A Comparative Study of Software Testing Techniques

Meriem Atifi(✉), Abdelaziz Mamouni(✉), and Abdelaziz Marzak(✉)

Faculty of Sciences Ben M'sik,
University Hassan II of Casablanca, Casablanca, Morocco
Meryematif@gmail.com, Mamouni.abdelaziz@gmail.com,
Marzak@hotmail.com

Abstract. Nowadays, software systems have become an essential element in our daily life. To ensure the quality and operation of software, testing activities have become primordial in the software development life cycle (SDLC). Indeed, software bugs can potentially cause dramatic consequences if the product is released to the end user without testing. The software testing role is to verify that the actual result and the expected result are consistent and ensure that the system is delivered without bugs. Many techniques, approaches and tools have been proposed to help check that the system is defect free. In this paper, we highlight two software testing techniques considered among the most used techniques to perform software tests, and then we perform a comparative study of these techniques, the approaches that supports studied techniques, and the tools used for each technique. We have selected the first technique based on the 2014 survey [62] that heighted the motivations for using the Model-based-testing, and by analyzing the survey results we have found that some MBT limits are benefits in Risk based testing, the second technique in our study.

Keywords: Software systems · Software testing · Software testing approaches · Model-based testing · Risk-based testing

1 Introduction

Software system consists of a number of separate programs, configuration files, which are used to set up these programs, system documentation, which describes the structure of the system, and user documentation, which explains how to use the system and web sites for users to download recent product information [59]. Nowadays, software systems have become an essential part of our daily life. We use these software systems daily and have generally tried to keep them updated as much as possible. While in many times these software systems do not work as expected. Therefore, creating high-quality software is an intellectual challenge. Generally this quality is ensured by a test activity. However, this activity is time consuming, and too demanding as far as resources are concerned, and it is essential to ensure a certain percent of software quality. Indeed, a defect in software can have serious consequences for users and companies. These consequences may cause trivial issues viz; loss of money, time, business credibility or even loss of life. This was the case, for example in medicine in 1985, the Therac 25,

A. El Abbadi and B. Garbinato (Eds.): NETYS 2017, LNCS 10299, pp. 373–390, 2017.
DOI: 10.1007/978-3-319-59647-1_27

was a radiation therapy machine for treating cancer. A dysfunction of software has led to an overdose of radiation and was the cause of several deaths. Also the flight 501 failure of Ariane 5 in 1996 caused by a problem in specification, the estimated loss of this failure is 8.5 BILLION dollars. And more recently, on 6 January 2010, the German bank card holders designed by Gemalto had the unpleasant surprise of not being able to use their cards in some terminals. However, even if this bug lasted only one day, Gemalto has estimated the cost associated between 6 and 10 million. These accidents show that regardless of the scope of the software, it is necessary to validate and verify its operation and quality before using it. To ensure the quality and operation of software, testing activities have become primordial in the software development life cycle (SDLC). Indeed, software testing is a process of validating and verifying that a software product works as expected. Recently, the Software testing activity has changed dramatically, the complexity of IT systems has increased, the applications areas have been expanded, and the costs and consequences of bugs became higher, turning testing into an essential activity that should not be overlooked during the software development cycle. Testing is a vital part of software development, and it is important to start it as early as possible, and to make testing a part of the process of deciding requirements [60]. For that, in this present paper, we will present the tools, the processes, and the approaches of existing testing techniques. Also, this paper offers a detailed comparison between these testing techniques, especially, two major techniques viz; the model based testing technique (MBT), which is an application of model-based design to perform software testing or system testing. Models can be used to represent the desired behaviour of a System Under Test (SUT), or to represent testing strategies and a test environment; and the risk based testing technique (RBT), which is a type of software testing that functions as an organizational principle used to prioritize the tests of features and functions in software, based on the risk of failure, the function of their importance and likelihood or impact of failure [61].

The rest of this paper is organized as follows. Section 2 exposes general processes of MBT and RBT techniques, used tools and stockholders of each technique. Section 3 presents a classification of approaches related to each technique. Section 5 present some MBT and RBT Advantages and Limits, Sect. 6 discuss and presents the results of our analysis. Finally, Sect. 7 describes conclusion and future work.

2 MBT and RBT Processes

2.1 Model-Based Testing

The MBT (Model-Based Testing) is a form or a technique that aims to automatically generate test cases from formal specification or models describing the expected behaviour of the system under test. The behaviour model or formal specification are built from requirements and represents the software characteristics. It consists in managing and listing the requirements, creating a behavioural model of the SUT based on this list of requirements in order to generate abstracts tests cases and Requirements Traceability Matrix (RTX) by using the selection criteria that aim the model coverage and detect certain types of fault. The tests cases generated are then concretized, making

them executable on the SUT, the actual result and the expected result are then compared in order to get a verdict. Based on MBT application steps, used tools and stockholders, we present the following process.

As shown in Fig. 1 above, the process of Model-based testing can be splitted into five main steps. These steps will be explained further below.

