we first look into whether this activity is explicitly considered and mentioned in an integration or not. Then, in case it is considered, we examine to what extent it is discussed by asking the following questions: (i) Is it stated when it is performed?; (ii) Is it stated by whom it is performed?; and (iii) Does it require any specific input? In cases where this activity is implied, we may find out and mention 'how it is implied'. Table 1 shows the evaluation values.

Defining test selection criteria and test case specifications: These two activities are usually not mentioned explicitly, and it is typically assumed that there is always a set of test selection criteria (and consequently, test case specifications) for generating the test cases. It is usually implied by the specific MBT technique or the

Table 2 Evaluation values for 'defining test selection criteria and test case specifications'

Case	Description and values
explicit	activity is considered *explicitly*
	specify the test selection **criteria** (if available)
	specify **by whom** the activity is carried out (if possible)
implicit	activity is *implied*
	specify **how** it is implied (if applicable)
no	activity is neither considered explicitly nor implied

Italic words show the general meaning of the corresponding case for the criterion. Bold words define the aspects that are considered for a criterion.

Table 3 Evaluation values for 'generating tests'

Case	Description and values
explicit	the activity is considered *explicitly*
	specify **when** the activity is carried out (if possible)
	specify **by whom** the activity is carried out (if possible)
	specify the test generation **algorithm** (if mentioned)
	specify the specific testing **tool** (if mentioned)
implicit	the activity is *implied*
	specify **how** it is implied (if applicable); e.g. by using a particular tool
no	activity is neither considered explicitly nor implied

Italic words show the general meaning of the corresponding case for the criterion. Bold words define the aspects that are considered for a criterion.

Table 4 Evaluation values for 'developing and setting up the adaptor'

Case	Description and values
explicit	activity is considered *explicitly*
	specify **when** the activity is carried out (if possible)
	specify **by whom** the activity is carried out (if possible)
implicit	activity is *implied*
	specify **how** it is implied (if applicable), e.g. by using a particular tool or framework
no	activity is neither considered explicitly nor implied

Italic words show the general meaning of the corresponding case for the criterion. Bold words define the aspects that are considered for a criterion.

Table 5 Evaluation values for 'executing the tests'

Case	Description and values
explicit	the activity is considered *explicitly*
	specify **when** the activity is carried out (if possible)
	specify **by whom** the activity is carried out (if possible)
	specify the test execution **mode** (online or offline), in case it is mentioned
	specify the specific testing **tool**, if it is mentioned.
implicit	The activity is *implied*
	specify **how** it is implied (if applicable), e.g. by using a particular tool
no	the activity is neither considered explicitly nor implied

Italic words show the general meaning of the corresponding case for the criterion. Bold words define the aspects that are considered for a criterion.

MBT tool that is used or implemented. In very few cases, test coverage criteria may also be worth mentioning. On the basis of Utting *et al.*'s taxonomy [12], test selection criteria can be of these types: Structural model coverage, data coverage, requirements coverage, random and stochastic, fault-based, or custom test specifications. An integration may also specify *who* would determine the coverage criteria (i.e. the roles involved). Table 2 shows the evaluation values related to this activity.

Generating tests: This activity is also an important activity in MBT, and hence should be addressed in any practical integration of MBT into development processes. However, the details (e.g. the test generation algorithm) are often not explained or are explained in an abstract manner. This activity is usually implied by the MBT tool or the technique used or implemented. The levels (types) of testing are also very important in determining when test cases are generated and what the output is (e.g. the format or type of the generated tests), and it is, therefore, recommended to mention the intended level of testing in a complete integration. Considering Utting *et al.*'s taxonomy, test generation technologies include random, search-based, model checking, symbolic execution, theorem proving, and constraint solving algorithms. Although tests are usually generated automatically (i.e. by a tool), in a complete integration it should also be specified *who* would perform this activity. Table 3 shows the evaluation values.

Developing and setting up the adaptor component: This activity depends on the technical properties of the testing environment and the SUT; thus, the adaptor is developed and set up in a case-by-case manner. This activity is not required to be discussed in the integration at the process level. Nevertheless, a very specific and low-level integration could specify and discuss the adaptor regarding the particular MBT tool used or suggested, though it might not provide useful information in terms of process-related issues. We focus on whether this activity is explicitly discussed, and if so when and by whom. Table 4 shows the evaluation values.

Executing the tests on the SUT: This is the main testing activity, and should, therefore, be specified by any integration, at least by indicating the level of testing at which MBT is applied. Depending on the level (type) of testing, test execution will be carried out in different phases of the development process and by different roles. Additionally, an integration may provide more details such as the testing mode (i.e. online or offline) or the particular tools used. Table 5 shows the evaluation values.

4.1.2 Criteria related to MBT artefacts: A practical integration should specify how each MBT artefact is generated or derived with respect to other artefacts (e.g. constructed from scratch, generated based on other artefacts, or reusing existing development artefacts). For example, Pretschner and Philipps [14] have recognised four high-level scenarios for using MBT, regarding the relationships between the models used for test case generation and the models used for development; these scenarios include: common model, automatic model extraction, manual modelling, and separate models.

Nevertheless, an integration may not specify the exact type/model of the test artefacts, thus allowing for flexibility. This flexibility should be mentioned explicitly in the integration, along with the relationships between the test models and other development artefacts. The main MBT artefacts have already been discussed in the context of MBT activities (i.e. which artefact pertains to which activity). In this section, each artefact is considered regardless of its associated activities to determine if an integration explicitly mentions the required testing artefacts, and if so, if it indicates (i) any specific type/format for the artefacts and (ii) how a testing artefact relates to other development artefacts. In some cases, the artefacts are implied by using a particular tool or framework. For test models, in particular, the degree of reuse is also mentioned including: complete reuse, partial reuse, and no reuse. The evaluation values related to the main four MBT artefacts, namely test models, test case specification, test suite and test cases, and adaptor, are respectively, shown in Tables 6–9.

