

Towards a Business Process Model-based Testing of Information Systems Functionality

Anastasija Nikiforova^a and Janis Bicevskis^b
Faculty of Computing, University of Latvia, 19 Raina Blvd., Riga, Latvia

Keywords: Information System, Model-based Testing, Functional Testing, Data Object, Executable Models.

Abstract: The main idea of the solution is to improve testing methodology of information systems (IS) by using data quality models. The idea of the approach is as follows: (a) first, a description of the data to be processed by IS and the data quality requirements used for the development of the test are created, (b) then, an automated test of the system on the generated tests is performed. Thus, the traditional software testing is complemented with new features – automated compliance checks of data to be entered and stored in the database. The generation of tests for all possible data quality conditions creates a complete set of tests that check the operation of the IS on all possible data quality conditions. Since this paper describes the first steps that are taken moving towards the proposed idea, it aims to (a) define the aim of the initiated research and (b) to choose the main components and to propose their combination resulting in the architecture of the idea to be implemented.


1 INTRODUCTION


Software testing is vitally important in the software development process, as illustrated by the growing market for automated testing tools (Utting & Legeard, 2010). Obviously, software testing plays a crucial role, therefore the problem of software correctness has been debated since the beginning of programming. At the beginning, software testing was considered as bug searching together with debugging. Testing was singled out as an independent phase only in the 70s. Nowadays, numerous scientific and practical studies are devoted to software testing. And the main aim of these studies is to get reliable and trustful software that can be used in everyday life. Unfortunately, this aim is not succeeded, yet, and a high number of studies as well as practical solutions are needed to solve this problem. The proposed testing strategies and methods cannot ensure the reliability of the software. Faults and bugs in the software still cause failures, despite the enormous resources spent on the developing and testing the software. For instance, according to (Utting & Legeard, 2010), software testing consumes between

30 and 60 percent of software development resources. Automating software testing is difficult, and this is sometimes done manually without any guarantees regarding the effectiveness of testing.

Therefore, we are launching a new research aimed to improve the complete testing methodology of information systems (IS) using business process and data quality models. The main idea of the approach to be developed is as follows: (a) first, the description of the data to be processed by IS and its processing rules are developed; they are further used to develop test manually or at least in a semi-automated way, (b) then, an automated test of the system on the generated tests is performed. Thus, traditional manual software testing is complemented with new feature – automated checks of compliance of data entered as a result of automated business processes and stored in a database.

This paper is only the first step towards this solution, and the main aim of this paper is to define an idea of the solution, that will include (a) a choice of the components among different alternatives that will form the proposed solution and (b) a proposal on the architecture of the idea to be implemented. The future works will cover the implementation of the

^a  <https://orcid.org/0000-0002-0532-3488>

^b  <https://orcid.org/0000-0001-5298-9859>

proposed idea and the approbation of new technology for testing and software development. This will at least partially lead to the main aim of developing technology of creating reliable and trustful software.

The paper deals with the following issues: a short overview of the concept of testing and the changes of testing paradigm through the years (Section 2), a rationale for the proposed solution (Section 3), a description of the model-based testing concept (Section 4), a description of the proposed solution (Section 5), conclusions and future work (Section 6).

2 THE CONCEPT OF TESTING

In recent years, the definition of the term “testing” has changed significantly, therefore, this term should be discussed at least a little. At the very beginning, by testing was meant bug searching together with debugging, however, nowadays this concept become broader. Nowadays, the main aim of testing is to get reliable and trustful software that can be used in everyday life.

According to IEEE Software Engineering Body of Knowledge (IEEE Computer Society, 2004) and (Olsen et al., 2018), “**testing** is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems”.

Another more extended definition states that “testing is the activity of executing a system in order to detect failures; it is different from, and complementary to, other quality improvement techniques such as static verification, inspections, and reviews. It is also distinct from the debugging and error-correction process that happens after testing has detected a failure” (Utting & Legeard, 2010). The necessity of distinction of two more concepts appears: (a) a **failure** - an undesired behaviour that can be typically observed during the execution of the system being tested and (b) a **fault** - an error in the software, usually caused by human error in the specification, design or coding process. The execution of the faults in the product cause failures. Once a failure is observed, an investigation to find the fault that caused this failure can be started, that results with a correction of that fault. Since both of these definitions are mainly related to the production process, more specific definition related to software testing should be discussed. According to Utting (2010, 2012) and Olsen (2018) with their co-authors’, **software testing** consists of “the dynamic verification of the behaviour of a program on a finite set of test cases, suitably selected from the usually finite execution domain, against expected behaviour”.