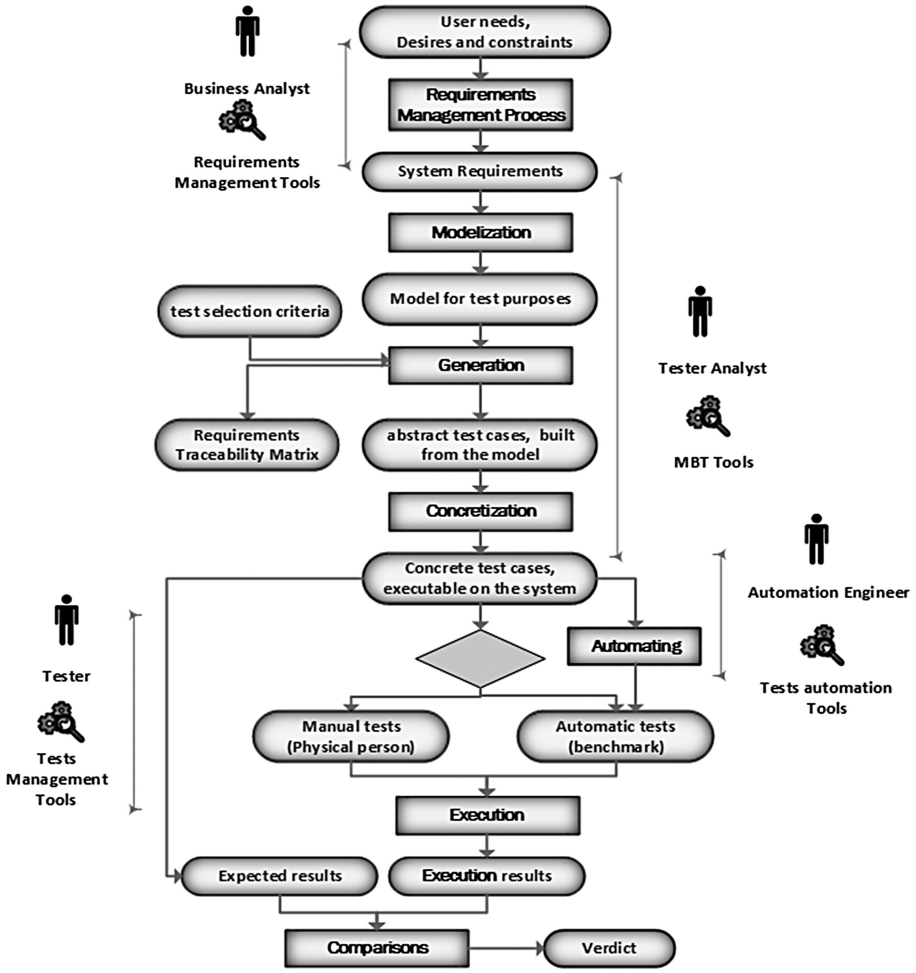

Fig. 1. Model-based testing general process

- Requirements management: The first step of model based testing is to collect customer needs, desires and constraints, manage and classify them as requirements. This first step is potentially the most important step in a process of testing software such as MBT. Requirement management step in MBT involves the collection, analysis, prioritization, validation, definition and control of all customer business

requirements, it serves to create a requirement repository that is the basis of communication between analysts and testers and is define in a structured way the expected result for the software, in different terms (functional, technical, security, load and response time…).

- Modelling: The main purpose of the modelling step in MBT is to model the system requirements, it consists in creating a behaviour model that describes the expected behaviour of the system under test and suitable for test purposes, this model is created by a tester analyst using requirements resulting from requirements management step and described in many ways, depending on the discipline. It can be described by use of diagrams, tables, text, or other kinds of notations. It might be expressed in a mathematical formalism or informally, where the meaning is derived by convention kinds of notations. They might be expressed in a mathematical formalism or informally, where the meaning is derived by convention.
- Generation: The generation step is realized on the basis of a test generator which takes as input the model designed in the modelling step and the test selection criteria selected by the test analyst and produces the abstract test cases from the model and a requirements traceability matrix that illustrates the link between tests and model elements covered by the tests.
- Concretization: The concretization step consists to translate the abstracts test cases to executables test cases in order to be executed on SUT. It consists in making the link between the model elements and the system's concrete elements, and involving specific adapters and manual intervention that requires the expertise of the test engineer.
- Execution: The execution step can be manually or automatically. In this phase, all the test cases are executed on the system under test, eventually the obtained results are then compared with the expected results to give a verdict for test cases and consequently give a status on the operation of the product [1, 2, 13, 15–18].

2.2 Risk-Based Testing

When we cannot test exhaustively, we must test selectively and achieve better with less in time and resources and without affecting the product quality. The RBT is a software testing technique or method that uses risk as a basis for test planning, It uses risk to select, prioritize and manage the appropriate tests during test execution and consequently to make sure that the limited time and resources are used to test the most important things [3, 5]. In simple terms, Risk is an undesirable event whose appearance is not certain and having as consequence negative results on the project objectives such as impact on completion date, costs, the quality, the company image, etc. Thus, the risk may be considered as the composition of two factors viz; the probability of occurrence of an undesirable event and the severity of the potential consequences of the undesirable event. Based on RBT application steps, used tools and stockholders [3, 7, 62], we present in Fig. 2 below, the process of risk-based testing that can be divided into five main steps.

