Scientific Journal of Riga Technical University
Computer Science. Applied Computer Systems
2011
Volume 47

Model-Driven Testing Approach Based on UML Sequence Diagram

Jurijs Grigorjevs, *Riga Technical University*

***Abstract* - This paper is devoted to model-driven testing technique of testing model generation from UML sequence diagram to UML Testing profile. The article discusses principles of model transformation and shows practical approach to specific model generation from another one. The idea of such transformation is to generate test cases from sequence diagram of the system. Described transformation rules are focused on timing aspect verification and the goal of generated test cases is to cover successful as well as unsuccessful system behaviour. Presented example of model transformation is based on a sequence diagram XMI representation, which firstly is pre-processed and moved into data base structures, and then transformation rules applied to generate testing model from data describing sequence diagram.**

Keywords: **model-driven testing, sequence diagram, transformation**

I. Introduction

Automation in software development becomes more and more popular. Program code generation is known and used for many years, but still automation of full development cycle is under research and is on its way to replace manual operations in this cycle. MDA [1] standardized principles, available development environments and tools stimulate such movement to automation. Similar activities are in process also in the area of testing. More than 5 years passed since UML provided Testing profile for test system management, but still there are no common methods of automated test case generation from a system model.

Testing profile standardization started movement to automated test case generation from UML models. During these years new methods of test model generation have been presented and most of them are based on principles described in [2]. Same high level concepts have been used in [3], where a practical model-driven testing method of embedded systems is presented. Described solution is focused on system functionality verification with different input data and on analysis of processing results. The goal of such approach is to provide better coverage with possible input data and do not focus on system's non-functional features and abnormal cases.

In this article a method with practical example of test cases generation for timing aspects verification of behavior is described. The presented method is based on one of the most frequently used diagrams for behavior modeling – UML sequence diagram and UML Testing profile. The idea of the method is to maximally use standardized models, available standards and environments and to develop a framework for automated test case generation based on UML diagrams. To provide flexible future use, the method allows dynamical definition of test case generation rules, which could be appended and customized. In this paper the author presents results of method approbation with timing test cases generation from UML sequence diagram.

II. Principles of Model-Driven Approach in the Context of System Testing

Generally, model-driven testing defines test case development strategy using the model of the system under test [5]. This means that using abstract functional model of the software, which represents also functional as well as non-functional requirements of the system, sets of test data, preconditions and test exit criteria are created. There are two different ways to implement test cases: manual and automated. In manual test case generation there should be at least one testing engineer, who analyzes given abstract functional model of the system and creates test cases. Test cases could be written in the native language as well as using some formal definition. Such test case generation technique is widely used and can be combined with a non-model-driven testing, for example based on requirements document.

Automated test case generation becomes more and more popular. There are 2 major reasons for that. The first reason is to cut time-to-market and to deliver products of higher quality faster [6]. The popular idea to start testing earlier is going to be realized in model-driven testing. Analysis phase produces models of the system, describing its functional and technical aspects and including previously prepared requirements. This means immediate starting of the testing process without waiting for programming activities to start. The second major reason to use model-driven testing is compliance to principles of MDA, where models (with some specific formal OCL restrictions) are the only possible system specification documents. In this case model transformation and generalization is compliant to classical testing V-model [5], where the main idea of the model is not to show appropriate related activities, but to represent program abstraction level.