By term “**dynamic**” is meant an execution of the program with specific input values in order to find failures in its behaviour. One of the main advantages of (dynamic) testing is that the actual program is executed either in its real environment or in an environment with simulated interfaces, as close as possible to the real environment. So not only the correctness of the design and code are tested, but also the compiler, the libraries, the operating system and network support etc.

By “**finite**” Utting means that exhaustive testing is not possible or practical for most programs since they usually have a high number of allowable inputs to each operation, and even more invalid or unexpected inputs that must processed as well, however the possible sequences of operations is usually infinite. Thus, there is necessity to select a limited number of tests, so that tests can be performed at an affordable time without interfering the staff working with software.

“**Selected**” Utting relates to the key challenge of testing, namely, how to select the tests that are most likely to expose failures in the system, since the set of possible tests can be huge or infinite, and only a small part of them can be can afforded to perform. This aspect requires knowledge about the system to guess which sets of inputs are likely to produce the same behaviour, that is called uniformity assumption, and which are likely to produce different behaviours. Thus, the expertise of a skilled tester is important here. There are many informal strategies, that can help in deciding which tests are likely to be more effective and some of these strategies are the basis of the test selection algorithms in the model-based testing tools and will be covered in Section 4.

And the last concept making a sense in the above given definition is “**expected**” that is related to so-called “oracle problem” since after each test execution, a decision on whether the observed behaviour of the system was a failure or not should be made. This problem is often solved via manual inspection of the test input; but for efficient and repeatable testing, it must be automated. This can be achieved by model-based testing, by automating the generation of oracles and the choice of test inputs.

In practice, most of the approaches test the system against a set of test cases without going in depth of the complete testing problem due to its impossibility in the general case (the most notable researches dealing with this issue are presented by authors of (Peleska et al., 2017)). This research deals with this issue as well.

3 RATIONALE FOR THE RESEARCH

The research reveals the topicality of software testing and challenges, which widely occurs in software development practice as the main program quality assurance method:

- The solutions proposed by the testing theory fail to meet the requirements of the practice (real-world). The old paradigm of testing has shifted from error/ fault-finding to the new one – the testing goal is to develop reliable programs that can be applied to real IS without risks to business and government management;
- Testing practice in many cases (in Latvia but not only) is still limited to executing manually developed tests without analysis of the complete testing. As a result, there are still cases when program faults cause significant financial losses and failures results in the government management.

The idea of the proposed solution is as follows: first, the specification of the information/ data processing system to be developed or tested is created in a language of a high level of abstraction. It contains the concepts of data object and conditions, where:

- (a) Data objects describe real-world objects that the IS accumulates data or more precisely - input messages;
- (b) The conditions describe the requirements that must be met by the values of the attributes of the data object in order to consider the data object as correct.

Since one of the main functions of IS is to accumulate and process data [objects], first, it is necessary to check that the data objects entered in the system are correct, that is achieved by checking the correctness of values of data objects by applying defined conditions on them. The correct data objects can be entered and stored in the database, however, if the incorrect ones are detected, the data owner is notified about them, which allows data correction and repeated input into the system.

Moreover, the checking of data object must be done on at least two levels – syntactic and contextual or semantic control. Syntactic control checks the conformity of entered attribute values of a data object to the attribute values syntax. Contextual control checks the attributes of data object against other data objects (Nikiforova & Bicevskis, 2019).

The first idea of the proposed solution is to compare the correspondence between data objects are

entered and those stored in the database against each other, in other words, whether the entered data is correctly allocated in the database. These checks are described in the specification and are not related to implementation in a particular programming environment. This idea is close enough to the use of OCL, which, for a particular data storage model (the relational model), proposes a description of the allowable values for data stored in a database. However, this approach is associated with a specific data storage model and is not related to an IS specification.

The second idea is to propose IS complete testing capability. This should be achieved by generating test cases using conditions on data object attributes, that would cover all cases of correct and incorrect behaviour of the data. This idea is close enough to one of the criteria for complete testing – the testing of all input data conditions. The use of concept of a data object allows to generate test cases to test all conditions in a constructive way.