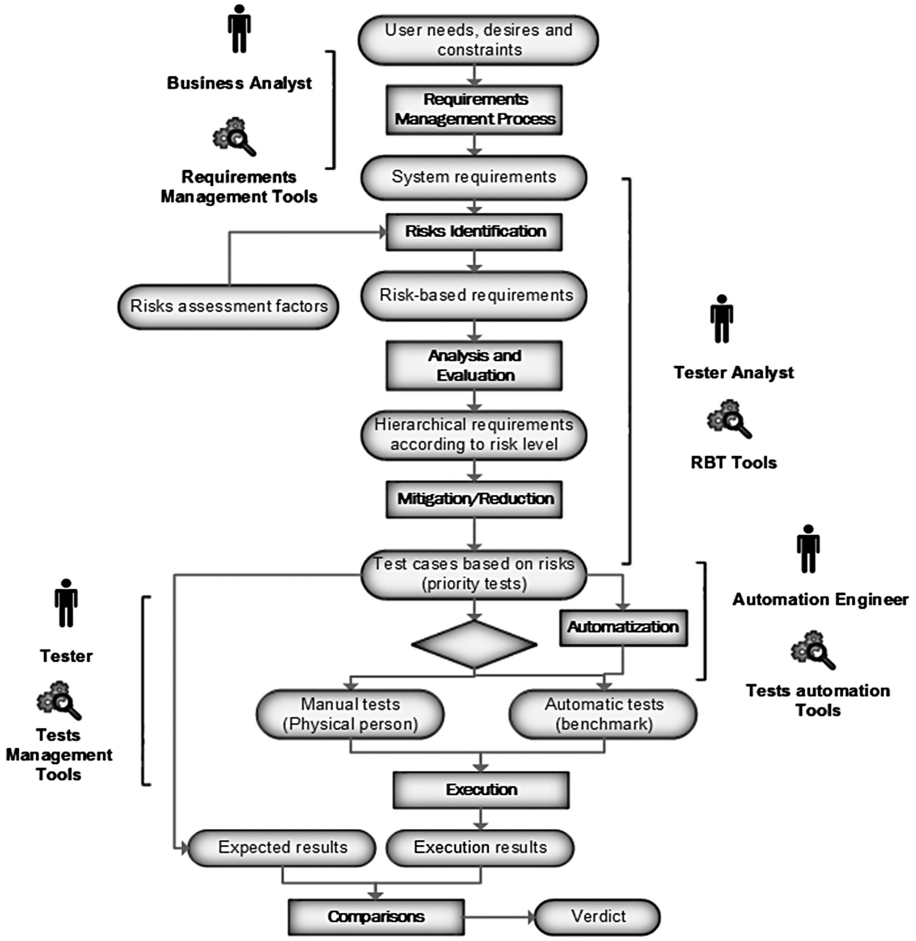

Fig. 2. Risk-based testing general process

- Requirements management: As the MBT process, RBT process start with requirements management step to extract and identify requirements from system specifications.
- Risks identification: Identifying risks is an absolutely essential activity in RBT process, it involves making a list of everything that might potentially come up and disrupt the normal flow of project, and provides the indicators that allows the organization to identify major risks before they impact operations and hence the business. It consists to identify and describe all requirements in terms of risk involved in the project. Thus at the end of this step all risk items are identified.
- Analysis and Evaluation: Risk analysis and evaluation is the second step of risk management in RBT process. It consists in studying the risks identified in risk identification phase, categorizing them, determining the level of risk by specifying likelihood and impact of the risk and then assigning the level of risk to each item.

- Mitigation/Reduction: The objective of Risk mitigation step is to reduce the Risk Impact or Risk Probability. It consists in looking for risk Mitigation where tests are built to mitigate the risk.
- Execution: The execution step in RBT consists in executing test cases resulting from reduction step according to prioritization and acceptance criteria identified in the risks report.

3 MBT and RBT Approaches Classification

3.1 Model-Based Testing Approaches Classification

Arilo et al. [16, 19] have classified MBT approaches into four categories, some of these approaches use UML diagrams, whereas, the others use a non-UML notations to represent software requirements or software internal structure. This classification is described and detailed in the form of table as show in Table 1 below.

Table 1. Classification of MBT approaches

Classifications	Approaches
C1	Model representing software requirements is described using UML diagrams
C2	Model representing software requirements is described using non-UML notation
C3	Model representing software internal structure is described using UML diagrams
C4	Model representing software internal structure is described using any non-UML notation

Utting et al. [2] have defined a taxonomy of model based testing that allows the characterization of different approaches to model-based testing, they have defined three general classes viz; model specification, test generation and test execution. Each of these classes is divided into various categories viz; model specification: It is divided into scope, characteristics and paradigm categories; test generation: It is divided into test selection criteria and technology categories; and test execution: It is divided into online test execution and offline test execution. Based on this taxonomy, Utting et al. have classified a collection of existing approaches in order to show the characteristics of those approaches to target various application domains. They have classified the existing approaches into two main categories: approaches to model-based test case generation and approaches to model-based test input generation. Felderer et al. [6] have defined a novel classification of model-based security testing approaches. They have classified the existing approaches into two dimensions viz; automated test generation and risk. The first dimension describes how much of the system and the security requirements is captured by formal models. The second dimension "risk" can have the values integrated into the model or not integrated into the model. Anand et al. [58] have performed a

survey of methodologies for automated software test case generation. They have classified the MBT approaches into three categories viz; axiomatic approaches that use scenario-oriented notations, finite state machine approaches that use state-oriented notations, and labelled transition system approaches that use process-oriented notations.