The above criteria focus on the process-related issues and characteristics and do not cover those aspects of MBT processes that are typically used for selecting an MBT technique; for example, those provided in MBT taxonomies. Nevertheless, these criteria can be extended and refined based on such taxonomies, if required in a particular evaluation scenario.

4.2 Agile process characteristics

AD is iterative, so that requirements and solutions can evolve, and high-quality software increments can be delivered rapidly. Two of the most prominent agile methods are Extreme Programming (XP) [1] and Scrum [17].

Table 6 Evaluation values for 'test models'

Case	Description and values
explicit	the artefact is considered *explicitly*
	specify the **type/format** of the test models, possibly different types of models for different types of testing (if possible)
	specify how the artefact **relates** to other development artefacts (if possible). The relationship is then specified in terms of the degree to which existing artefacts are reused: *complete reuse, partial reuse,* or *no reuse*
implicit	the artefact is *implied*
	specify **how** it is implied, e.g. by using a particular tool or framework
no	the artefact is neither mentioned explicitly nor implied

Italic words show the general meaning of the corresponding case for the criterion.
Bold words define the aspects that are considered for a criterion.

Table 7 Evaluation values for 'test case specification'

Case	Description and values
explicit	artefact is considered *explicitly*
	specify the **type/format** of test case specifications, possibly different types of specifications for different types of testing (if possible)
	specify how the artefact **relates** to other development artefacts (if possible)
implicit	artefact is *implied*
	specify **how** it is implied, e.g. by using a particular tool or framework
no	artefact is neither mentioned explicitly nor implied

Italic words show the general meaning of the corresponding case for the criterion.
Bold words define the aspects that are considered for a criterion.

Table 8 Evaluation values for 'test suite and test cases'

Case	Description and values
explicit	artefact is considered *explicitly*
	specify the **type/forma**t of the test suite and test cases, possibly different types of suites/cases for different types of testing (if possible)
	specify how the artefact **relates** to other development artefacts (if possible)
implicit	artefact is *implied*
	specify **how** it is implied, e.g. by using a particular tool or framework
no	artefact is neither mentioned explicitly nor implied

Italic words show the general meaning of the corresponding case for the criterion.
Bold words define the aspects that are considered for a criterion.

In this research, we are interested in those characteristics of agile processes that can affect the application of MBT techniques in such processes or make the integration a challenge. Faragó [2] identifies two key requirements for MBT in AD, namely flexibility and rapid delivery, and argues that they can be achieved by underspecifying the models that MBT uses; the test models should be underspecifiable and iteratively refined so that they can be used in AD. 'Minimal modelling' and 'iterative/incremental development' were found as the most related and effective agile characteristics in practical application of MBT in agile processes; we will discuss each one separately.

Minimal modelling: This requirement is widely considered in existing studies on the application of MBT in agile methodologies under the well known challenge of misalignment of artefacts developed in agile software development projects and those required by MBT, in general. MBT's need for precise and elaborate models, often avoided by agile methods, is the most important reason cited in the literature for MBT's lack of suitability for agile processes [2, 4]. Recent proposals on the use of (lightweight) models/modelling in agile processes (e.g. [1]) and the use of practical models in MBT (e.g. Unified Modeling Language (UML) models [18]) has presented an opportunity for benefiting from MBT in agile processes.

Table 9 Evaluation values for 'adaptor'

Case	Description and values
explicit	artefact is considered *explicitly*
	specify any required **detail** about the adaptor (if possible)
	specify how the artefact **relates** to other development artefacts (if possible)
implicit	artefact is *implied*
	specify **how** it is implied, e.g. by using a particular tool or framework
no	artefact is neither mentioned explicitly nor implied

Italic words show the general meaning of the corresponding case for the criterion.
Bold words define the aspects that are considered for a criterion.

Table 10 Evaluation values for 'minimal modelling'

Case	Description and values
reuse	existing agile artefacts are **directly** used as test models
generated	test models are **generated** using existing artefacts
	test models are generated *automatically*
	test models are generated *manually*
	generation mode is not mentioned
no	test models are constructed from scratch

Italic words show the general meaning of the corresponding case for the criterion.
Bold words define the aspects that are considered for a criterion.

It is important to note that MBT artefacts, particularly the test models, have to be in line and compatible with the characteristics of agile processes. This also addresses an important maxim of AD: valuing 'working software over comprehensive documentation'; thus reducing/limiting extra non-software artefacts, which are typically required by MBT.

In the context of 'minimal modelling', we define the following integration criterion: Is it specified which agile artefact is used as the test models or as input for building the test models? In the latter case, is it indicated whether the test models are generated automatically or constructed manually? The evaluation values with respect to minimal modelling are shown in Table 10.

Iterative/incremental development: Agile software development methodologies use iterative and incremental development in order to handle evolving systems and changing requirements. In each iteration, a subset of the system requirements is analysed, designed, implemented, and tested incrementally. Accordingly, iterative testing is essential to quality assurance in agile/lightweight methodologies. As for MBT, test models have to be iteratively created to provide adequate information for the current iteration; test cases are then generated automatically.

In this context, test models evolve to specify new behaviour, and consequently, new versions of the test models are created. The most common approach in dealing with system evolution is to generate a new set of test cases from the new version of the test models. However, a more effective approach, which reduces the effort required, is to reuse previously generated test cases and identify those test cases that will support the development of new features in the current iteration (possibly with limited updates). In this case, new test cases may be generated for testing the new features that cannot be covered by existing test cases.

Additionally, test models become larger and more complex over time, and hence, the number of test cases grows continuously and rapidly; if not managed properly, this can increase the required testing effort to unmanageable levels, and will make it impossible to get frequent and quick feedback on the whole system, which is a mandatory requirement in agile/lightweight processes. Effective management of change (mainly via reuse of test cases) would be a promising approach for addressing this challenge.