The main idea of testing improvement proposed by the initiated research is close enough to ideas of model-based testing. First, a testing model that will be used to generate tests is created. Testing the operation of the programs on these tests ensures that the programs work correctly according to the created testing model. The main challenge of use of model-based testing is to find such a testing model that would explicitly/ adequately describe what the program needs to do. Therefore, the next section is dedicated to model-based testing, discussing both, main concepts, and the choice of components for their further use in implementation of the proposed solution.

4 MODEL-BASED TESTING

4.1 Model-based Testing Process

The model-based testing (MBT) process can be divided into 5 interconnected phases: (1) model the system under test (SUT) and/ or its environment; (2) generate abstract tests from the model; (3) concretize the abstract tests to make them executable; (4) execute the tests on the SUT and assign verdict; (5) analyse the test results ((Muniz et al., 2015), (Schieferdecker, 2012), (Utting & Legard, 2010), (Utting et al., 2012), (Zander et al., 2017) etc.). 4th and 5th phases are a part of any testing process, even manual, however, steps from 2 to 4 sometimes are

merged into one step, for instance, in case of online model-based testing.

The presented research follows this model, allowing just minor changes. Let's briefly discuss every MBT phase and its key points.

By the 1st step and **creation of the model** is meant the creation of an abstract model of the system is going to be tested. This model is an abstract model since it should be smaller and simpler than the system itself, focusing on the key aspects only. When the model is designed, it is recommended to check its consistency and behaviour by using appropriate tools. This step is often skipped as too resource-consuming since errors are usually detected at later stages. The aim of this step is to link model elements such as states, transitions, and decisions to the requirements to ensure bidirectional traceability between the requirements and the model and later to the generated test cases and test results. The test models must be precise and complete enough to allow automated derivation of tests from them.

As for model, the presented study proposes to use a data quality model containing 3 components: (1) descriptions of data objects, (2) quality requirements for data objects, and (3) a process for evaluating data quality. A detailed description of this model is available in ((Nikiforova & Bicevskis, 2019) and (Nikiforova et al., 2020).

The 2nd step, namely, **generation of abstract tests from the model**, is related to the choice of some test selection criteria, to determine which tests should be generated from the model. **Selection criteria** is vital in order to limit the number of tests (Schieferdecker, 2012). One of the most traditional criteria is structural-model coverage or model coverage criterion, however Muniz with co-authors (2015) highlight such criteria as test purpose, similarity of paths, weight similarity of paths, most probable path and minimum probability of path. The main output of this step is a set of abstract tests, which are sequences of operations from the model. Since the model uses a simplified view of the SUT, these abstract tests lack some of the detail needed by the SUT and are not directly executable.

The proposed solution uses the concept of data quality requirements that are formulated using flowcharts (using graphical domain specific language (DSL)). Their structure and nature will be discussed in more detail in Section 5. As for selection criteria, the proposed solution uses the combination of test case definition and requirements coverage criteria as follows from the previous section (according to classification described in (Zander et al., 2017)).

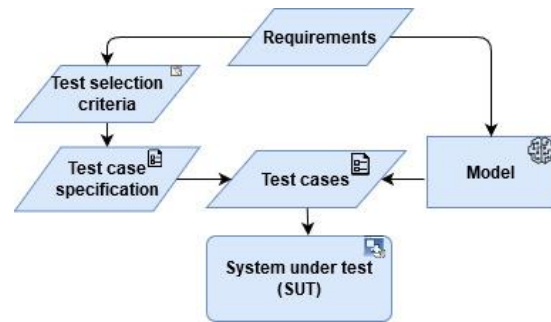


Figure 1: MBT process.

The 3rd step supposes **transformation of the abstract tests into executable concrete tests** which may be done by a transformation tool, which uses various templates and mappings to translate each abstract test case into an executable test script or by writing some adaptor code wrapping around the SUT and implementing each abstract operation in terms of the lower-level SUT facilities. The goal of this step is to add the low-level SUT details that were not mentioned in the abstract model, thus, filling the gap between the abstract tests and the concrete SUT. The nature of transformation, namely, automated, semi-automated or manual, is under discussion and all three options have their own audience.

