Automatic Test Case Generation from Functional Requirements in NDT

Javier Gutiérrez, Gustavo Aragón, Manuel Mejías, Francisco Jose Domínguez Mayo, and Carmen M. Ruiz Cutilla

IWT2 Research Group, University of Seville, Seville, Spain
{javierj,risoto,fjdominguez}@us.es,
{gustavo.aragon,carmen.ruiz}@iwt2.org

Abstract. Navigational Development Techniques (NDT) is a Model-driven framework focused on defining Web requirements and obtaining related artefacts from them by means of transformations. Testing is one of the key elements in a software development process, however NDT neither include models to define artefacts nor transformations to obtain them from requirements. This paper presents how NDT improves with new models and transformations in order to generate test cases.

1 Introduction

Model-Driven Engineering (MDE hereinafter) is a Software Engineering paradigm focused on creating and exploiting domain models [19]. In the last years, this paradigm was used in several domains of Software Engineering providing relevant results.

Web Engineering constitutes one of these domains [5]. Research groups are using MDE for requirements treatment, design, development and several aspects of Web development. This field is commonly named Model-Driven Web Engineering.

However, one of the less treated phases is the testing and validation phase. In the survey presented in [5] relevant conclusions about the suitability of applying MDE in this context are stated. This paper presents the application of MDE in a Web context. It focuses on the first phases of the lifecycle and it studies how functional testing can be deeply improved by means of *early testing*. Thus, this paper analyses an approach that uses the MDE context and illustrates such uses in a concrete environment, NDT approach (Navigational Development Techniques)[6].

NDT was initially defined to deal with Web development requirements, but it has evolved in the last years and nowadays it offers a complete support for the complete lifecycle. NDT covers viability study, requirements treatment, analysis, design, construction or implementation as well as maintenance and test processes. Additionally, it supports a set of processes to bear out project management and quality assurance.

This paper describes how NDT has been extended to incorporate functional system test cases. These test cases verify that the system under test commits the behaviour defined in its functional requirement [12]. NDT models the functional requirements as use cases, thus, both terms will be used as synonyms in this paper.

This paper is organised as follows. Section 2 introduces a motivating example from a real project that enhances us to extend NDT. Then, section 3 cites related work dealing with generating test cases from use cases. Section 4 puts forward how NDT has been extended by means of metamodels and transformations so as to generate test cases from use cases. Finally, Section 5 states the conclusions and ongoing work.

2 A Motivating Example

During 2.008, the IWT2 research group participated in a technological migration of EMASESA information systems. EMASESA is a public company that manages the urban water cycle, providing and ensuring water supply to all citizens in Seville. IWT2 members' collaborative work focused on using NDT for the quality management of the methodological process.

AQUA-Web-Services project (also called AQUA-WS) consists in developing an application of an integrated business system for customer management and involvement in water distribution, cleaning, and net management. The software system was composed of three subsystems, each one representing a legacy system. There was a subsystem for managing the infrastructure of the pipe net, a subsystem for managing clients and another one for managing the whole organization. The total system includes 1.808 functional requirements, containing several scenarios and alternatives in each functional requirement.

The use cases were defined by means of Enterprise Architect tool, linked to an Oracle Database Server and a Subversion repository. This platform enables the parallel work of several teams: developers of the two software factories implied, EMASESA's managers and the group who works in quality assurance.

Use cases were modelled using two techniques: UML Activity Diagrams and text templates. Activities diagrams were modelled according to UML specifications. Text templates were modelled according to the previous work developed by the IWT2 group on functional requirements [6].

The estimate amount of time needed to generate the package structure, elaborate the test case suite that covers all scenarios from the functional requirements, design those test cases in Enterprise Architect and trace them with the functional requirement under test was vast. Estimating 5 minutes to create a test scenario in the modelling tool, the amount of time gained with NDT-Tool was 583 hours (73 days working eight hours a week). This was a big amount of time for a task that was repeated and systematic, so this tool support was proposed.

During the AQUA-WS project improvement teams used a software prototype to produce the test plan. This plan generated about 7,000 test cases from different scenarios of the use case in a few minutes, by repeating the package structure of the functional requirements and adding tracing relations to the functional requirements under test.

3 Related Work

Several approaches in the literature study how to generate functional test cases specifically from a functional requirements model defined as use cases. There are also two surveys analysing the existing literature. The most recent survey, which updates the original survey published in [4], has been published in [5] at the end of 2011. Some specific approaches studied in Escalona's survey are described in next paragraphs.