3.2 Risk-Based Testing Approaches Classification

In other respects, for RBT technique, Erdogan et al. [5] have classified the approaches that use both tests and risks into two global categories, some approaches when risk is used to focus testing, and others when test is used to focus risk. It defines the first category for test-based risk analysis (TR), and the second category for risk-based testing (RT). In Table 2 below we expose the approaches studied by Erdogan et al. [5] and Alam et al. [3] in order of this classification. Otherwise, depending on main focus, Erdogan et al. have classified RBT approaches into eight categories viz;

Table 2. TR & RT approaches classification

TR category	RT category
Wong 2005 [39] Schneidewind2007 [49] Bach- Inside-Out1999 [52]	Amland2000 [20], Felderer2012 [21], Felderer2013 [22], Felderer2015, Redmill2004 [23], Redmill2005 [24], Yoon 2011 [25], Gleirscher2011 [26], Gleirscher2013 [27], Ray2013 [28], Kloos2011 [29], Nazier2012 [30], Wendland2012 [31], Xu2012 [32], Zimmermann2009 [33], Chen2003 [34], Chen2002 [35], Entin2012 [36], Stallbaum2008 [37], Hosseingholizadeh2010 [38], Kumar2009 [40], Rosenberg1999 [41], Bai [42, 43], Casado2010 [44], Murthy2009 [46], Zech2011 [47], Zech2012 [48], Souza2010 [50], Bach-Outside-In1999 [51], Paul2002 [52], Stålhane2003 [54] and Palanivel2014 [9]

- Approaches that combine risk analysis and testing at a general level as Amland2000, Felderer2012, Felderer2013 and Redmill2004 and Redmill2005;
- Approaches with main focus on model-based risk estimation as Gleirscher2011, Gleirscher2013 and Ray2013;
- Approaches with main focus on test-case generation as Kloos2011, Nazier2012 and Xu2012;
- Approaches with main focus on test-case analysis as Chen2003, Chen2002 and Entin2012;
- Approaches based on automatic source code analysis as Wong 2005 and Hosseingholizadeh2010;
- Approaches targeting specific programming paradigms as Kumar2009 and Rosenberg1999;
- Approaches targeting specific applications as Bai2009, Bai2012, Casado2010 and Zech2011;
- Approaches' aiming at measurement in the sense that measurement is the main issue as Schneidewind2007 and Souza2009.

4 MBT and RBT Supporting Tools

To benefit fully from any technique or approach as MBT or RBT, the automation supports are required to automate as much as possible and to increase the reliability of the software testing process. In MBT, the challenge is that from a formal, semi-formal or informal models generate complete test cases without human interference. On the other hand, in RBT approach, the challenge is how to manage, select, and evaluate risk in testing process. In this context, when practitioners want to adopt an MBT or RBT approach, they therefore seek associated tools. For MBT, a number of model-based testing tools have been proposed [8, 10–12, 18]. We can classify these tools in different criteria viz; tool category: Commercial (CL), Open Source (OS) or Academic (AC); model type: UML, SysML, FSM, EFSM, Textual Models (TM), and more; and test type: Functional Testing (FT), Non Functional Testing (NFT), Structural Testing (ST). The Table 3 below describes these tools according to these criteria. For RBT, the most used tools are test management systems that support RBT approaches [14]. Table 4 exposes some of the test management tools that support RBT and some tools intended for RBT technique.

Table 3. MBT tools classification

Tool	Category	Model type	Software area	Test type
4Test	CL	TM	All	FT
BPM-Xchange	CL	BPMN	All	FT
Conformiq Creator	CL	Activity Diagrams & DSL	All	FT
Conformiq Designer	CL	UML State Machines and QML	All	FT
DTM	CL	Custom activity model	All	ST
MaTeLo	CL	Markov chains	All	FT
MBTsuite	CL	UML, BPMN	All	FT
RT-Tester	CL	UML, SysML and Matlab	All (embedded real-time systems)	FT
Smartesting CertifyIt	CL	UML, OCL and BPMN	All (Enterprise IT applications)	FT
Microsoft's SpecExplorer	CL	model programs in C#, FSM/ASM	All	FT
TEMPPO Designer (IDATG)	CL	Task flow model	All	FT
TestCast	CL	UML State Machines	All (Embedded Systems)	FT & ST
TestOptimal	CL	FSM & E-FSM	All	FT & NFT
T-VEC	CL	Simulink	All (Embedded Systems)	ST

(*continued*)

Table 3. (*continued*)

Tool	Category	Model type	Software area	Test type
Conformiq's Qtronic	CL	UML, QM & TM	All	FT
Test Designer	CL	UML	All	FT
LTG [2, 57]	CL	UML, OCL and B abstract machines	All	FT
mbt	OS	FSM/EFSM	All	FT
GraphWalker	OS	FSM	All (nondeterministic systems)	FT
JTorX [2, 56]	OS(AC)	LTS	All	FT
Modbat	OS	EFSM	Specialized for API testing of program libraries	FT
ModelJUnit	OS	FSM & EFSM	All	FT
OSMO	OS	Model programming Java	All	FT
PyModel	OS	Python source	All	ST
Tcases	OS	TM	All	FT
JSXM	AC	SXM	All	FT
MISTA	AC	PrT net	All	FT & NFT
MoMuT::UML	AC	UML state machines & OOAS	All	FT
MOTES	AC	EFSM	All (Embedded Systems)	FT
AGEDIS	AC	UML (AML)	All (Component based distributed Systems)	FT
ParTeG	AC	UML & OCL	All	FT