A given study supposes symbolic execution of data quality requirements (its suitability for complete testing is discussed in (Peleska et al., 2017)). For each case of the requirements, the conditions of the requirements are established which, when resolved, result in specific tests which further test the system.

The 4th step is to **execute the concrete tests on SUT**, while the 5th step is to **analyse the results of the test executions and take corrective action**. For each test that reports a failure, the fault that caused that failure must be detected. This is similar to traditional test analysis process, therefore, won't be discussed.

The general idea of MBT process which the current research follows is shown in the Figure 1.

Since the model is the central object of MBT and the initiated research, the next subsection is devoted to an overview of models, basic concepts that need to be taken into an account, existing approaches, their popularity etc. This discussion will result by the choice of the approach that will be used in the further research and an implementation of the proposed idea.

4.2 Modelling Approaches of MBT

Model-based testing requires an accurate model, written in formal modelling notation that has precise

semantics. One of the key aspects that are generally taken into account, developing MBT solution, is that models must be small in relation to the size of the system that is tested. It is important to guarantee that created models are (a) not too costly to produce, but (b) detailed enough to describe the characteristics that should be tested (is in line with (Utting et al., 2012)).

The development of models is one of the widely discussed topics since this is one of the most crucial aspects in MBT. Since the design of the system is almost always related to the creation of number models, the question about their reuse often reveals. However, in spite of the existence of some models that could be reused, this option is not the best one for several reasons, and the most critical are: (a) usually models that were created by system developers have too many details, most of which are not needed for testing; (b) development models rarely describe the SUT dynamic behaviour in enough detail for test generation. In other words, they are rarely abstract and precise enough for the test generation purposes. Moreover, not always developed systems have model at all (Brahim et al., 2019).

So, the best options for the given purpose are (1) to create a model by yourself or (2) to create it, using a high-level class diagram if it is available, supplying it with the behavioural details. Both options are acceptable in the scope of the proposed solution, however, the basic idea is to create a model by yourself, but the use of already existing high-level class diagram isn't denied. In other words, the choice is up to the tester.

As for a technique that should be used creating models, same as in the case of MDA (model-driven architecture), UML class diagram is one of the most traditional and common options for such a task and is often considered as a standard (corresponds with (Utting & Legeard, 2010) (Muniz et al., 2015) and (Felderer et al., 2010)). According to (Schieferdecker, 2012), UML is aimed to formalize the points of control and observation of the SUT, its expected behaviour, the entities associated with the test, and test data for various test configurations. However, neither UML, nor an informal use case diagram by itself are not precise or detailed enough for MBT, since the description of the dynamic behaviour of the system can't be achieved using traditional UML (without extensions). UML model that would be suitable for MBT requires details about the behaviour of the methods in the class diagram that can be achieved by using OCL postconditions or state machine diagrams. However, this significantly complicates models and requires significant changes

by a skilled person every time the necessity for changes occurs.

Considering existing limitations of UML, one of the most common solutions is to search for another technique. Since two main goals of the models involved, namely, small size of the model and its level of detail, can be conflicted, it is an important engineering task to decide (a) which characteristics of the system should be modelled to satisfy the test objectives, (b) how much detail is useful, and (c) which modelling notation can express those characteristics most naturally.

One of the studies devoted to the analysis of models used for the test generation is (Dias Neto et al., 2007), in which MBT approaches were divided into UML and non-UML models. Authors have collected 406 papers and divided them into 6 categories. Since 2 of 6 categories are out of scope of this research, they are ignored, recalculating provided numbers. Thus, four groups of existing models were analysed, taking into account such aspects as MBT approach, namely, UML or non-UML, and the perspective from which the model represents information. Results of the analysis demonstrate that 46% researchers give their preference to model representing information from software requirements and is described using any non-UML diagrams, while 12,9% uses UML diagrams, 30,7% prefer to develop model representing information from software internal structure and is described using any non-UML notation, while 10,4% uses UML. This demonstrates that UML is less popular in MBT since both categories/ perspectives unify approach used in the proposed solutions. As for the second aspect, the preference is given to model representing information from software requirements, that for both, UML and non-UML approaches, is in force. This can be easily explained by the initial purpose of MBT – to represent information from software requirements only.

To sum up, most of the developers choose non-UML approaches, and this research is not an exception. As for non-UML technique, there are no one common standard to be used ((Utting & Legeard, 2010), (Dias Neto et al., 2007)), therefore, developers choose the technique by themselves, considering pros and cons depending on the specific case.