Ruder's [18] approach starts with functional requirements written in natural language. The result is a set of functional test cases obtained from a coverage criterion based on combinations that support Boolean propositions. Binder's book [1] describes the application of the Category-Partition Method in use cases. Categories are any points in which the behaviour of the use case may be different between two realizations of the use case. This application is named the Extended Use Case Pattern. Finally, Ibrahim et-al. [8] offers a tool, called GenTCase, which generates test cases automatically from a use case diagram enriched with each use case tabular text description.

Frölich et-al. [7] introduces an approach describing how to translate a functional requirement from natural language into a state-chart diagram in a systematic way, as well as how to generate a set of functional test cases from that diagram. Naresh [13] presents an approach dealing with translating a functional requirement from natural language into a flow diagram and performing a path coverage technique to generate test cases. Mogyorodi [10] introduces an approach analysing functional requirements as cause-effect graphs that generate test cases from diagrams. Boddu et-al. [2] presents an approach divided into two blocks: the first one describes a natural language analyzer generating a state machine from functional requirements, and the second one shows how to create test cases from such state machine.

Escalona's survey claims that there is no definitive approach that closes the problem of generating functional text cases automatically in a satisfactory way. Thus, there are some aspects to be improved, for example, the use of standards for inputs and outputs, the application of standards and more formal methods to describe the process itself, the need for empirical results or measuring the possible automation and a profitable tool supporting, among others. Conclusions of Denger's survey goes in the same line.

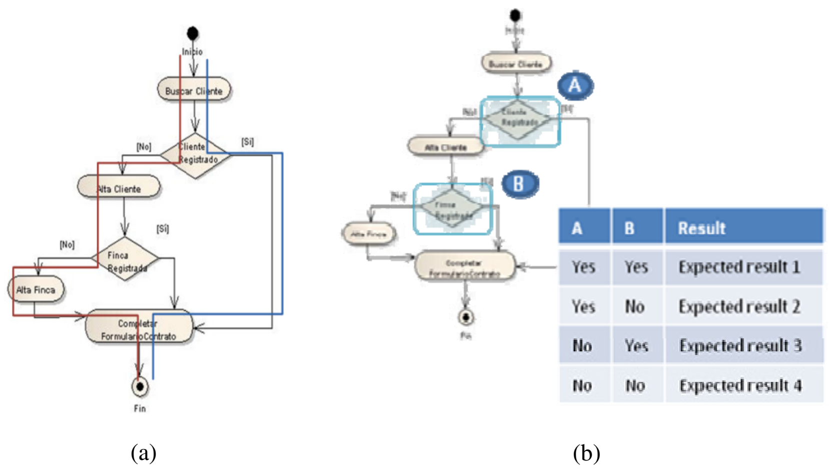
4 Extension of NDT

This section describes the extension of NDT with new metamodels and transformations. Section 4.1 describes the two testing techniques used for generating functional test cases identified in previous work (Section 3). Then, section 4.2 introduces the metamodel selected to define the results of the previous testing techniques. Section 4.3 analyses both testing techniques as a set of relations between previous models and their implementation in QVT code. Finally, section 4.4 describes the implementation of the new transformations in the existing set of supporting tools of NDT.

4.1 Techniques for Test Cases Generation

After mentioning the existing work in the previous section, it is worth mentioning that there are two techniques emerging as the most important ones for generating test cases from use cases: Round-Strip Strategy and Extended Use Cases (names given by Binder in [1]). Both techniques are described in next paragraphs.

The Round-strip strategy consists in applying a classic algorithm of path-finding in a state machine. The behaviour described in a functional requirement may be managed as a graph or as a state machine despite its concrete syntax. Hence, a path searching allows identifying all the different paths through behaviour. Each path will be a scenario designed together with the system. At the same time, each scenario is a potential test case for assessing the right implementation of such scenario in the tested system. Generation of test cases from state-machines is a widely described topic in research literature. Previous section presented several references about this topic in the specific use cases context, like [7], [13] or [2]. Figure 1(a) shows an example of the Round-Strip Strategy using the behaviour of a use case defined as an activity diagram.