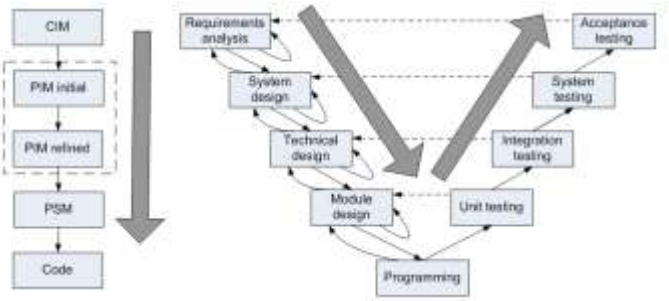

Fig. 1. MDA model transformation comparing to testing V-model [9]

Figure 1 presents the compliance between model transformations under MDA and V-model of testing. MDA principles specify models on different abstraction levels. The model with the highest abstraction level is CIM (Computation independent model), which represents system requirements. In MDA specification CIM requirements should be traceable to the PIM and PSM constructs that implement them and vice versa [4]. The next Platform independent model (PIM) describes the system and its structure, but does not show details of its use. PIM does not show any platform specific features and is used to specify general systems functional principles. The Platform specific model (PSM) is produced by transformation from PIM model. If PSM model contains all necessary information needed for the system implementation, the program code could be generated.

Timing details of the system and its component behavior are specified on PSM level during module design. Such details usually do not cover functional requirements and are subject of unit testing. This means that proposed approach for timing details verification is a part of model-based unit testing.

A. Modeling artifacts in the context of testing

Models are the primary artifacts during model-driven development. Model consists of sets of elements that describe some physical, abstract, or hypothetical reality. UML is the central component of MDA and is used for presentation of system model at different levels of abstraction. It can be appended with formal notations written in OCL. A metamodel is a model of a modeling language. It defines the structure, semantics, and constraints for a family of models. A model is captured by a particular metamodel. For example, a model that employs UML diagrams is captured by the UML metamodel, which describes how UML models can be structured, the elements they can contain, and the properties those elements exhibit. In turn, a metamodel may describe some properties of a particular platform, not only the UML, while a platform's properties may be described by more than one metamodel.

Due to its nature, test case also can be represented as a model, because it also consists of a set of elements that describe abstract reality, which is a set of preconditions and data inputs for some program functionality as well as a set of expected results for described abstraction. In model transformation test case generation means some set of transformation rules necessary to get destination model from systems model, presented in the form appropriate for test case generation.

A projection of MDA definitions of model, metamodel and model transformations into process of system testing gives us the possibility to define test case generation process in terms of model transformations under main statements of model-driven approach. Before model specification, appropriate metamodel should be defined. Metamodels specify all possible elements and relations in-between of target models. Metamodels are based on Meta Object Facilities (MOF).

In this paper the author describes a source and test case model necessary for system and test case specification. In the following chapters the author deals with system source model based on UML sequence diagram and test case model based on the concepts taken from UML Testing profile. The last profile provides standalone metamodel describing all necessary artifacts mentioned in Testing profile. UML sequence diagram provides specification of system behavior and includes timing aspects. This article focuses on these time constraints and suggests method of test cases generation for their verification.

MDA principles are based on the model transformation, where new models are created from existent models applying special transformation. Transformation between models is made by transformation definition, which is a collection of transformation rules in the form of an unambiguous specifications of the way that (a part of) one model can be used to create (a part of) another model [7]. Transformation rules are presented with predefined transformation definition language. Automated transformation of models can be provided only by special tool, which is able to understand a notation of a source and target models and to apply transformation rules according to their logic to the given source model. The result of such process is a set of destination models. MDA framework described in [7] defines a general scheme for transformations between models. It is applied to a test case generation and the result is shown in Figure 2.

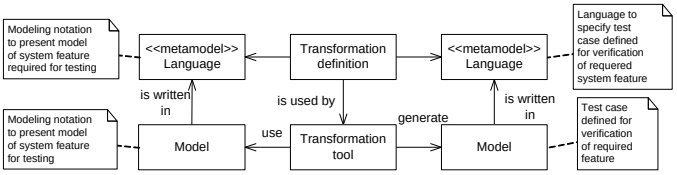

Fig. 2. Application of basic MDA framework for test case generation [9]

Same principles are applied in test case model generation, where several test case models are created from one or more system models.