5 BASICS OF THE PROPOSED IDEA

The proposed solution follows principles of MBT but is not an MBT tool in its common sense.

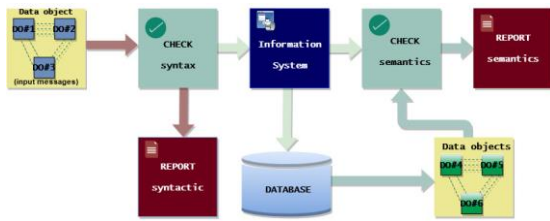


Figure 2: The architecture of the proposed idea.

Its primary task is to test a specific part of an IS – the correctness of input messages which are inserted and their correct allocation in database. This is just one but nevertheless one of the main tasks of IS which is followed by a various different tasks and scenarios which depends on the stored data.

An architecture of the proposed solution is demonstrated in Figure 2.

The most significant change of the MBT process is related to the 1st phase, since only part of the SUT is modelled by creating data objects' and quality specifications diagrams. The test generation step takes place after input is received – when the data objects and conditions for the input message are defined, supplying them with data objects retrieved from the database and the conditions that apply for these data objects. This means that we mainly need a model that would represent all entities of the SUT and their relationships. Each time testing is carried out, only a part of the modelled system can be used in order to improve the performance of the solution, leaving only those entities that are tested and ignoring other (corresponds to the principle of abstraction (Mellor et al., 2004)).

As mentioned earlier, test generation should only be done after selection criteria is chosen. Various options are possible in the way of selection criteria. However, since we mainly discuss one very specific purpose of testing, we limit the other components to this specific purpose – (1) testing whether data to be entered is correct and (2) whether they are allocated in the database correctly (without contradictions in relation to internal constraints). Therefore, as mentioned earlier, a combination of test case definition and requirements coverage criteria seems to be an appropriate choice.

The proposed idea is close enough to the black-box testing, since we can have any knowledge about how the SUT is implemented or its internal behaviour, and we base further steps on its specification only (Muniz et al., 2015). However, since it highly dependent on the model – if the software specification lacks any possible behaviours, the generated test cases will not cover all possible SUT behaviours, as correct model as possible is

required, as well as additional mechanism that will ensure the most complete testing. Therefore, the next subsection is devoted to the choice of the model.

5.1 MBT Modelling Language

In scope of the given research, following requirements were formulated: the model must be (a) concise ensuring it does not take too long to write and easy to validate with respect to the requirements and (b) precise enough since complete testing option is under consideration. Instead of UML, which is characterized by the high number of cons, the preference was given to domain specific language (DSL) creating flowchart-based diagrams. Flowcharts belongs to behavioural models same as Decision Tables, Finite State Machines, Petri nets, [Swim Lane] Event-Driven Petri Nets, Statecharts, UML (use cases and activity charts), and BPMN, while traditional UML belongs to another group - structural models (Jorgensen, 2017).

The proposed solution requires performing the modelling task twice: for input and output data. Both models follow the same principles. Since according to (Nikiforova et al., 2020), the previously proposed DSL (for definition of data quality model) syntax and semantics are easily applicable to the new IS, as well as reusable if necessary, there is no need to develop a new DSL - the previously proposed graphical DSL can be reused. Graphical models are preferred for several reasons. Firstly, models can be used as a communication tool (Mellor et al., 2004), improving the readability of information since graphical representation in models is perceived better by readers than textual representation. Visual information is easier and faster to read and to modify. Moreover, requirements defined in terms of DSL are precise enough (Cunha et al., 2019), therefore complete testing seems to be an achievable goal (in line with (Peleska et al., 2017), (Hübner, 2017)). At the same time, the structure of flowchart is simple. For each condition model, each chart consists of vertices and arcs where (a) the vertices indicated by mnemonic graphic symbols represent actions with data, (b) the arcs connect the vertices, indicating the sequence of actions that must be performed in order to test data (Nikiforova, 2018). The charts also include an element for preparing error reports that are designed to record problems, i.e. creating an enforcement protocol that records data that do not meet conditions. The resulting execution protocols are then used to correct the data. Charts allows users/testers to define a data object and corresponding data that will be tested, requirements that should be met by

the data. Describing the requirements in this way excludes the need to describe the requirements in textual form, which may be interpreted differently.