5 MBT and RBT Advantages and Limits

The MBT and RBT solutions are highly effective testing techniques that can be used to perform and manage software testing. Each solution has distinct benefits and limits. In this context, Legeard [4], has classified the major MBT benefits that solved some problems of classical approaches into six areas viz; SUT fault detection, reduced testing cost and time, improved test quality, requirements defect detection, tractability management, and requirements evolution. Also, he discussed some of fundamental limitations that limit the usage areas of MBT approaches. In the other hand, Alam [3] in his paper highlight some benefits and limits of RBT. Table 5 present some major advantages and disadvantages of RBT and MBT Approaches.

Table 4. RBT tools classification

Tool	Category	Model type	Software area	Test type
HP Quality Center	CL	-	All	FT
Kristoffer Tool [14]	AC	-	All	FT & NFT
NORIZZK.COM (SaaS platform)	CL	-	All	FT & NFT
Sonata	OS	-	All	FT & NFT
Casado Framework [45]	AC	Transaction Model	Web services models and standards	NFT
ReQtest(SaaS platform)	CL	-	All	FT & NF
SOASense™ framework [40]	AC	-	Aspect oriented programming	ST
Hosseingholizadeh Tool [38]	AC	-	All	ST
RBTTool [55]	AC-OS	-	All	FT
RiteDAP [37]	AC	Activity diagrams	All	FT

Table 5. MBT and RBT limitations and advantages

Ref.	Remaining problems (Limits)	Solved problems (Advantages)
Hartman et al. [53] Arilo et al. [19] Monalisa et al. [18]	**MBT approach** • Cannot manage outdated requirements when the software evolves • One of practical limitations of model based testing is tester skills; the model designers must be able to design the models, in addition to being experts in the application area • Difficulty to analyze failed tests when any of the generated tests is failed • Difficulty to model some parts of the system under test • Requires a formal specification or model to carry out testing • Test cases are tightly coupled to the model; the change of model gives rise to a generation of altogether different test cases	**MBT approach** • Allows improving the bugs' detection in system under test • Allows reducing testing time and costs • Allows improving testing quality and therefore software quality • The fact that the model is derived from the requirements allows to start very early in system cycle life and so allows to detect defaults in requirements • Allows the traceability management between requirements and the abstract model through the requirements traceability matrix generated in generating step in MBT process

(*continued*)

Table 5. (*continued*)

Ref.	Remaining problems (Limits)	Solved problems (Advantages)
	• Writing test cases that cover dynamic aspects of the system dependent on the engineer expertise • Difficulty to detect all the differences between model and implementation	• Allows to easily adapting the model to the new changes contributed to system under test and re-generate test case what makes easy adaptation of requirement's evolution and reduce the maintenance costs • Using MBT approach in testing activity allows to high level of automation and to generate high volumes of non-repetitive useful tests
Mottahir et al. [3] Erdogan et al. [5] Felderer et al. [7]	**RBT approach** • When some risks are not identified or marked as low, may cause problems in future if they become a reality • Managing traceability between requirements and tests is too expensive • Difficulty to associate concretes test cases to risks identified too abstract • Sometimes, some mitigation are very expensive in cost and time • Difficulty to identify and select the right stakeholders for risk assessment	**RBT approach** • Allows optimizing available time and resources without affecting product quality • The RBT activities can be started early in system cycle life and discovered defaults • Test in risk order gives the highest likelihood to discovering defects in severity order and therefore allows risking reducing • When time, money and resources are limited, RBT reduces the number of tests for adapting with available resources without impacting product quality • In RBT the communication is based on risk that is understandable by all stakeholders • Prioritize testing tasks more efficiently • Allows to detect high risk defects in software and therefore to reduce risks

6 Analyse and Discussion

Both studied techniques have their own processes, approaches, tools, merits and demerits. For risk-based testing, all approaches described in this paper use risk to prioritize what to test and focus on activities related to risk identification, analysis and prioritizing. Most RBT approaches are black-box testing that takes as input software requirements. In this category we find some approaches which are intended to functional testing as Amland, Chen, Bach, and others that are proposed for non functional testing as Zech, Xu and Bai. Otherwise, for white-box testing, we find a limited number of RBT approaches which are intended to Structural testing like Wong and

Hosseingholizadeh. For model-based testing, the general idea is that from an explicit behaviour model that represents behaviour of system under test, generate test cases to validate the expected behaviour of the system under test. Based on studied classifications and after analyzing different approaches of model-based testing, we concluded that we can also classify existing MBT approaches according to different criteria viz; testing type, testing level, testing sources, and notation type used to represent testing sources. This classification is very detailed and it facilitates the selection of MBT approaches according to the test context (Table 6).

Table 6. MBT approaches classification

Criteria	Categories
Testing type	• Category approaches which are intended to functional testing • Category approaches which are intended to non functional testing • Category approaches which are intended to Structural testing
Testing level	• Category approaches which are intended to system level testing • Category approaches which are intended to integration level testing • Category approaches which are intended to unit/component level testing • Category approaches which are intended to regression level testing
Testing source	• Category approaches which uses software requirements as testing source • Category approaches which uses software internal structure as testing source
Testing notation type	• They uses Graphical notation to describe and represent testing source • They uses Textual/Scripting notation to describe and represent testing source • They uses Symbolic (completely mathematical) notation to describe and represent testing source

Based on MBT Approaches Characterization proposed by Arilo and Guilherme [19], we expose some approaches in order of proposed classification (Tables 7, 8, 9 and 10).