B. UML sequence diagram

In [10] [11] different modeling notations for real time systems are analyzed with conclusion about UML diagram advantages over other notations for timing details specification. Sequence diagram is one of UML diagrams, which provide timing details definition in system dynamical behavior specification. Based on this analysis UML sequence diagram is chosen as a source model of timing behavior specification. Figure 3 shows metamodel elements of sequence diagram, which cover timing details specification within the diagram.

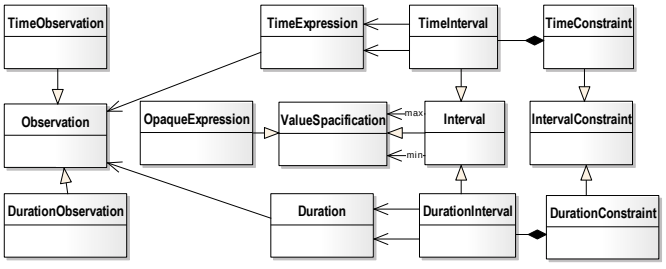

Fig. 3. Timing elements from metamodel of sequence diagram

UML sequence diagram provides definition of 2 types of timing details: time and duration constraint. Time constraint defines time requirements at some point during processing. Duration constraint defines time limitation between 2 points in time, for example between message sending and receiving. Both these constraints are analyzed and processed during model transformation, and behavior of the system, which depends on these constraints, is stored in destination model according to UML Testing profile and transformation rules.

C. UML Testing profile

For testing purposes, OMG has created and standardized UML Testing profile, which is used for test case model development. The profile provides standardized approach for test cases and test suites implementation in model-driven development. It represents MOF-based standalone metamodel for testing aspects modeling. Figure 4 presents MOF-based metamodel of Testing profile [8].

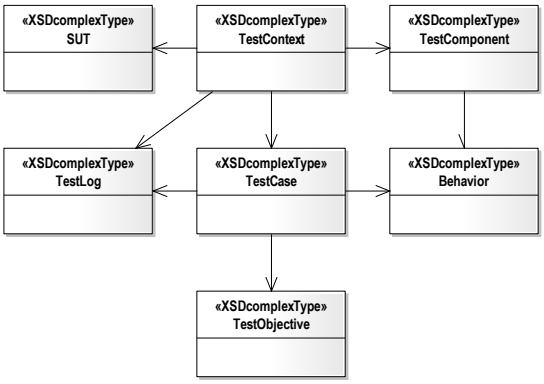

Fig. 4. Testing profile metamodel elements [8]

This profile suggests new approach for testing management, which consists of test case, test data, program behavior and test results. Using Testing profile test cases, the related data could be stored and represented using defined elements of metamodel. The profile describes major classes of testing aspects and suggests general approach to test data management. As it is shown in Figure 4, standardized Testing profile presents general view on testing aspects and specifies high abstraction level of test data management structure. Profile does not specify any system related aspects and leaves this for concrete systems developers.

Method currently does not cover functional aspects of system functionality and does not require profile elements, which are focused on test data definition including input and output data. Also proposed method does not cover automated test case execution and does not use elements related to execution. In the proposed method only main profile elements are used. More details of target model elements are described in the next section.

III. APPROACH FOR AUTOMATED TEST CASE GENERATION

The described approach provides automated test case generation for timing details verification of the system component. For test case generation sequence diagram in XMI format and transformation rules within defined notation are required. The goal of the approach is to automatically generate and store test cases, but automated execution of them is out of the scope of this approach.

A. Data transformation model

As it is mentioned in Section 2, transformation uses UML sequence diagram as a source model and UML Testing profile as a destination model. Figure 5 shows data mapping schema between source and destination models.