In addition, diagrams can be transformed as soon as changes or new details appear. This corresponds to (Tretmans et al., 2010), according to which MBT should also be able to deal with an incomplete real world in which requirements are never fulfilled and are constantly evolving. Authors consider this as one of the most vital limitations of the existing solutions. Changes can be introduced by several users (if several users are involved in diagrams creation), and they can also be reused.

However, in addition to DSL in the future works we will consider the use of OCL, proposing a comparison of both approaches. In accordance with (Abbers et al., 2009), OCL rules check the static semantics of the models and can be used to describe constraints that are specific to the domain, modelling language, modelling process, etc. However, it is still not clear whether OCL can be used for checking the dynamic semantics of the models.

5.2 Data Object and Requirements

According to Section 2, the specification of the information/ data processing system to be developed is created in a language of a high level of abstraction, that contains the concepts of (a) data object and (b) conditions or requirements. These concepts were previously presented in a series of researches, including (Nikiforova & Bicevskis, 2019), (Nikiforova et al., 2020), therefore, just main concepts will be presented here, highlighting their main aspects in the scope of the given solution.

As it was mentioned, data objects describe real-world objects that the IS accumulates data or, more precisely, input messages. The use of data object allows to limit the number of parameters that should be analysed, thus leaving only those parameters that need to be tested. Several interconnected data objects can be created as well, thus allowing to perform contextual checks (Nikiforova & Bicevskis, 2019), that become necessary performing the test against data stored in the database. While previous researches, including the concept of a data object, understand it as a set of parameter values that characterize any real-life object (Nikiforova, 2018), this research mainly deals with input messages and objects stored in an IS or database. In addition, the nature of the data object allows to process both, structured and semi-structured data, thus, the proposed solution does not meet the restrictions on the type of data to be tested.

Requirements or conditions are defined for previously defined data object. They are intended to describe the requirements that must be met by the values of attributes of a data object in order to consider data object as correct, i.e. without defects. Requirements regarding attributes of a data object are used to prepare/ generate test cases, which would cover all correct and incorrect inputs.

As in (Nikiforova et al., 2020) created models follow principles defined by Mellor (2004): abstraction and classification. By abstraction Mellor understands “ignoring information that is not of interest in a particular context” (in line with MBT principles for model of the SUT). In the presented approach, it is achieved by using data object exclusively with the parameters representing real objects that are of interest for specific users. By classification Mellor means “grouping important information based on common properties”. This principle is partially followed when grouping quality conditions for each parameter involved in check/ test. Models are machine-readable. In addition, they can be easily updated and reused, that is in line with (Kleppe, 2003), i.e. a good practice for diagrams.

Positive aspects of the proposed components were already discussed in detail in (Bicevskis et al., 2019) and (Nikiforova et al., 2020), as well as demonstrated in a series of articles, therefore, such model and its components seems to be an appropriate choice for the proposed solution.

6 CONCLUSIONS

The paper is a first step to the new solution that [hopefully] will improve existing testing approaches. The solution proposes a new criterion of a complete testing verifying the correctness of all input data in the retention database with tests containing all possible conditions for input data values.

The “black box” testing methods are based on the verification of all requirements formulated for the program. If the requirements are formulated in a natural language, different interpretations, misunderstandings and inaccuracies are possible. If the requirements are expressed in a precise manner offered by the use of the data quality model, the complete testing according to the formulated requirements become realistic. The study proposes to develop this approach.

In the future we are planning not only to implement described idea but also to approbate it on the real system of e-scooters that become the more popular not only in Latvia but also in other countries.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the research project "Competence Centre of Information and Communication Technologies" of EU Structural funds, contract No. 1.2.1.1/18/A/003 signed between IT Competence Centre and Central Finance and Contracting Agency, Research No. 1.7 "The use of business process models for full functional testing of information systems".