We assess evolution in test models and test case reuse as an integration criterion that determines if an integration specifies whether (and how) it deals with 'evolution', and whether it

Table 11 Evaluation values for 'evolution' in test models

Case	Description and values
explicit	evolution is considered explicitly
implicit	evolution is implicitly considered
no	evolution is not mentioned (neither explicitly nor implicitly)

Table 12 Evaluation values for 'reuse' in test generation

Case	Description and values
yes	reuse of previously generated test cases is supported
no	reuse of previously generated test cases is not supported

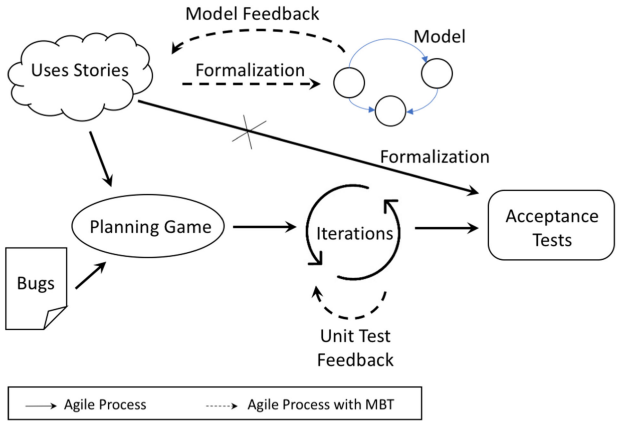

Fig. 2 *XP enhanced with MBT [23]*

supports test case 'reuse'. The evaluation values for evolution and reuse are shown in Tables 11 and 12, respectively.

4.3 Intended level of detail

Incorporation of the MBT process into a development process can be discussed at different levels with respect to the detail provided. An approach could provide abstract information, and thus allow for more freedom in the integration of testing activities or in the generation of testing artefacts. The following are the typical levels of integration; this is a non-exhaustive list, and there could be integrations that fall in between:

- *High/abstract*: A very abstract/high-level integration that does not consider the details including how, when, and by whom each MBT activity/task is carried out. In this case, MBT is typically discussed in the context of a generic definition of development processes.
- *Medium*: The use of MBT in a specific type/style of development process such as AD or model-driven development. In this case, the generic process of a given type or a particular process model (e.g. XP or Scrum for agile) is considered, and the integration of MBT is discussed in that context.
- *Low/detailed*: The application of MBT in a concrete/project-specific development process. In this case, a detailed description of MBT activities and artefacts in a concrete development process is provided (a complete process that has been used in a project).

The level of detail is somehow related to and affected by the target process: whether it is generic, is of a specific type or model, or is described by metamodels. The level provides an overview of the integration and guides us throughout the evaluation to identify how much detail should be provided in an integration. For now, the levels are defined as high (H), medium (M), and low (L).

4.4 Intended type of testing

As mentioned above, the types (levels) of testing in which MBT is going to be used affect the integration of MBT into a development process; e.g. to what extent MBT activities and development activities are required to be integrated or in which development phases (steps) MBT activities are carried out. MBT can be used for different levels of testing, namely UT, integration testing (IT), ST, acceptance testing (AT), and regression testing (RT).

5 MBT and agile processes: overview

This section presents an overview of the works that demonstrate the application of MBT in an agile process or provide an MBT approach for such processes. MBT can be applied in agile/lightweight processes using testing techniques based on UML models, metamodels, requirements, and architectural models [19]; hence, there are several studies on such testing techniques [20–22]. Herein, we are interested in those studies that in particular consider MBT throughout a complete agile/lightweight development process and illustrate its application.

MBT and AD are the two major approaches to increase the quality of software, and hence, MBT has been extensively studied in relation to AD. However, existing MBT techniques are not directly applicable in agile processes due to the well known problem of misalignment of artefacts developed in agile software development projects and those required by MBT, in general. MBT's need for precise and elaborate models is considered detrimental to agility; however, the use of (lightweight) models/modelling in agile processes (e.g. [1]) and practical models in MBT (e.g. [18]) seem to offer viable solutions.

There are several studies addressing this problem such as [4, 7], which, however, do not address the integration of tasks, the level of testing where MBT is used, and the interleavings of testing activities with other development activities.

Jalalinasab and Ramsin [19] take a general viewpoint on the use of MBT in agile processes and express the possible solutions as patterns, based on previous studies on using MBT including testing based on UML models, metamodels, requirements, and architectural models. Although the integration of MBT activities and artefacts is not explicitly discussed in their work, the patterns provide the details required for their application in a process, as they have illustrated in the context of the feature-driven development methodology.

Katara and Kervinen [4] introduce a domain-specific, use case-driven testing approach in which use cases are converted into sequences of so-called 'action words', which represent the events listed in the use cases. Action words are automatically translated into labelled transition systems (LTSs) for test generation. The approach relies on domain experts to design the test models.

Utting and Legeard [23] study the suitability of MBT-for-agile processes with respect to the three practises of agile methods that are related to MBT, namely test-driven development (TDD), customer acceptance tests, and agile modelling. They argue that MBT can offer an opportunity for generating a suite of unit tests from a small model (for TDD), which may reduce the cost of developing unit tests and allow a more rapid response to evolving requirements. They also contend that MBT can be applied for AT, for example, for ST in XP (Fig. 2), since test engineers help the customer in preparing the test models. Finally, they argue that agile methods encourage the customer to write acceptance tests in his/her own business language, which is quite close to MBT, as the business language is actually playing the role of a domain-specific language. They suggest that MBT tasks should preferably be carried out separately from the mainstream development activities, as illustrated in Fig. 3*a*. Nevertheless, they do not provide any detail on how MBT activities should actually be carried out.