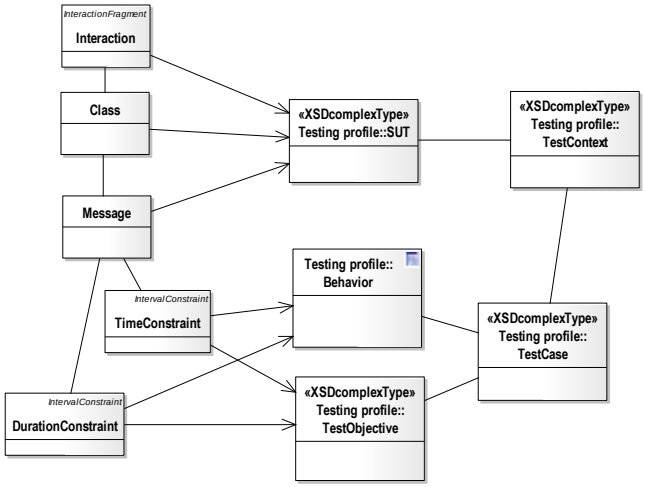

Fig. 5. Conceptual mapping between metamodels

Figure 5 is the simplified conceptual schema of data mapping between sequence diagram and Testing profile elements. Approach uses 5 elements from Testing profile: SUT, Behavior, TestObjective, TestContext and TestCase. SUT element describes system under test and contains information about system components. Behavior element contains timing details definition of the tested components for the concrete test case. TestObjective presents expected result for each test case and at the same time TestObjective could be verified according to TimeConstraint and DurationConstraint of the sequence diagram. TestCase together with related elements describes each independent program execution case. TestContext groups several test cases into sets.

B. Principles and detailed description of the approach

Test cases for specific system at the same time are models of UML Testing profile metamodel. Test cases usually are based on system specification or system model. Figure 6 shows complete approach of automated test case generation, which is based on common model transformation principles of MDA.

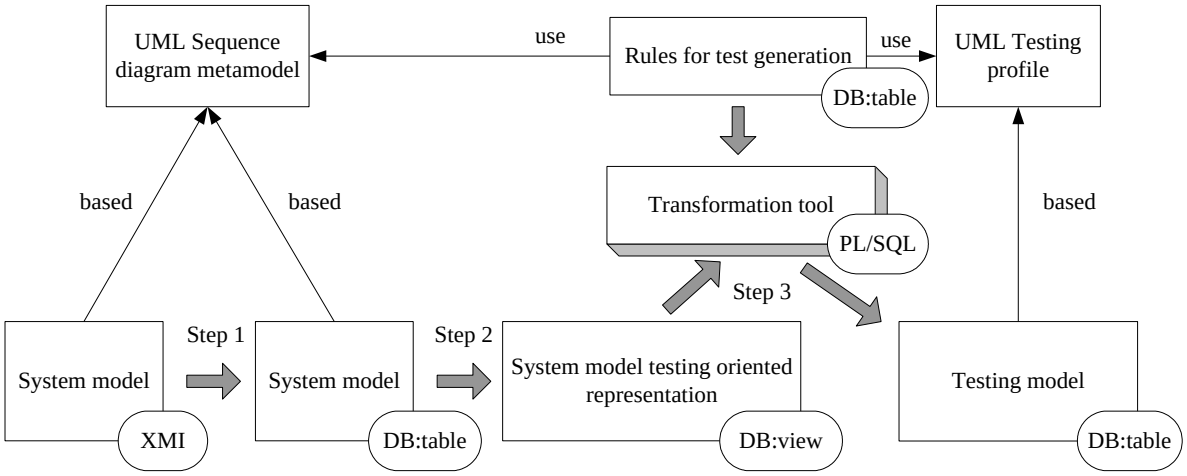

Fig. 6. Complete approach of automated test case generation

The main principle of this approach is to use standardized model presentation, but perform all processing and data generation in database, because data manipulation and management is the purpose of it. Test data, as the result of model transformation, is stored in appropriate tables and linked in-between.

As input data XMI (XML Metadata Interchange, http://www.omg.org/spec/XMI/) representation of system sequence diagram is used. XMI is a standard provided by OMG group for UML model representation in XML format. It means that approach doesn't have dependency on model definition tool and various tools could be used for model development (for example, Enterprise Architect or Eclipse).