A	B	Result
Yes	Yes	Expected result 1
Yes	No	Expected result 2
No	Yes	Expected result 3
No	No	Expected result 4

Fig. 1. Examples of Round-Strip Strategy (a) and Extended Use Cases (b) techniques

The Extended Use Case pattern consists in applying the Category-Partition Method [17] to use cases. The Category-Partition Method is a technique based on identifying categories and partitions to then generate combinations among such partitions (Figure 1(b)). In the context of functional requirements, a category is any point for which the functional requirement defines an alternative behaviour (Figure 1 (b)). Besides, a partition is defined as a subset of the domain of the condition evaluated in the category which decides whether a concrete piece of behaviour is executed or not. Once all categories and partitions are identified, a combination among them is performed and each combination becomes a potential test case. The previous section presented several references about this topic in the specific context of use cases, like [18] or [1]. Figure 1(b) shows an example of the Category-Partition Method (as described in [1]) using the same behaviour as Figure 1(a).

4.2 Metamodels

Due to the Model-driven nature of NDT, the concepts involved in functional test cases should be identified and defined as metamodels. A metamodel defines the concept in terms of its attributes and its relationships with other concepts [19]. Four metamodels were designed. These metamodels are described in next paragraphs.

The first one (Figure 2) defines the necessary elements from functional requirements to generate test cases. The *Subsystem* element represents a package or a container for functional requirements and other related elements (as *SysteActor*).

The key concept in this metamodel is the *FunctionalRequirement* element. The behaviour of a functional requirement is modelled using the elements *Step* and *ExecutionOrder*. The *Step* element models a concrete chunk of behaviour of the functional requirement, such as requesting information or calculating a result. The *ExecutionOrder* element defines the order in which steps are executed. Using a metaphor, the functional requirement may be seen as a finite-state machine (usually called FSM), the steps as states and the execution order as the transition from one state to another one.

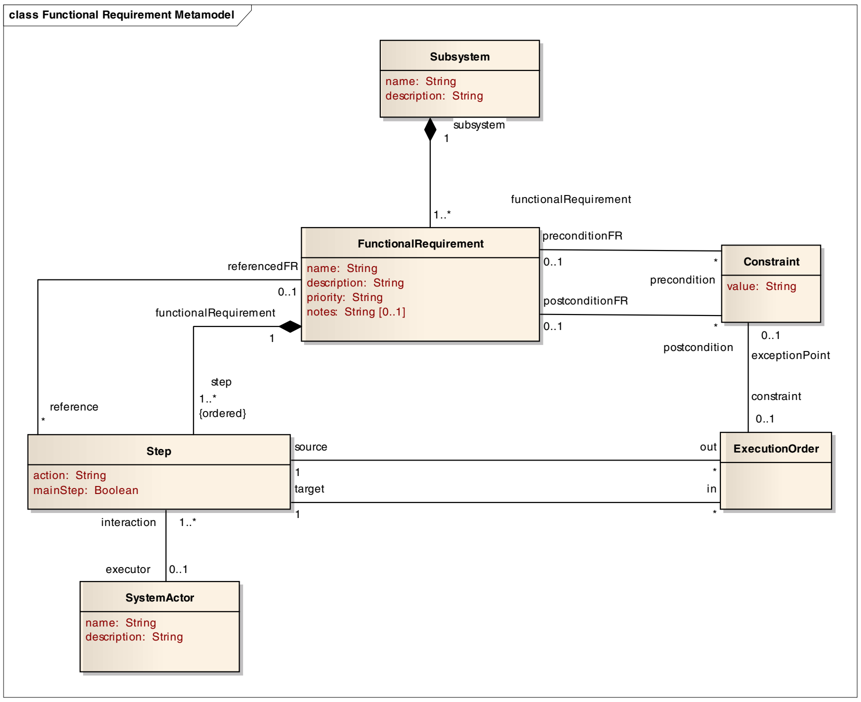


Fig. 2. Metamodel for Functional Requirements

The *SystemActor* element models an external entity that collaborates with the system during the steps performance.

The introduction of additional functional requirements as part of the behaviour of a functional requirement has been considered by using the relation *reference-referencedFR* (from *Step* to *FunctionalRequirement*). This mechanism allows defining the semantic expressed through inclusion and extension relations as defined in UML Use Case metamodel.

The metamodel in Figure 2 directly matches with the functional requirement model defined in NDT. This means that each functional requirement defined with NDT has the concepts exposed in Figure 2, and it may be used with the transformations and tools described in next sections.

The second metamodel (Figure 3) defines the concepts resulting from the Round-Trip technique (Figure 1(a)). Each path is called test scenario (element *TestScenario* in Figure 3) and the traverse steps are classified into actions, (element *ActionFromTestScenario* in Figure 3) when performed by an external actor or into verifications, (element *VerificationFromTestScenario* in Figure 3) when performed by the system and, therefore, it is suitable to introduce a assert during the test.