Puolitaival [6] considers the adoption of MBT in an agile context. He first studies this adoption from a theoretical perspective and then presents a case where MBT is used in an agile project. He suggests including MBT test engineers in the agile team (Fig. 3*b*), thus avoiding the information gap between development teams and testing teams. He also states that two factors are important in the adoption of MBT in an agile context: (i) reasonable testing effort and (ii) managing MBT activities in the same way as development activities, e.g. by performing MBT through pair programming and TDD. Mobile-D^{TM} [24] has been used as the development process in the case study, which is an

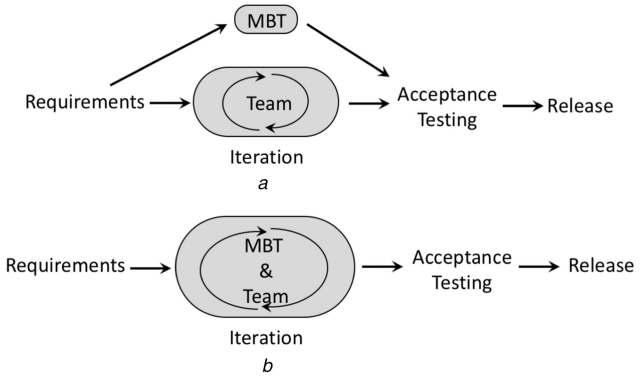

Fig. 3 *MBT team in agile processes*
(a) MBT outside an agile team [23], ***(b)*** MBT inside an agile team [6]

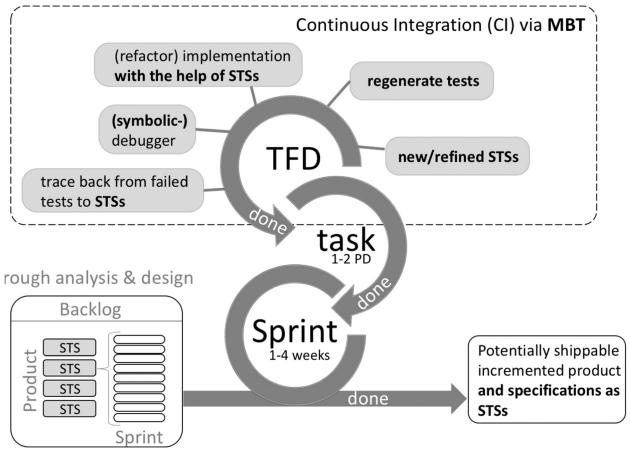

Fig. 4 *MBT with STSs in Scrum [2]*

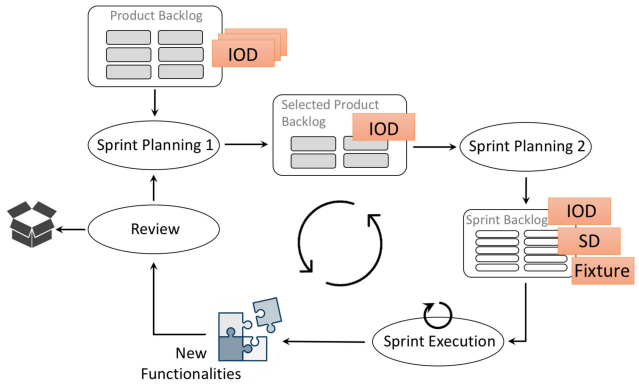

Fig. 5 *Scrum process extended with test models [26]*

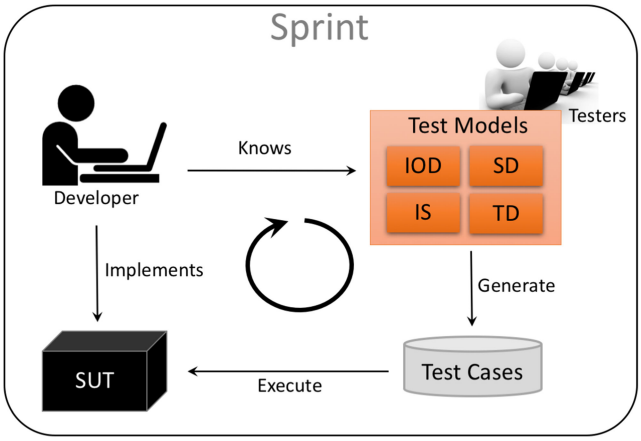

Fig. 6 *MBT activities embedded into the Scrum sprint [26]*

agile method integrating XP's development practises, Scrum's management practises, and rational unified process's phases. Conformiq Qtronic has been chosen as the test case generator, which exports the test suite as a text file. The test suite is then imported to the JwebUnit tool, which is used as the test execution platform. Test models are iteratively developed by Conformiq Qtronic Modeller.

In [2], Faragó investigates how MBT can improve AD and vice versa. He discusses the main benefits of applying MBT in AD and identifies two key requirements for using MBT in agile methods, namely flexibility and rapid delivery, which are achieved by underspecifying the models used for MBT. Accordingly, in [25], he presents a theoretical basis and introduces a new method for MBT, called lazy on-the-fly MBT, which addresses the aforementioned issues efficiently. He considers a combination of the most prominent agile methods, namely XP and Scrum, assuming that other agile methods lead to the same implications on verification. The test-first development cycle is thus changed into a rather specification-first development cycle, as illustrated in Fig. 4. Faragó uses the input–output conformance (ioco) theory [3], and introduces symbolic transition systems (STSs) and their underspecification and refinement possibilities, to address the requirements for MBT in AD. The models are underspecifiable and iteratively refined so that they are efficiently handled in test generation. Hence, the proposed approach can be used in AD for selecting more revealing tests and achieving higher coverage and reproducibility.

Löffler *et al.* [26] introduce a model-based AT method for Scrum that addresses two major challenges: (i) poor requirements specifications and communication with customers and (ii) automatic test script generation from abstract (test) models. Their approach follows the ideas of Utting and Legeard [23], in that (i) MBT must be embedded into agile processes and existing tool chains; (ii) there is a need for dedicated domain-specific modelling languages for making behavioural modelling easier and more user-friendly; and (iii) MBT must be linked to requirements analysis.

They extend Scrum by (i) adding two new models, namely interaction overview diagrams (IOD) and sequence diagrams (SD), for specifying the user stories and (ii) using FitNesse [27] as a well-established AT framework. They then refine the sprint-level activities via using the new (test) model artefacts throughout the sprint. Their extension of the Scrum process is shown in Fig. 5; Fig. 6 shows how MBT activities are embedded into Scrum sprints.