In Step 1 only technological representation of system model is changed from XMI to database format. This step is necessary to prepare all model related data for further processing. As the result of this step database structure with inserted data is received. Structure of database is based on UML sequence diagram metamodel and is appended with additional properties from XML file (for example, parent element type and parent element id). In general, for each UML metamodel class appropriate table is created and necessary data are inserted.

Complete structure of UML sequence diagram metamodel is made universal to support various specifications. According to XMI specification, class is linked to message and timing details of the message through 7 layers: Class → Property → Lifeline → OccurenceSpecification → Message → TimeConstraint →TimeExpression → OpaqueExpression → TimeObservation. Such structure is too complicated for transformation rules and it would require definition of technical details in transformation rules. For test case generation purpose system model should be presented in simplified format – views. In this approach Step 2 introduces view for messages presentation and includes all necessary timing details of them. The result of this view is the data ready for transformation.

Transformation tool is developed as PL/SQL procedure with set of additional functions to provide affected system object selection, transformation rules processing and test data generation. In Step 3 transformation is done based on simplified model representation and test data is generated according to defined transformation rules.

C. Notation of the transformation rules

During transformation process each transformation rule sequentially is read from table and processed. Transformation rules are defined and stored in TRANSFORMATION_RULES table according to strictly predefined notation. General format of notation for test case generation transformation rules is the following:

FOR: <*object type*> <*object name*> **WITH:** <*precondition*>
DO: <*operation*> **WITH:** <*behavior def.*> **WHICH:** <*objective*>

<*object type*> - defines type of the object to be processed during transformation. For example, object type MESSAGE could be specified.

<*object name*> - name of the object to be selected for validation and processing. If transformation rule is applied to all objects of the specified type, "*" should be used. Otherwise, exact name of the object of appropriate type should be specified here.

<*precondition*> - precondition section for object selection. Precondition should be written in SQL statement WHERE clause format, which is activated during search of objects to be processed.

<operation> - activity that should be processed with found objects. Currently supported operation is "CREATE TC" – new test case creation for activated rule and found object.

<*behavior definition*> - definition of the timing behavior of affected object in created test case. 2 options are supported for behavior definition:

- static definition – when timing aspects of the object are specified by static data. For example, "durationconstraint=0", "timingconstraint=99" and similar;
- dynamic definition – when timing aspects are defined with the means of data from specification. Timing aspects (duration constraint and timing constraint) according to UML could be specified in time interval format: like interval between min and max (1..7) or

less/more/less or equal/more or equal/equal ("t<=4" or "d>5"). In such case behavior definition could use 2 predefined words "min" and "max" for value specification. For example, "durationconstraint = min +1" or "timingconstraint = max + 2". In case of dynamic definition test case behavior will be automatically calculated based on specification from system model and behavior definition from transformation rule.

<*objective*> - objective of the generated test case. In this section expected result of the generated test case is specified.

Below is given example of transformation rule, the purpose of which is to generate possible test cases for all messages with specified timing constraints for cases, when timing constraint is more than maximal allowed time by 1.

```
FOR: MESSAGE * WITH: timingconstraint is not null
DO: CREATE TC WITH: durationconstraint = max + 1
WHICH: EXPECTED_RESULT = FAILED
```

IV. Practical Outline of Model Construction

Approach approbation is based on a simple sequence diagram with several message transmissions with timing constraints. Figure 7 shows this sequence diagram and time constraints.