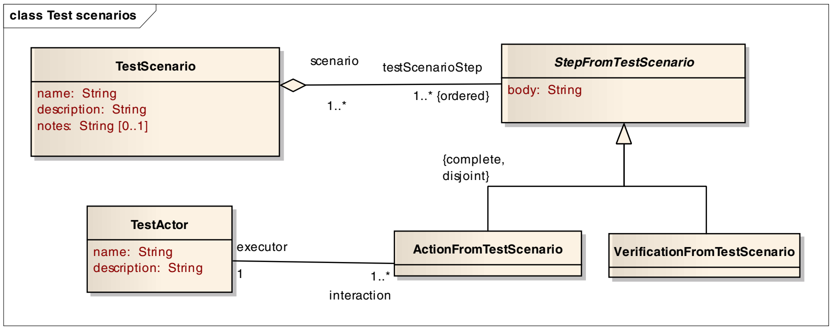


Fig. 3. Metamodel for test scenarios

The third metamodel (Figure 4) defines the concepts resulting from the Category-Partition Method. Categories are modelled by means of the element *OperationalVariable* (as named in [1]) whereas partitions are modelled through the element *Partition.* The element *Instance* points out an evaluation of an operational variable, for example A or B cells in Figure 1(b), and allows distinguishing it from other evaluations of the same operational variable, in case the behaviour of the functional requirement has loops. A *Quantum* element models a value transfer from a partition into an instance. A combination (a row in Figure 1(b)) is modelled using the element *InstanceCombination*.

Finally, the last metamodel introduces artifacts that combine the results of the two previous techniques in the same model. This last metamodel does not introduce any new information. However, it offers linking elements to represent the information through a common artifact (called test case), the steps from a functional requirement as well as a combination of partitions. Figure 5 shows the tracing relation between the four metamodels. Tracing enables knowing which test artifacts have been generated for each functional requirement.

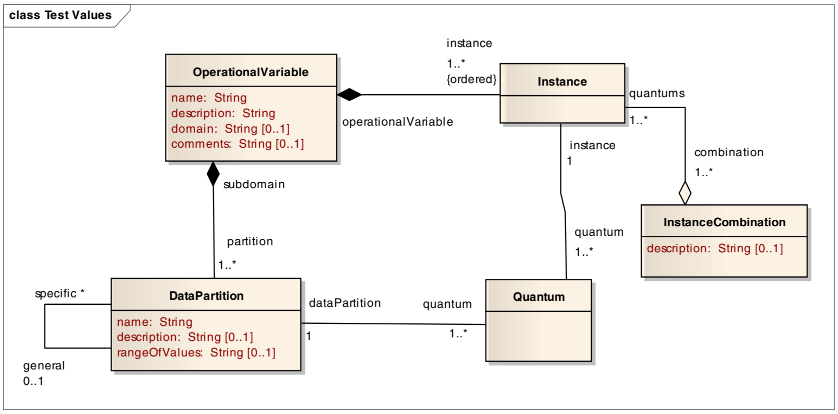


Fig. 4. Metamodel for test values

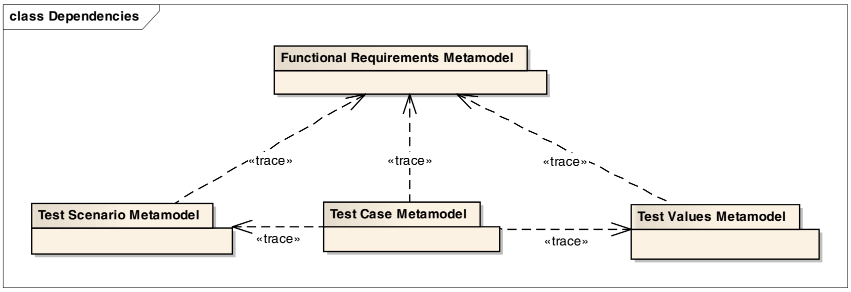


Fig. 5. Tracing relationships among metamodels

Former metamodels have been added to the set of metamodels managed and supported by NDT as part of its MDD development process.

4.3 Transformations and QVT

This section describes how to apply the two techniques presented in Section 2 (Round-Trip and Extended Use Cases) taking the information from functional requirements metamodels (in the previous section) as a source and the information from testing metamodels as a target.