Vuori [28] reports very primitive incorporation of MBT into a simplified version of Scrum. He considers MBT in the overall flow of testing, assuming that the required test models are created in some way and are then used throughout the development process. The integration proposed is very abstract, and the report does not explain the testing activities and how they should be carried out.

Having considered iterative and incremental processes for system evolution, Ussami *et al.* [8] address the challenges of how to reuse test artefacts and how to select the relevant tests for implementing a new version of the system. They propose a process called D-MBTDD in which AD of a system is guided by model-based tests, focusing on reusing test artefacts and identifying the tests relevant to the development. They adapt the idea of delta-oriented model-based Software Product Line (SPL) RT [29]. In their work, finite state machines (FSMs) are used as test models to represent the system's behaviour and the deltas, which contain the differences between two versions. Although they explain the process for the first iteration and subsequent iterations, they do not provide the details on how testing artefacts are related to other development artefacts. They also assume that test models have already been created and validated by specialists.

Spichkova and Zamansky [30] propose a formal framework for MBT, called AHR: agile, human-centred and refinement-oriented. AHR is based on iterative construction of models and test plans and supports refinement at different levels of abstraction. It addresses the inevitable inconsistency, incompleteness, and inaccuracy of models and test plans in MBT, which necessitates constant revision and refinement of models and test plans.

Elallaoui *et al.* [31] present a model-driven testing approach within Scrum via applying an MDA approach for model transformation, as illustrated in Fig. 7. It generates TestNG [32] test cases based on their earlier works, which introduce techniques for (i) generating UML SDs from a set of user stories [33] and (ii) generating TestNG tests cases from UML SDs [34]. To generate the test model, a user story is chosen from a list of requirements, and the platform-independent model (PIM) is generated. This design model (PIM) is then transformed into a platform-independent test, which represents the test model. These two transformations, implemented in the AndroMDA framework [35], constitute the core of the approach. The first transformation takes the design model represented by UML SDs and generates specialised SDs conforming to the UML 2.0 Testing Profile (U2TP) [36]; these SDs constitute the test models. The second transformation takes the test models as input and produces TestNG test cases.

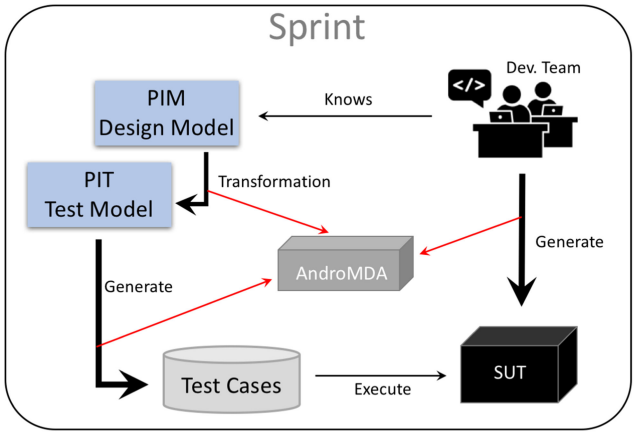

Fig. 7 *Transformation process within a Scrum sprint [31]*

6 Evaluation of existing works

Tables 13 and 14 present the results of evaluating existing works on the incorporation of MBT into AD processes with respect to our proposed criteria. These studies demonstrate the use of MBT in an agile/lightweight development process.

The evaluation results in Table 13 show that all of the existing works explicitly consider the 'building test models' activity. In all of them, the intended type of testing (e.g. AT) implies when this activity is carried out; it is assumed that the required test models are generated whenever the corresponding tests are created in the original process, without applying MBT. Excluding the works by Faragó [2, 25], all of the studies mention who would carry out this

Table 13 Evaluation of exiting works on the use of MBT, considering MBT activities, in the context of agile processes; empty cells represent 'no' values, which are not shown in this table for sake of clarity

Study	Level	Type	MBT activities				
			Building test models	Defining test case specification and selection criteria	Test generation	Setting up adaptor	Test execution
Katara and Kervinen [4]	M	AT	explicit (when: generating acceptance tests, roles: domain expert and tester, input: use cases)	explicit (criteria: custom coverage language, roles: —)	explicit (when: online test generation, roles: —, algorithm: search-based, tool: —)	—	explicit (when: for AT, roles: tester, mode: online, tool: —)
Utting and Legeard [23]	M	UT, AT	explicit (when: generating acceptance tests and (possibly) unit tests, roles: customer and tester, input: user stories)	requirements coverage (implied by AT based on user stories)	—	—	explicit (when: for AT, roles: —, mode: —, tool: —)
Puolitaival [6]	L	AT	explicit (when: generating acceptance tests, roles: tester, input: —)	criteria are implied by the testing framework: JWebUnit	implied by using the Conformiq Qtronic testing tool	implied by using the Conformiq Qtronic and JWebUnit testing tools	explicit (when: for AT, roles: —, mode: —, tool: Conformiq Qtronic and JWebUnit)
Faragó [2, 25]	M	UT, RT	explicit (when: generating acceptance tests, roles: —, input: product specification)	explicit (structural model coverage)	explicit (when: —, roles: —, algorithm: model checking, tool: —)	—	explicit (when: for AT, roles: —, mode: online/offline (lazy off-the-fly MBT), tool: —)
Löffler *et al.* [26]	M	AT	explicit (when: specifying requirements and generating AT, roles: product owner creates user stories with UML IODs, and developers refine the IODs by adding UML SDs, input: requirements)	explicit (criteria: data coverage, roles: tester)	explicit (when: before coding, roles: tester, algorithm: —, tool: selenium)	implied by using the selenium testing tool and the selected fixture	explicit (when: AT and IT, roles: testers, mode: —, tool: selenium)
Ussami *et al.* [8]	H	UT	explicit (when: before development, roles: specialist, input: requirements)	explicit (criteria: custom, roles: customer and developer and tester)	implied by the testing tool that is used for FSM-based test generation	—	—
Elallaoui *et al.* [31, 33, 34]	M	UT	explicit (when: —, roles: —, input: scenarios modelled as UML SDs)	—	explicit (when: before development, roles: —, algorithm: —, tool: U2TP and TestNG)	explicit (in the context of MDA, via model transformations and the testing tool applied)	implied by the testing tool and MDA