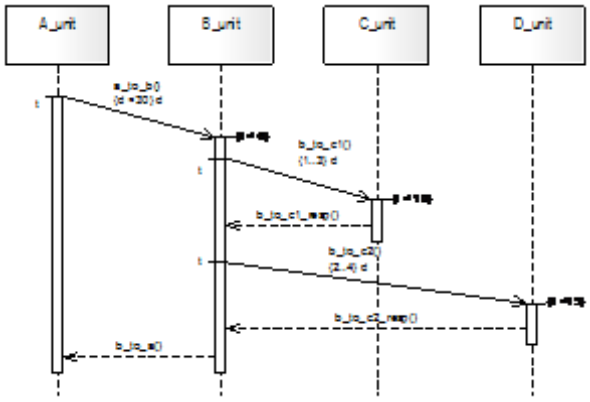

Fig. 7. Example of sequence diagram with time constraints

After diagram export into XMI file and processing in Step 1 of the approach, diagram data is inserted into database tables and is available for further processing.

In Step 2 previously created view contains necessary information about messages to be used in transformation.

```
NAME    SOURCE DESTIN TIMINGC TO DURATIO DO
------- ------ ------ ------- -- ------- --
a_to_b  A_unit B_unit t < 6   t  d <30   d
b_to_c1 B_unit C_unit t < 10  t  1..3    d
b_to_c2 B_unit D_unit t <15   t  2..4    d
```

For transformation the following 3 rules are defined in TRANSFORMATION_RULE table:

```
FOR: MESSAGE * WITH: timingconstraint is not null
DO: CREATE TC WITH: timingconstraint = max + 1
WHICH: EXPECTED_RESULT = FAILED

FOR: MESSAGE b_to_c1 WITH: durationconstraint is
not null DO: CREATE TC WITH: durationconstraint =
max WHICH: EXPECTED_RESULT = SUCCES

FOR: MESSAGE * WITH: timingconstraint is not null
DO: CREATE TC WITH: timingconstraint = max -1
WHICH: EXPECTED_RESULT = SUCCES
```

After transformation processing with these rules we got 7 test cases with the following behaviors and objectives.

```
BEHAVIORDEFINITION                          TESTOBJECTIVE
for MESSAGE b_to_c2  timeconstraint = 16    FAILED
for MESSAGE a_to_b   timeconstraint = 7     FAILED
for MESSAGE b_to_c1  timeconstraint = 11    FAILED
for MESSAGE b_to_c1  duration = 3           SUCCES
for MESSAGE a_to_b   timeconstraint = 5     SUCCES
for MESSAGE b_to_c1  timeconstraint = 9     SUCCES
for MESSAGE b_to_c2  timeconstraint = 14    SUCCES
```

This is the simplest example of test case generation. The idea of this approbation is to show usage of MDA principles in test case generation. These examples cover timing constraints verification according to transformation rules. The result of such generation is the set of test cases of all system components, which satisfy conditions in transformation rules. In such cases test objective is taken from transformation rule, but it also can be generated based on the constraint value.

V. Conclusions

Model-based software development requires appropriate testing techniques. Source for test cases development in such processes is the model. In this paper the author presented approach for test cases development based on UML sequence diagram. This method follows up MDA principles of model transformation. Sequence diagram is used as a source model to describe system dynamic behavior including time constraints. UML testing profile is selected as destination model to define test cases and to them related aspects. During transformation development it was stated that transformation rules for direct transformation from sequence diagram to Testing profile could be complicated and require simplification of this process. For this purpose suggested approach includes addition step of source data preprocessing to simplify them and to make them ready for transformation rules. After such preparation transformation rules could focus only on logical data manipulation.

From technical point of view the described approach is based on XMI standard and is able to process sequence diagram in XMI format. This option allows using a variety of UML diagram development tools. Transformation rules and source models are stored in database and transformation tool is developed in PL/SQL. Transformation results in new data generation, which complies with UML Testing profile definition.

Developed transformation tool has limitations and currently supports only timing details verification from sequence diagram. Structure of the tool is able to extend its functionality and after improvements could support other UML diagrams.