Table 7. MBT approaches classification in term of testing type

Functional testing	Non-functional testing	Structural testing
Abdurazik2000, Crichto 2001, Chen2002, Cavarra2003, Botaschanjan2004, Andrews2005, Bernard2006, Sokenou2006	Bousquet1999, Garousi2006, Mandrioli1995, Offutt1999b, Parissis1996, Pretschner2001, Richardson1992, Rumpe2003, Felderer2012	• Xu2006 • Kim1999 • Chang1999

Table 8. MBT approaches classification in term of testing level

Unit/Component testing	Integration testing	System testing	Regression testing
• Kim1999 • Dalal999 • Briand2006 • Barbey1996	• Bertolino2003 • Bertolino2005 • Beyer2003 • Chen2005	• Abdurazik2000 • Briand2004 • Crichton2001 • Legeard2004	• Briand2002 • Chen2002 • Den2004 • Tahat2001

Table 9. MBT approaches classification in term of testing source

Software requirements	Software internal structure
Ammann1994, Bernard2006, Belletini2005, Cavarra2003, Friedman2002 and Offut1999	Kim1999, Legard2004, Chang1999, Garousi2006 and Xu2006

Table 10. MBT approaches classification in term of testing notation type

Graphical notation	Textual notation	Symbolic notation
Bertolino2003, Briand2002, Kansomkeat2003, Lund2006, Sokenou2006 and Zhen2004	Bousquet1999, Mandrioli1995, Tahat2001, Hartmann&Nagin2004 and Tan2004	• Legeard2004 • Richardson1996 • Ammann1994

Table 11. RBT approaches classification in term of testing type

Functional testing	Non-functional testing	Structural testing
Amland2000, Bach-Outside-In1999, Bach-Inside-Out1999, Chen2003, Chen2002, Paul2002, Scheafer, Felderer2012, Stallbaum2008, Zimmermann2009, Wendland2012, Stålhane and Souza2010	Bach-Outside-In1999, Palanivel201, Zech2011, Zech2012, Murthy2009, Casado2010, Xu2012, Bai2009	Rosenberg1999, Hosseingholizadeh2010 and Wong 2005

Table 12. RBT approaches classification in term of testing level

Unit/Component testing	Integration testing	System testing	Regression testing
Paul2002, Wong 2005	Paul2002, Wong 2005	Amland2000, Bach-Outside-In1999, Bach-Inside-Out1999, Paul2002, Stallbaum2008, Zimmermann2009 and Wendland2012	Chen2003, Chen2002

For RBT technique, After Analyzing existing approaches, we concluded that we can also classify them according to the following criteria: Testing type, Testing level, Testing sources, and Notation type used to represent testing sources if exist (Tables 11, 12, 13 and 14).

Table 13. RBT approaches classification in term of testing source

Software requirements	Software internal structure
Amland2000, Chen2003, Chen2002, Bach1999, Redmill2004, Redmill2005, Felderer2012, Zech2011, Zech2012, Entin2012, Stallbaum2008, Zimmermann2009, Wendland2012, Stålhane2003, Souza2010 and Bai2009	Rosenberg1999, Hosseingholizadeh2010 and Wong 2005

Table 14. RBT approaches classification in term of testing notation type

UML notation	Non UML notation
Stålhane, Chen2002, Chen2003, Entin2012, Stallbaum2008, Wendland2012, Stallbaum2008 and Wendland2012	Felderer2012, Felderer2013, Zech2012 and Xu2012

7 Conclusion and Perspective

In this paper, we have studied the main two techniques of software testing. The idea of the first technique is to use the abstractions of a system under test and its environment to automatically generate test cases. MBT consists to create an abstract system model that specifies the behaviour of the SUT, and then generate test cases. The key points of MBT are the modelling behaviour of the SUT for test generation, the test generation strategies and techniques, and the concretization of abstract tests into concrete, executable tests. The second is a technique that aims to minimize the software risks and testing problems. RBT consists of a set of activities regarding risk factors identification associated to software requirements. Once identified, the risks are prioritized according to its likelihood and impact and the test cases are projected based on the strategies or approaches for treatment of the identified risk factors. The test efforts are continuously adjusted according to the risk monitoring. Based on our study between MBT and RBT techniques we are identifying the following research tasks in the area of model-based testing and risk based testing viz; Proposition of a meta-model that represent model-based testing technique; Proposition of a meta-model that represent risk-based testing technique; Proposition of a novel testing approach based on model based testing and risk based testing techniques to overcome some testing limitations; and Make some case studies by applying the novel testing approach to obtain empirical results and compare our approach over existing approaches.