Table 14 Evaluation of exiting works on the use of MBT, considering MBT artefacts and agility, in the context of agile processes; empty cells represent 'no' in the evaluations which are not shown in this table for clarity

Study	MBT artefacts				Agility		
	Test models	Test case specifications	Test cases and test suite	Adaptor	Minimal modelling	Test models evolution	Test case reuse
Katara and Kervinen [4]	mentioned (use cases, by domain experts, are translated into action words and LTSs, LTSs)	mentioned (Test Case (TC) specifications are described in the introduced coverage language)	mentioned (sequence of action words: high level and then keywords: low level)	—	use cases are incrementally defined and translated into action words	implied by incremental use case definition and refinement and new test models	
Utting and Legeard [23]	mentioned (—, user stories are formalised as test models by the customer)	—	—	—	manually generated based on user stories (as AT by customer)	implied by creating/ evolving model for new user stories	—
Puolitaival [6]	mentioned (usage scenarios are modelled using the Qtronic modelling language (state machine) and then translated into JWebUnit, —)	mentioned (Conformiq Qtronic)	mentioned (Conformiq Qtronic)	mentioned (Conformiq Qtronic and JWebUnit)	manually generated based on scenarios	continuous modelling	—
Faragó [2, 25]	mentioned (user stories are identified based on product specifications and modelled as LTS)	implied by the testing technique: ioco	implied by the testing technique: ioco	—	manually generated based on user stories and product specifications	—	—
Löffler *et al.* [26]	mentioned (UML IOD and SD and fixture based on requirements)	implied by the testing framework	implied by the testing framework	implied by the testing framework	complete reuse: customer requirements are modelled by UML IODs which are then refined	—	—
Ussami *et al.* [8]	mentioned (system behaviour is represented with FSMs)	implied by the tool	implied by the tool	—	—	explicit (delta-oriented model-based TDD)	yes (reusable test cases)
Elallaoui *et al.* [31, 33, 34]	mentioned (test models are represented as U2TP SDs, automatically generated based on user scenarios modelled as UML SDs)	mentioned (TestNG test cases, automatically generated from U2TP SDs)	implied by the tool	mentioned (via MDA and the testing framework)	complete reuse: scenarios are modelled as UML SDs	implied by MDA and automatic model transformations	

activity and how the test models are built. All of the proposed approaches specify the information used to generate the test models, excluding the study by Puolitaival [6] which does not explicitly mention the exact input for this activity. Puolitaival does, however, indicate the type of testing (AT), which can indicate that the test models are related to customer requirements.

Most of the works specify the test specification and test selection criteria via considering the type of the test models or the particular testing tool used for MBT.

Other MBT activities including test generation, setting up the adaptor, and test execution are typically considered in the context of the testing platform or the test tool. In a few cases (e.g. [8, 25, 31]), more details (e.g. test execution mode and the adaptor) have been provided, depending on the scope and target of the research conducted.

According to Table 14, 'test models' are the main MBT artefact considered in the studies, which is somehow predictable as they are the main artefact in an MBT process. In all of the studies, test models are generated based on either use cases or user stories, which represent the requirements or system behaviour. The type/ format of the test models is also specified and described in all of the studies, which is highly dependent on the testing platform or the tool. Other artefacts are mostly implied by the testing tool or the proposed platform.

In terms of agile characteristics, all of the studies intend to introduce minimum additional models or modelling effort. In all of the techniques suggested, existing user stories, use cases, or usage scenarios, already produced in the development process, are formalised and translated into test models; in many cases, this is performed automatically or otherwise in a straightforward manner, so that agility is not jeopardised. Consequently, all of the existing works on the application of MBT in agile/lightweight processes consider 'minimal modelling' as an essential criterion or requirement for effective use of MBT. 'Evolution of test models', if explicitly mentioned, is addressed by generating new test models when new/updated requirements are identified. Only one paper [8] addresses evolution properly by introducing delta-oriented MBT for TDD; in this work, new test models are generated/updated based on previous test models. This paper is also the only one which discusses the 'reuse of test cases'. The other studies assume that test cases are always regenerated in case the models are changed.

7 Discussion

Our evaluation shows that few studies consider MBT in the context of a complete agile/lightweight development process. Although any MBT technique based on UML models, metamodels, requirements, and architectural models is of potential use in agile/ lightweight processes (as mentioned in Section 4), a practical solution needs to consider the details and implications throughout the whole development life cycle, addressing the process-related aspects as well.

Moreover, our evaluation shows that most of the existing works on the application of MBT in agile/lightweight processes consider AT as the main target for using MBT. Also, they largely focus on

the ‘building the model’ activity; other activities are typically considered or implied in the context of the testing tool or platform that is applied.

‘Test models’ are the main MBT artefact considered in these studies. Without exception, test models are created based on the requirements or the system behaviour (represented as user stories, use cases, usage scenarios etc.); this means that requirements are formalised as test models. This is largely because MBT is used for AT in these studies. Other MBT artefacts are implied or specified considering the testing platform or the tool used for testing. This observation follows the fact that most of the existing studies have argued that a major obstacle in the application of MBT in lightweight processes is that the models that are typically required for MBT are not the same as those produced in agile/lightweight processes or require details that are not considered essential in such processes. Therefore, using MBT in agile/lightweight processes has been mainly concerned with identifying models that can be adapted for use as test models in lightweight processes, focusing on how they should be constructed or generated so that agility is preserved.