REFERENCES

[1] OMG, Model Driven Architecture [Online]. Available: http://www.omg.org/mda/ [Accessed: Oct. 10, 2010].
[2] P. Baker, Z. R. Dai, J. Grabowski, O. Haugen, I. Schieferdecker, C. Williams, *Model-Driven Testing*, Springer-Verlag, 2010.
[3] J. Zander-Nowicka, *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain*, doctoral thesis, Technical University Berlin, 2008.
[4] S. J. Mellor, K. Scott, A. Uhl, D. Weise, *MDA Distilled: Principles of Model-Driven Architecture*, Addison Wesley Professional, 2004.
[5] A. Spillner, T. Linz, H. Schaefer, *Software Testing Foundations*, Santa Barbara: Rocky Nook Inc., 2007.
[6] G. Engels, B. Guldali, M. Lohmann, Towards Model-Driven Unit Testing [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.62.200 [Accessed: Sep 12, 2010].
[7] A. G. Kleppe, J. Warmer, W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Addison Wesley Professional, 2003.
[8] OMG, UML Testing Profile, V 1.0 [Online]. Available: http://www.omg.org/technology/documents/formal/test_profile.htm [Accessed: Sep 10, 2010].
[9] J. Grigorjevs, O. Nikiforova, "Several Outlines on Model-Driven Approach for Testing of Embedded Systems" in *Scientific Proceedings of Riga Technical University, 5, Computer Science. Applied Computer Systems*, Riga: RTU, 2009. pp. 108-118.
[10] J. Grigorjevs, O. Nikiforova, "Modelling of Non-Functional Requirements of Embedded Systems" in *Scientific Proceedings of 42nd Spring International Conference MOSIS2008*, Ostrava: MARQ, 2008. pp. 13-20.
[11] J. Grigorjevs, O. Nikiforova, "Complience of Popular Modelling Notations to Non-functional Requirements of Embedded Systems" in *Proceedings of the International Scientific Conference Informatics in the Scientific Knowledge 2008*, Varna: University publishing house VFU "Chernorizets Hrabar", 2008. pp. 139-149.

Jurijs Grigorjevs holds a master degree in computer science. In 2005 he graduated Riga Technical University with the thesis related to testing of real-time systems. Since that time he has continued research in testing area and his current research topic is devoted to integration of MDA principles with testing process and automated test case generation.
In 2001 he started testing engineer career at Tieto Corporation and currently is an Integration testing group manager.

Jurijs Grigorjevs. Modeļvadāmā testēšanas pieeja balstoties uz UML secību diagrammu
Raksts ir veltīts modeļvadāmās testēšanas principiem. Autors piedāvā savu pieeju testēšanas gadījumu izstrādei. Modeļvadāmā programmatūras izstrāde paredz programmas koda iegūšanu no sistēmas modeļiem, lietojot transformācijas rīkus un definējot transformācijas likumus. Pašlaik arvien plašāk tiek attīstītas komerciālas un brīvi pieejamas programmatūras, kas nodrošina programmas koda automātisko ģenerēšanu no sistēmas modeļiem. Raksta autors piedāvā savu metodi transformācijas principu pielietošanai ar mērķi automātiski ģenerēt testēšanas gadījumus. Par pētījuma problēmu tika izvēlētas laicīguma īpašības, kas prasa specifiskus testēšanas gadījumus tās verificēšanai. Balstoties uz iepriekšējo pētījumu rezultātiem, par sākotnējo modeli sistēmas modelēšanai tika izvēlēta UML secību diagramma, kas nodrošina laicīguma aspektu definēšanu. UML testēšanas profils, ko piedāvā OMG, tiek lietots beigu modeļa atspoguļošanai, testēšanas gadījumu un ar to saistītu aspektu glabāšanai. Piedāvātā metode paredz secību diagrammas ielasīšanu XMI standarta formātā, ielasīto datu ielādi datu bāzes tabulās, kas atbilst secību diagrammas metamodelim un turpmāko modeļa transformāciju testēšanas modelī. Lai atvieglotu transformācijas likumu rakstīšanu, metode paredz sākotnēja modeļa vienkāršošanu un tā atspoguļošanu skata formā, kas iekapsulē svarīgākās objekta īpatnības. Pētījuma ietvaros ir izstrādāta transformācijas programma, kas ir spējīga lasīt transformācijas likumus un pielietot tās attiecībā pret sākotnēja modeļa datiem. Transformācijas rezultātā tiek aizpildītas datu bāzes tabulas, kas atbilst UML testēšanas profila metamodelim. Raksta beigās autors sniedz piedāvātās metodes aprobāciju ar vienkāršas secību diagrammas palīdzību ar diviem ziņojumiem ar laika ierobežojumiem. Aprobācijas laikā tiek definēti 3 transformācijas likumi, kas rezultātā iegūst 7 testēšanas gadījumus. Aprobācijas rezultāts ļauj secināt, ka piedāvātā metode ir spējīga darboties un ģenerēt testēšanas gadījumus. Savukārt, izstrādātā rīka ierobežotā funkcionalitāte neļaus to lietot pilnvērtīgai testēšanai, bet izvēlētā struktūra atbalsta jaunu transformācijas objektu pielikšanu un transformācijas likumu papildināšanu. Pētījumā gaitā tika konstatēts, ka sākotnējā datu struktūra transformācijai ir pārāk sarežģīta transformācijas likumiem, tāpēc ir nepieciešama datu vienkāršotā reprezentācija. Pētījuma turpinājumā ir nepieciešams nodrošināt sarežģītāku gadījumu apstrādi, ka arī papildināt testējamo funkcionālo apgabalu.