The process of applying both techniques is analysed according to the identification of a set of relations between source concepts (functional requirements) and target concepts (test scenarios and operational variables combinations), as observed in Figure 6. The task of identifying these relations consists in detecting how one target element is built up, for example a test case, by means of the source elements and their information. Next paragraphs provide an overview of the three relations (named T1, T2 and T3 in Figure 6) defined to create test scenarios, combinations of operational variables and test cases from functional requirements.

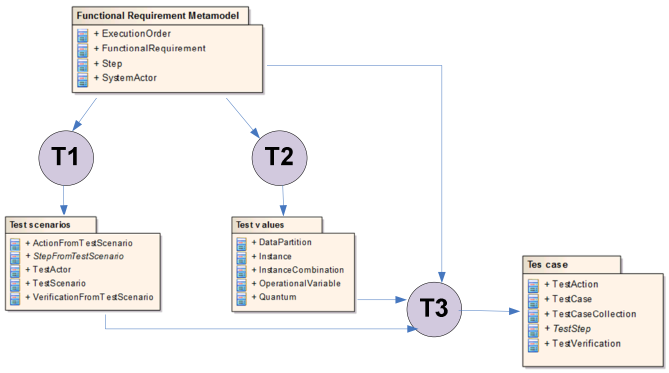


Fig. 6. Transformations among models

Relation T1 involves functional requirements and the Round-Strip strategy. As it was represented in Figure 1(a), the functional requirement behaviour may be modelled as a state-machine, the concept *Step* from Figure 2 models states, and the concept *ExecutionOrder* models transitions. Thus, a classic coverage criterion may be selected to traverse the functional requirement and generate test scenarios. The all-loops criterion, in which all combinations among loops are traversed at least once, is the one selected to extend NDT. Test scenarios steps are generated from all the functional requirements steps. Action (element *ActionFromTestScenario*) and verification (element *VerificationFromTestScenario*) classifications depend on whether there is a relation with a system actor. Finally, test actors are generated from actors, which, due to their attributes are the same ones.

Relation T2 in Figure 6 involves functional requirements and the Category-Partition Method. Operational Variables are created from steps that have more than one output transition (modelled as an ExecutionOrder *element). The outputs of the steps generate the different partition. Again, combinations may be calculated using several criteria, ranged from calculating all possible solutions to calculating only a subset.*

Table 1. Metrics for QVT-Operational code

	T0	T1	T2	T3
Total lines	124	118	290	170
Lines of codes	104	97	238	124
No. of Mappings	1	4	5	3
No. of Helpers	1	2	3	1
No. of Queries	3	2	1	3
No. of Input models	1	1	1	3
No. of Output models	1	1	1	1

Relation T3 (Figure 6) combines both techniques results. Test scenarios and combinations of operational variables merge using test cases.

The relations stated in the previous paragraphs (T1, T2 and T3 from Figure 6) were defined through QVT Operational language as a necessary step to know how to implement the transformation process into an automatic tool. QVT code may be downloaded from [20]. Metrics of QVT code are represented in Table 1 and defined in [16].

Table 1 adds an additional transformation, called T0, not included in Figure 6. This transformation contains common a code used in other transformations. As reference, the Umls2Rdb transformation written in QVT Operational and included in the QVT reference [15] has 65 lines of code, 6 mappings, and 1 query.

4.4 NDT Extension

Nowadays, several companies in Spain work with NDT. This is possible due to the fact that NDT is completely supported by a set of free tools, mainly grouped in NDT-Suite [9]. This suite enables the definition and use of every process and task supported by NDT (Figure 1) and offers relevant resources for quality assurance, management and metrics with the aim of developing software projects. The suite was also extended to implement the first technique for test case generation using activity diagrams as the concrete input for functional requirements, and for the concrete syntax of the test scenarios generated. The implementation of the second technique is still an ongoing work.

However, the MDD perspective allows the concrete notations independency. Thus, the metamodels and transformations defined in previous section may be used out of the scope of NDT. The only request is that the source functional requirements must include the concepts defined in the functional requirements metamodel used as the basis for the process. To remark this independency, a second tool, called MDETest was created. The main differences between this tool and NDT-Suite are that MDETest implements the three target metamodels and it generates the tool uses instances only for metamodels, so that, it does not impose any restrictions on the concrete notations of the functional requirements input. Nowadays, this tool supports activity diagrams such as the syntax for functional requirements, although it does not support any concrete syntax for the output. This tool is also available in [20].