The main outcome of the evaluation conducted herein is that in terms of agile characteristics, all the studies focus on minimal modelling, and only one study considers the ‘evolution’ of test models and the ‘reuse’ of test cases. Other works either address evolution implicitly by re-generating test cases from new/updated models or are completely oblivious in this regard. A substantial challenge in using MBT techniques in lightweight processes is to effectively deal with change. Evolution of the development artefacts including models is an inherent characteristic of lightweight processes. Therefore, keeping the test models updated is a substantial problem, which is not restricted to the use of MBT in lightweight processes; any development approach that is based on models has to tackle this problem in an effective manner.

In the context of MBT, incremental FSM-based testing, which is extensively researched in the past few years [37–42], can provide promising ideas for addressing the requirements that have not been addressed in existing MBT-for-agile techniques. This approach aims at modularising test case generation and execution based on evolving test models. Such a modularisation should eventually lead to saving time and effort in re-generating or re-executing the tests by focusing only on those parts that are influenced by the change. MBT of software product lines (e.g. [43–45]) can also provide viable solutions for dealing with incremental change and evolution in test models. Nevertheless, dealing with evolution and reuse depends on the specific MBT technique used for testing, and should be considered separately for each such technique.

8 Conclusion

This paper introduced a process-driven view on the use of MBT in agile/lightweight processes. It demonstrated that incorporation of MBT into a development process should consider an MBT approach throughout the whole development process, and thus explicitly deal with process-specific characteristics of the target development process. It defined two perspectives on the application of MBT in agile processes, namely the general MBT process and its requirements, and the particular characteristics of agile processes as related to testing. Accordingly, it introduced a set of integration criteria for complete incorporation of MBT into agile/lightweight processes, focusing on the activities, artefacts, and people involved in MBT. The integration criteria depict the specific characteristics of an appropriate MBT process for agile processes and allow analysts to identify the shortcomings and potential benefits of existing studies on the use of MBT in such processes.

Evaluation of existing studies shows that all the studies focus on minimal modelling and only one study considers the ‘evolution’ of test models and the ‘reuse’ of test cases, whereas these are considered as essential characteristics in agile processes. It can, therefore, be deduced that existing methods for applying MBT in an agile/lightweight development context are far from satisfactory, and extensive research is required for reaping the potential benefits of MBT in this important context.

The evaluation results produced herein can be used for improving existing approaches. They can also be used for developing a complete AD process that provides full and efficient support for MBT by addressing the shortcomings in existing approaches.

9 Acknowledgment

The work of M. Taromirad is supported by Iran's National Elites Foundation.

10 References

[1] Ambler, S.: ‘*Agile modeling: effective practices for eXtreme programming and the unified process*’ (John Wiley & Sons, Inc., NY, USA, 2002)

[2] Faragó, D.: ‘Model-based testing in agile software development’. 30 Treffen der GI-Fachgruppe Test, Analyse & Verifikation von Software (TAV), Testing Meets Agility, Munich, Germany, 2010

[3] Tretmans, J.: ‘Formal methods and testing’, in Hierons, R.M., Bowen, J.P., Harman, M. (Eds.): ‘*Model based testing with labelled transition systems*’ (Springer-Verlag, Berlin, Heidelberg, 2008), pp. 1–38

[4] Katara, M., Kervinen, A.: ‘Making model-based testing more agile: a use case driven approach’, in Bin, E., Ziv, A., Ur, S. (Eds.): ‘*Hardware and Software, Verification and Testing. HVC 2006. Lecture Notes in Computer Science*’ vol **4383**, (Springer, Berlin, Heidelberg, 2006)

[5] Tan, L., Sokolsky, O., Lee, I.: ‘Specification-based testing with linear temporal logic’. Proc. Int. Conf. Information Reuse and Integration IRI ‘04, Las Vegas, NV, USA, 2004, pp. 493–498

[6] Puolitaival, O.: ‘Adapting model-based testing to agile context’. Master thesis, University of Oulu, 2008

[7] Rumpe, B.: ‘Agile test-based modeling’. Proc. Int. Conf. Software Engineering Research & Practice SERP ‘06, Las Vegas, NV, USA, 2006

[8] Ussami, T.H., Martins, E., Montecchi, L.: ‘D-MBTDD: an approach for reusing test artefacts in evolving system’. Proc. Int. Conf. Dependable Systems and Networks Workshop, DSN-W ‘16, Toulouse, France, 2016, pp. 39–46

[9] Pretschner, A., Lötzbeyer, H., Philipps, J.: ‘Model based testing in incremental system development’, *Syst. Softw.*, 2004, **70**, (3), pp. 315–329

[10] Dias-Neto, A.C., Travassos, G.H.: ‘Model-based testing approaches selection for software projects’, *Inf. Softw. Technol.*, 2009, **51**, (11), pp. 1487–1504

[11] Dias-Neto, A.C., Travassos, G.H.: ‘Supporting the combined selection of model-based testing techniques’, *IEEE Trans. Softw. Eng.*, 2014, **40**, (10), pp. 1025–1041

[12] Utting, M., Pretschner, A., Legeard, B.: ‘A taxonomy of model-based testing approaches’, *Softw. Test. Verif. Reliab.*, 2012, **22**, (5), pp. 297–312

[13] da Silva, A.R.: ‘Model-driven engineering’, *Comput. Lang. Syst.*, 2015, **43**, (C), pp. 139–155

[14] Pretschner, A., Philipps, J.: ‘*Methodological issues in model-based testing*’ (Springer, Berlin, Heidelberg, 2005), pp. 281–291

[15] Dalal, S.R., Jain, A., Karunanithi, N.*, et al.*: ‘Model-based testing in practice’. Proc. Int. Conf. Software Engineering ICSE ‘99, Los Angeles, CA, USA, 1999, pp. 285–294

[16] Hierons, R.M., Bogdanov, K., Bowen, J.P.*, et al.*: ‘Using formal specifications to support testing’, *ACM Comput. Surv.*, 2009, **41**, (2), pp. 9:1–9:76

[17] Schwaber, K., Beedle, M.: ‘*Agile software development with scrum*’ (Prentice-Hall PTR, NJ, USA, 2001)

[18] Rumpe, B.: ‘*Model-based testing of object-oriented systems*’ (Springer, Berlin, Heidelberg, 2003), pp. 380–402