References

1. Pretschner, A.: Model-based-testing. In: ICSE, pp. 722–732 (2005)
2. Utting, M., Pretschner, A., Legeard, B.: A Taxonomy of Model-Based Testing, a School of Computing and Mathematical Sciences (2006)
3. Alam, M.M.: Risk-based testing techniques: a perspective study. Int. J. Comput. Appl. **65**, 33–41 (2013). (0975–8887)
4. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. ACM, Inc. (2017)
5. Erdogan, G.: Approaches for the combined use of risk analysis and testing: a systematic literature review. Int. J. Softw. Tools Technol. Transfer **16**, 627–642 (2014). Springer-Verlag Berlin Heidelberg
6. Felderer, M.: A classification for model-based security testing. In: The Third International Conference on Advances in System Testing and Validation Lifecycle (2011)
7. Felderer, M.: A taxonomy of risk-based testing. Int. J. Softw. Tools Technol. Transfer **16**, 559–568 (2014)
8. Micskei, Z.: Model-Based Testing (MBT), Online Dictionary. http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html
9. Palanivel, M., Selvadurai, K.: Risk-driven security testing using risk analysis with threat modeling approach, Palanivel and Selvadurai SpringerPlus (2014)
10. Hartman, A.: AGEDIS: Model-Based Test Generation Tools, January 2009. http://www.agedis.de
11. Dranidis, D.: JSXM: a tool for automated test generation. In: Proceeding SEFM 2012 Proceedings of the 10th International Conference on Software Engineering and Formal Methods (2012)
12. Aichernig, B.: MoMuT: UML model-based mutation testing for UML. In: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST) (2015)
13. Pretschner, A.: One evaluation of model-based testing and its automation. In: ICSE 2005, 15–21 May 2005
14. Kristoffer, L.: A Software Tool for Risk-based Testing. http://www.idi.ntnu.no/grupper/su/fordypningsprosjekt2004/Jorgensen2004.pdf
15. Dalal, S.R., Jain, A., Karunanithi, N.: Model-based testing in practice. In: ICSE 1999, May 1999
16. Dias Neto, A.C., Travassos, G.H.: A survey on model-based testing approaches: a systematic review. In: WEASEL Tech 2007, November 2007
17. Cristi, M.: Using {log} as a Test Case Generator for Z Specifications. GNCS project PICT 2011-1002
18. Sarma, M., Murthy, P.V.R.: Model-based testing in industry – a case study with two MBT tools. ACM, 2–8 May 2010
19. Dias Neto, A.C., Travassos, G.H.: Characterization of model-based software testing approaches, Technical report at PESC/COPPE/UFRJ, Brazil, Published August 2007
20. Amland, S.: Risk-based testing: risk analysis fundamentals and metrics for software testing including a financial application case study. J. Syst. Softw. **53**, 287–295 (2000)
21. Felderer, M., Haisjackl, C., Breu, R., Motz, J.: Integrating manual and automatic risk assessment for risk-based testing. In: Biffl, S., Winkler, D., Bergsmann, J. (eds.) SWQD 2012. LNBIP, vol. 94, pp. 159–180. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27213-4_11

22. Felderer, M., Ramler, R.: Experiences and challenges of introducing risk-based testing in an industrial project. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2013. LNBIP, vol. 133, pp. 10–29. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35702-2_3
23. Redmill, F.: Exploring risk-based testing and its implications. Softw. Test. Verif. Reliab. **14**, 3–15 (2004)
24. Redmill, F.: Theory and practice of risk-based testing. Softw. Test. Verif. Reliab. **15**, 3–20 (2005)
25. Yoon, H., Choi, B.: A test case prioritization based on degree of risk exposure and its empirical study. Int. J. Softw. Eng. Knowl. Eng. **21**, 191–209 (2011)
26. Gleirscher, M.: Hazard-based selection of test cases. In: Proceeding if the Sixth International Workshop on Automation of Software Test (AST 2011), pp. 64–70. ACM, New York (2011)
27. Gleirscher, M.: Hazard analysis for technical systems. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2013. LNBIP, vol. 133, pp. 104–124. Springer, Heidelberg (2013). doi:10.1007/978-3-642-35702-2_8
28. Ray, M., Mohapatra, D.P.: Risk analysis: a guiding force in the improvement of testing. IET Softw. **7**, 29–46 (2013)
29. Kloos, J., Hussain, T., Eschbach, R.: Risk-based testing of safetycritical embedded systems driven by fault tree analysis. In: Proceeding of the Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 2011), pp. 26–33. IEEE, New York (2011)
30. Nazier, R., Bauer, T.: Automated risk-based testing by integrating safety analysis information into system behavior models. In: Proceeding of the 23rd International Symposium on Software Reliability Engineering Workshops (ISSREW 2012), pp. 213–218. IEEE, New York (2012)
31. Wendland, M.-F., Kranz, M., Schieferdecker, I.: A systematic approach to risk-based testing using risk-annotated requirements models. In: Proceeding of the Seventh International Conference on Software Engineering Advances (ICSEA 2012), pp. 636–642. IARA (2012)
32. Xu, D., Tu, M., Sandford, M., Thomas, L., Woodraska, D., Xu, W.: Automated security test generation with formal threat models. IEEE Trans. Dependable Secure Comput. **9**, 526–540 (2012)
33. Zimmermann, F., Eschbach, R., Kloos, J., Bauer, T.: Risk-based statistical testing: a refinement-based approach to the reliability analysis of safety-critical systems. In: Proceeding of the 12th European Workshop on Dependable Computing (EWDC 2009), pp. 1–8 (2009)
34. Chen, Y., Probert, R.L.: A risk-based regression test selection strategy. In: Proceeding of the 14th IEEE International Symposium on Software Reliability Engineering (ISSRE 2003), Fast Abstract, pp. 305–306. Chillarege Press (2003)
35. Chen, Y., Probert, R.L., Sims, D.P.: Specification-based regression test selection with risk analysis. In: Proceeding of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON 2002), pp. 1–14. IBM Press, New York (2002)
36. Entin, V., Winder, M., Zhang, B., Christmann, S.: Introducing model-based testing in an industrial scrum project. In: Proceeding of the Seventh International Workshop on Automation of Software Test (AST 2012), pp. 43–49. IEEE, New York (2012)
37. Stallbaum, H., Metzger, A., Pohl, K.: An automated technique for risk-based test case generation and prioritization. In: Proceeding of the Third International Workshop on Automation of Software Test (AST 2008), pp. 67–70. ACM, New York (2008)
38. Hosseingholizadeh, A.: A source-based risk analysis approach for software test optimization. In: Proceeding of the Second International Conference on Computer Engineering and Technology (ICCET 2010), vol. 2, pp. 601–604. IEEE, New York (2010)