REFERENCES

- Abbors, F., Truscan, D. & Lilius, J. (2009). *Tracing requirements in a model-based testing approach*. In First International Conference on Advances in System Testing and Validation Lifecycle (pp. 123-128). IEEE.
- Bicevskis, J., Nikiforova, A., Bicevska, Z., Oditis, I. & Karnitis, G. (2019). *A Step towards a Data Quality Theory*. In 2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS) (pp. 303-308). IEEE, DOI: 10.1109/SNAMS.2019.8931867.
- Brahim, A., Ferhat, R. & Zurfluh, G. (2019). *MDA Process to Extract the Data Model from Document-oriented NoSQL Database*. In Proceedings of the 21st International Conference on Enterprise Information Systems – Vol. 2: ICEIS, p. 141-148. DOI: 10.5220/0007676201410148
- Cunha, A., Fernandes, S. & Magalhães, A. (2019). *Integrating SPL and MDD to Improve the Development of Student Information Systems*. In Proceedings of the 21st International Conference on Enterprise Information Systems - Volume 2: ICEIS, ISBN 978-989-758-372-8, pages 197-204. DOI: 10.5220/0007711201970204
- Dias Neto, A. C., Subramanyan, R., Vieira, M. & Travassos, G. H. (2007). *A survey on model-based testing approaches: a systematic review*. In Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies (pp. 31-36). ACM.
- Felderer, M., Chimiak-Opoka, J., & Breu, R. (2010). *Model-Driven System Testing of Service Oriented Systems-A Standard-Aligned Approach Based on Independent System and Test Models*. In International Conference on Enterprise Information Systems (Vol. 2, pp. 428-435). SCITEPRESS.
- Hübner, F., Huang, W. L., & Peleska, J. (2019). *Experimental evaluation of a novel equivalence class partition testing strategy*. Software & Systems Modeling, 18(1), 423-443.
- IEEE Computer Society Professional Practices Committee. (2004). *SWEBOOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society.
- Jorgensen, P. C. (2017). *The craft of Model-Based testing*. Auerbach Publications.
- Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA explained. The practice and promise of the model driven architecture*. Boston Pearson Education, 1-31.
- Mellor, S. J., Scott, K., Uhl, A. & Weise, D. (2004). *MDA distilled: principles of model-driven architecture*. Addison-Wesley Professional.
- Muniz, L. L., Netto, U. S. & Maia, P. H. M. (2015). *A Model-based Testing Tool for Functional and Statistical Testing*. In Proceedings of the 17th International Conference on Enterprise Information Systems (ICEIS), p. 404-411. DOI: 10.5220/0005398604040411
- Nikiforova, A. & Bicevskis, J. (2019). *An Extended Data Object-driven Approach to Data Quality Evaluation: Contextual Data Quality Analysis*. In Proceedings of the 21st International Conference on Enterprise Information Systems (ICEIS), Vol. 2, p. 274-281. DOI: 10.5220/0007838602740281
- Nikiforova, A., Bicevskis, J., Bicevska, Z. & Oditis, I. (2020). *User-Oriented Approach to Data Quality Evaluation*. Journal of Universal Computer Science, 26(1), 107-126.
- Nikiforova, A. (2018). *Open Data Quality Evaluation: A Comparative Analysis of Open Data in Latvia*. Baltic Journal of Modern Computing, 6(4), 363-386.
- Olsen, K., Parveen, T., Black, R., Friedenber, D., Zakaria, E., Hamburg, M., McKay, J., Walsh, M., Posthuma, M., Smith, M., Smilgin, R., Ulrich, S. & Toms S. (2018). *Certified tester foundation level syllabus*. Journal of International Software Testing Qualifications Board, available: <https://isqi.org/img/cms/ISTQB/syllabuses/C-TFL-Syllabus-2018-GA.PDF>
- Peleska, J., Huang, W. L., & Hübner, F. (2017). *Complete Model-based Testing*. Test, Analyse und Verifikation von Software-gestern, heute, morgen, 81-92.
- Schieferdecker, I. (2012). *Model-based testing*. IEEE software, 29(1), 14.
- Tretmans, J., Prester, F., Helle, P., & Schamai, W. (2010). *Model-based testing 2010: Short abstracts*. Electronic Notes in Theoretical Computer Science, 264(3), 85-99.
- Utting, M., Pretschner, A. & Legeard, B. (2012). *A taxonomy of model-based testing approaches*. Software Testing, Verification and Reliability, 22(5), 297-312.
- Utting, M., & Legeard, B. (2010). *Practical model-based testing: a tools approach*. Elsevier.
- Zander, J., Schieferdecker, I., & Mosterman, P. J. (Eds.). (2017). *Model-based testing for embedded systems*. CRC press.