5 Conclusions and Ongoing Work

This paper presents an extension of NDT, based on metamodels and transformations, with the aim of generating test cases from functional requirements. The extension has been tested in several projects and it opens new research lines. Firstly, we have to work in test cases prioritization mechanisms, consisting in giving relevance to functional requirements, as well as in redundant test cases detection. The practice concludes that it continues producing a high number of redundant test cases that the test teams have to detect by hand. One last ongoing work would deal with supporting the semantic of the inclusion and extension relations defined in UML [14] for use cases.

Acknowledgements. *This research has been supported by the Tempros project (TIN2010-20057-C03-02) and Red CaSA (TIN 2010-12312-E) of the Ministerio de Ciencia e Innovación, Spain, and NDTQ-Framework project of the Junta de Andalucía, Spain (TIC-5789).*

References

[1] Binder, R.V.: Testing Object-Oriented Systems. Addison Wesley (1999)

[2] Boddu, R., Guo, L., Mukhopadhyay, S.: RETNA: From Requirements to Testing in Natural Way. In: 12th IEEE International Requirements Engineering, RE 2004 (2004)

[3] Cutilla, C.R., García-García, J.A., Alba, M., Escalona, M.J., Rodríguez-Catalán, L.: Aplicación del paradigma MDE para la generación de pruebas funcionales. In: Experiencia Dentro del Proyecto AQUA-WS, ATSE 2011, Chaves, Portugal (2011)

[4] Denger, C., Medina, M.: Test Case Derived from Requirement Specifications. Fraunhofer IESE Report, Germany (2003)

[5] Escalona, M.J., Gutiérrez, J.J., Mejías, M., Aragón, G., Ramos, I., Torres, J., Domínguez, F.J.: An Overview on Test Generation from Functional Requirements. The Journal of Systems and Software (2011)

[6] Escalona, M.J., Aragón, G.: NDT. A Model-Driven Approach for Web Requirements. IEEE Transaction on Software Engineering 34(3), 370–390 (2008)

[7] Fröhlich, P., Link, J.: Automated Test Case Generation from Dynamic Models. In: Bertino, E. (ed.) ECOOP 2000. LNCS, vol. 1850, pp. 472–491. Springer, Heidelberg (2000)

[8] Ibrahim, R., Saringat, M.Z., Ibrahim, N., Ismail, N.: An Automatic Tool for Generating Test Cases from the System's Requirements. In: 7th International Conference on Computer and Information Technology, Fukushima, Japan (2007)

[9] García-García, J.A., Cutilla, C.R., Escalona, M.J., Alba, M., Torres, J.: NDT-Driver, a Java Tool to Support QVT Transformations for NDT. In: 20th International Conference on Information Systems Development, Edinburgh, Scotland, August 24-26 (2011)

[10] Mogyorodi, G.E.: What Is Requirements-Based Testing? In: 15th Annual Software Technology Conference, Salt Lake City, USA, April 28-May 1

[11] Gutiérrez, J.J., Nebut, C., Escalona, M.J., Mejías, M., Ramos, I.M.: Visualization of Use Cases through Automatically Generated Activity Diagrams. In: Czarnecki, K., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MoDELS 2008. LNCS, vol. 5301, pp. 83–96. Springer, Heidelberg (2008)

[12] Myers, G.: The Art of Software Testing, 2nd edn. Addison-Wesley, USA (2004)

[13] Naresh, A.: Testing From Use Cases Using Path Analysis Technique. In: International Conference on Software Testing Analysis & Review (2002)

[14] Object Management Group, Unified Modelling Language 2.4 (2011), `http://www.omg.org` (last visit June 24, 2011)

[15] Object Management Group. Query View Transformation Specification 1.0 (2010), `http://www.omg.org` (last visit June 24, 2011)

[16] Kapová, L., Goldschmidt, T., Becker, S., Henss, J.: Evaluating Maintainability with Code Metrics for Model-to-Model Transformations. In: Heineman, G.T., Kofron, J., Plasil, F. (eds.) QoSA 2010. LNCS, vol. 6093, pp. 151–166. Springer, Heidelberg (2010)

[17] Ostrand, T.J., Balcer, M.J.: Category-Partition Method. Communications of the ACM, 676–686 (1988)

[18] Ruder, A.: UML-based Test Generation and Execution. Rückblick Meeting, Berlin (2004)

[19] Schmidt, D.C.: Guest Editor's Introduction: Model-Driven Engineering. Computer 39(2) (2006)

[20] Supporting web, `http://www.iwt2.org/mdetest` (last updated April 15, 2012)