Юрий Григорьев. Метод тестирования, основанный на UML диаграмме последовательностей
Статья посвящена процессу тестирования, основанному на моделях системы, где автор предлагает свой подход для генерации тестовых случаев. Разработка программного обеспечения, основанная на моделях системы, подразумевает использование моделей для автоматической генерации кода посредством использования программы трансформации и правил трансформаций. В данное время всё чаще встречаются коммерческие и свободно доступные программы для автоматической генерации программного кода из моделей системы. В статье автор предлагает свой метод использования аналогичных принципов генерации для формирования тестовых случаев, основываясь на модели системы. В качестве проблемы для данного исследования были выбраны временные ограничения функционирования программ, которые нуждаются в соответствующих тестовых случаях. Основываясь на результатах предыдущих исследований, за исходную модель была выбрана диаграмма последовательностей, которая также обеспечивает поддержку временных ограничений. Конечная модель для формирования тестовых данных основывается на UML профиле для тестирования. Предложенный метод подразумевает зачитывание диаграммы последовательностей в XMI формате, загрузку этих данных в таблицы базы данных, которая соответствует метамодели диаграммы последовательностей, и дальнейшую трансформацию в модель тестирования. Для облегчения написания правил трансформаций метод предусматривает упрощение и отображение изначальной модели в виде представления базы данных. В рамках данного исследования была разработана программа трансформации, позволяющая зачитывание правил трансформации и их дальнейшую обработку. В результате трансформаций заполняются таблицы базы данных, соответствующие формату метамодели UML диаграммы последовательности. В продолжение статьи автор приводит апробацию предложенного метода с помощью простой диаграммы последовательности, где имеются сообщения с временными ограничениями. Во время практической проверки метода предлагаются 3 правила трансформаций, что в результате приводит к генерации 7 тестовых случаев. Результат апробации даёт возможность сделать вывод о дееспособности данного метода и разработанных программ. В свою очередь, функциональные ограничения разработок не позволяют использовать их в полноценных тестах. Однако структура, выбранная при разработке программ, позволяет дополнять их функциональные возможности и развивать в дальнейших исследованиях. Во время данного исследования также было констатировано, что структура исходных данных модели сложна для прямого использования их в трансформации и есть необходимость в их упрощении, что и было сделано во время исследований. В продолжение исследования необходимо обеспечить поддержку более сложных диаграмм последовательностей и дополнить тестируемую область.