39. Wong, W.E., Qi, Y., Cooper, K.: Source code-based software risk assessing. In: Proceeding of the 2005 ACM Symposium on Applied Computing (SAC 2005), pp. 1485–1490. ACM, New York (2005)
40. Kumar, N., Sosale, D., Konuganti, S.N., Rathi, A.: Enabling the adoption of aspects-testing aspects: A risk model, fault model and patterns. In: Proceeding of the Eighth ACM International Conference on Aspect-Oriented Software Development (AOSD 2009), pp. 197–206. ACM, New York (2009)
41. Rosenberg, L., Stapko, R., Gallo, A.: Risk-based object oriented testing. In: Proceeding of the 24th Annual Software EngineeringWorkshop, pp. 1–6. NASA, Software Engineering Laboratory (1999)
42. Bai, X., Kenett, R.S.: Risk-based adaptive group testing of semantic web services. In: Proceeding of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009), vol. 2, pp. 485–490. IEEE, New York (2009)
43. Bai, X., Kennett, R.S., Yu, W.: Risk assessment and adaptive group testing of semantic web services. Int. J. Softw. Eng. Knowl. Eng. **22**, 595–620 (2012)
44. Casado, R., Tuya, J., Younas, M.: Testing long-lived web services transactions using a risk-based approach. In: Proceeding of the 10th International Conference on Quality Software (QSIC 2010), pp. 337–340. IEEE, New York (2010)
45. Casado, R., Tuya, J., Younas, M.: A framework to test advanced web services transactions. In: Proceeding of the 4th International Conference on Software Testing, Verification and Validation (ICST 2011), pp. 443–446. IEEE, New York (2011)
46. Murthy, K.K., Thakkar, K.R., Laxminarayan, S.: Leveraging risk based testing in enterprise systems security validation. In: Proceeding of the First International Conference on Emerging Network Intelligence (EMERGING 2009), pp. 111–116. IEEE, New York (2009)
47. Zech, P.: Risk-based security testing in cloud computing environments. In: Proceeding of the Fourth International Conference on Software Testing, Verification and Validation (ICST 2011), pp. 411– 414. IEEE, New York (2011)
48. Zech, P., Felderer, M., Breu, R.: Towards risk-driven security testing of service centric systems. In: 2012 12th International Conference on Quality Software (QSIC). IEEE (2012)
49. Schneidewind, N.F.: Risk-driven software testing and reliability. Int. J. Reliab. Qual. Saf. Eng. **14**, 99–132 (2007)
50. Souza, E., Gusmão, C., Venâncio, J.: Risk-based testing: a case study. In: Proceeding of the Seventh International Conference on Information Technology: New Generations (ITNG 2010), pp. 1032–1037. IEEE, New York (2010)
51. Bach, J.: Heuristic Risk-Based Testing. Software Testing and Quality Engineering Magazine, November 1999
52. Paul, G.: Risk-Based E-Business Testing, pp. 3–29, 51–80 (2002). ISBN: 1580533140
53. Hartman, A.: The AGEDIS tools for model based testing. In: ISSTA 2004 Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (2004)
54. Stålhane, T.: Risk Analysis as a Prioritizing Mechanism in EuroSPI (2003)
55. http://promise.cin.ufpe.br/rbttool/
56. Tretmans, J., Brinksma, E.: Côte de Resyste – automated model based testing. In: Progress 2002 – 3rd Workshop on Embedded Systems (2002)
57. Bouquet, F., Legeard, B., Peureux, F., Torreborre, E.: Mastering test generation from smart card software formal models. In: Proceedings of International Workshop on Construction and Analysis of Safe, Secure and Interoperable Smart devices (2004)

58. Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P.: An orchestrated survey of methodologies for automated software test case generation. J. Syst. Softw. **86**, 1978–2001 (2013)
59. http://unina.stidue.net/Ingegneria%20del%20Software%202/Materiale/Sommerville%20-%20Software%20Engineering%208e.pdf
60. https://msdn.microsoft.com/en-us/library/jj159342.aspx
61. https://www.dice.com/skills/Risk-based+testing.html
62. Legeard, B., et al.: Model-based-Testing User Survey: Results (2014). http://model-based-testing.info/wordpress/wp-content/uploads/2014_MBT_User_Survey_Results.pdf