[19] Jalalinasab, D., Ramsin, R.: ‘Towards model-based testing patterns for enhancing agile methodologies’. Proc. Conf. New Trends in Software Methodologies, Tools and Techniques SoMeT ‘12, Genoa, Italy, 2012, pp. 57–72

[20] Bouquet, F., Grandpierre, C., Legeard, B.*, et al.*: ‘A subset of precise UML for model-based testing’. Proc. Int. Workshop on Advances in Model-based Testing, A-MOST ‘07, London, UK, 2007, pp. 95–104

[21] Kaplan, M., Klinger, T., Paradkar, A.M.*, et al.*: ‘Less is more: a minimalistic approach to UML model-based conformance test generation’. Proc. Int. Conf. Software Testing, Verification, and Validation ICST ‘08, Lillehammer, Norway, April 2008, pp. 82–91

[22] Sarma, M., Mall, R.: ‘System testing using UML models’. Proc. Asian Test Symp. ATS ‘07, Beijing, China, October 2007, pp. 155–158

[23] Utting, M., Legeard, B.: ‘*Practical model-based testing: a tools approach*’ (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007)

[24] Abrahamsson, P., Hanhineva, A., Hulkko, H.*, et al.*: ‘Mobile-D: an agile approach for mobile application development’. Proc. OOPSLA Companion to Annual ACM SIGPLAN Conf. Object-Oriented Programming Systems, Languages, and Applications, Vancouver, BC, Canada, 2004, pp. 174–175

[25] Faragó, D.: ‘Improved underspecification for model-based testing in agile development’. Proc. Int. Work Formal Methods and Agile Methods FM + AM ‘10, Pisa, Italy, 2010, pp. 63–78

[26] Löffler, R., Güldali, B., Geisen, S.: ‘Towards model-based acceptance testing for scrum’, *Softwaretechnik-Trends*, 2010, **30**, (3), pp. 1–5

[27] Martin, R.C., Martin, M.D., Wilson-Welsh, P.: ‘FitNesse – acceptance testing framework’, 2008

[28] Vuori, M.: ‘*Model-based testing in modern agile software development – how to integrate it into the development process?*’ (ATAC, Tampere University of Technology, Finland, 2014)

[29] Lity, S., Lochau, M., Schaefer, I., *et al.*: 'Delta-oriented model-based SPL regression testing'. Proc. Int. Work Product LinE Approaches in Software Engineering PLEASE '12, Zurich, Switzerland, 2012, pp. 53–56
[30] Spichkova, M., Zamansky, A.: 'A human-centred framework for sup-porting agile model-based testing'. Proc. Int. Conf. Advanced Information Systems Engineering CAiSE '16, Ljubljana, Slovenia, 2016
[31] Elallaoui, M., Nafil, K., Touahni, R.: 'Introducing model-driven testing in scrum process using U2TP and AndroMDA', *Int. Rev. Comput. Softw. (I.RE.CO.S.)*, 2017, **12**, (1), pp. 30–39
[32] 'TestNG'. Available at http://testng.org/doc/index.html/, accessed 29 September 2018
[33] Elallaoui, M., Nafil, K., Touahni, R.: 'Automatic generation of UML sequence diagrams from user stories in scrum process'. Proc. Int. Conf. Intelligent Systems: Theories and Applications SITA '15, Rabat, Morocco, 2015, pp. 1–6
[34] Elallaoui, M., Nafil, K., Touahni, R.: 'Automatic generation of TestNG tests cases from UML sequence diagrams in scrum process'. Proc. Int. Colloquium Information Science and Technology CiSt '16, Tangier, Morocco, 2016, pp. 65–70
[35] 'AndroMDA'. Available at http://www.andromda.org/, accessed 29 September 2018
[36] The Object Management Group. UML Testing Profile (UTP), 2013
[37] EI-Fakih, K., Yevtushenko, N., Bochmann, G.: 'FSM-based incremental conformance testing methods', *IEEE Trans. Softw. Eng.*, 2004, **30**, (7), pp. 425–436
[38] Jääskeläinen, A.: 'Filtering test models to support incremental testing'. Proc. Testing – Practice and Research Techniques, Berlin, Heidelberg, 2010, pp. 72–87
[39] Németh, G.Á., Pap, Z.: 'The incremental maintenance of transition tour', *J. Fundam. Inf.*, 2014, **129**, (3), pp. 279–300
[40] Pap, Z., Subramaniam, M., Kovács, G., *et al.*: 'A bounded incremental test generation algorithm for finite state machines', in Petrenko, A., Veanes, M., Tretmans, J., Grieskamp, W. (Eds.): '*Testing of Software and Communicating Systems. FATES 2007, TestCom 2007. Lecture Notes in Computer Science*' vol **4581**, (Springer, Berlin, Heidelberg, 2007)
[41] Simão, A., Petrenko, A.: 'Fault coverage-driven incremental test generation', *Comput. J.*, 2010, **53**, (9), pp. 1508–1522
[42] Varshosaz, M., Beohar, H., Mousavi, M.R.: 'Delta-oriented FSM-based testing', in Butler, M., Conchon, S., Zaidi, F. (Eds.): '*Formal methods and software engineering*' (Springer International Publishing, Cham, Switzerland, 2015) pp. 366–381
[43] Engström, E., Runeson, P.: 'Software product line testing – a systematic mapping study', *Inf. Softw. Technol.*, 2011, **53**, (1), pp. 2–13
[44] Oster, S., Wübbeke, A., Engels, G., *et al.*: 'Model-based software product lines testing survey', in Zander, J., Schieferdecker, I., Mosterman, P.J. (Eds.): '*Model-based testing for embedded systems*' (CRC Press, Boca Raton, FL, USA, 2011), pp. 339–381
[45] Thüm, T., Apel, S., Kästner, C., *et al.*: 'A classification and survey of analysis strategies for software product lines', *ACM Comput. Surv.*, 2014, **47**, (1), pp. 6:1–6:45