Key Features for a Successful Model-Driven Development Tool

Beatriz Marín[1], Andrés Salinas[1], Juan Morandé[1], Giovanni Giachetti[2] and Jose de la Vara[3]
[1]*Facultad de Ingeniería, Universidad Diego Portales, Av. Ejército 441, Santiago, Chile*
[2]*Facultad de Ingeniería, Universidad Andres Bello, Sazié 2325, Santiago, Chile*
[3]*Certus Centre for Software V&V, Simula Research Laboratory, P.O.Box 134, 1325 Lysaker, Norway*

Keywords: Model-Driven Development (MDD), Tools, MDA, Features.

Abstract: The main focus of Software Engineering discipline is to establishing methods and processes for the effective and efficient development of software projects. One of the most relevant paradigms for achieving this goal is Model-Driven Development (MDD). MDD uses models at different abstraction levels to automatically generate software products by means of model-transformations. However, one of the main issues related to the development of MDD technologies is the lack of standardization in terms of the features that need to be considered to support the current industry needs. This difficults the comparison of existing technologies since there are not a reference point for the creation of new MDD approaches with their corresponding supporting tools. This paper analyses these industry needs through an exploratory study. From the results obtained, it states the main features that must be supported by MDD tools. In addition, this paper presents an analysis about the presence or absence of these features in a set of industrial MDD tools.

1 INTRODUCTION

The goal of Software Engineering is to establishing methods and processes for the effective and efficient development of software projects (Kitchenham and Pfleeger, 1996). The Model-Driven Development paradigm (Selic, 2003) has become into a relevant mean for achieving this purpose at both, industry and academic domains.

MDD approaches use models at different abstraction levels that are independent of technical platforms to generate different kinds of software products by means of model transformations (such as code, documentation, new modeling artifacts, etc.). Thus, a correct model specification is essential to automatically generate a complete software solution without programming a single line of code.

In this context, the MDD paradigm is moving the software development processes to a different dimension, from the solution space (software product) to the problem space (conceptual models). In this journey, the success of adopting MDD solutions is in direct relation with the capability of the available tools to satisfying the needs of the different development projects.

However, even though there exists an important amount of MDD tools in the software industry (Marín et al., 2013), there is a lack of references that state the key aspects that need to be considered for selecting (or developing) a MDD tool properly aligned with industrial MDD processes.

In this context, this paper presents a twofold contribution: (1) this paper presents and analyses a set of key features that an MDD tool must have to be successfully adopted in industry software developments; and (2) the paper also presents which of these key features are currently taken into account by considering a set of existing MDD tools.

The rest of the paper is organized as follows: Section 2 presents some relevant related work. Section 3 presents an exploratory study. Section 4 presents the key features. Section 4 presents an analysis of these features in relation to a set of MDD tools that are currently available. Finally, Section 5 presents some conclusions and further work.

2 RELATED WORK

Bran Selic in (Selic, 2003) proposes the following success criteria for the development MDD tools by considering his experience in the development of the IBM Rational Software.

Standards. The use of standards is essential to

Marín B., Salinas A., Morandé J., Giachetti G. and De la Vara J..
Key Features for a Successful Model-Driven Development Tool.
DOI: 10.5220/0004723705410548
In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development* (MODELSWARD-2014), pages 541-548
ISBN: 978-989-758-007-9

spread the adoption of new technologies and reduce the learning curve related to the application of MDD software production processes. In this context, the Model-Driven Architecture (MDA) proposed by OMG is probably the most referenced implementation schema for MDD tools.

Observability and executability of the model. The observability refers to the integration of comparing tools that helps in the identification of model's versions. The executability refers to the capacity to execute the models in early stages of software development, providing a learning ability though experimentation.

Efficiency of the generated code. The efficiency of the generated code can be decomposed in performance and use of memory. In addition, it is also important taking into account the size of the generated system and the compilation time of the models.

Furthermore, Richard Paige and Dániel Varró in (Paige and Varró, 2012) presents the knowledge obtained during the development of two tools: Epsilon (Epsilon, 2010) and VIATRA2 (Balogh and Varró, 2006), which is briefly analysed below.

Use of requirement models for driving the development process. The use of requirement models in MDD processes help to motivate and thread the development complexity, i.e.; it drives the construction in terms of development iterations.

Flexibility of architectures and modeling tools. At the beginning, it is promoted the flexibility of architectures instead of the selection of the correct architecture. Having a flexible architecture it is possible to adapt the generation of the final software products to different architectural patterns in accordance to the needs of the development project.

Modeling technologies. Modeling technologies refers to the methods used to represent the models, for instance EMF model representation. The selection of the modeling technology significantly affects in the usability and flexibility of the modeling tool in terms of the implementation of model transformations, typing of model elements, etc. In addition, the final user do not necessarily must to know how the representation of models is implemented; final users just need to know the graphical representation of models in order to avoid complexity of the modeling technology.

Summarizing, the works analyzed consider specific views of the development of MDD tools by introducing some desirable features. However, these features are not validated in industrial contexts. To tackle this situation, we perform an exploratory study in order to discover the relevant features that an MDD tool must have.

3 EXPLORATORY STUDY

The exploratory study consists in an online questionnaire of 19 questions. These questions have been defined by considering strong and weak points of previous works analyzed. Moreover, the model verification and interoperability criteria were added to the study.

The model verification is a relevant feature in MDD tools since this assures that all the model information can be transformed into the corresponding software artifacts (syntactic verification) and the final software product is properly aligned with the users' requirements (semantic verification).

The model interoperability is a key feature to facilitate (and automate) the interchange of modeling information among tools related to a same domain (e.g. interoperability among different UML tools), as well as the interoperability among different modeling approaches (e.g. for bridging requirement and design modeling approaches).

Table 1 shows the questions and the related feature of the questionnaire. Each question was answered by using a 5-point Likert scale. To obtain qualitative aspects related to each question, an open text box that can be filled with open answers was placed.

The estimated time to response was specified in 15 minutes. In order to present the questions in an attractive way, different icons that represent the values of the Likert scale were used (from totally disagree represented by 1 to totally agree represented by 5). The questionnaire can be seen at https://testmodeproject.typeform.com/to/eUhWIU.

Four relevant tool vendors response the online questionnaire and automatically the answers were sent to the authors. Collected answers were located in a table to further analysis (see Table 2). In Table 2, the column Id represents the number of the question, and TV1 to TV4 represents the tool vendors that response the questionnaire.

Table 1: Questions and related features of the questionnaire.

Id	Question	Feature
Q1	The MDD tool must support the standard UML.	Standarization
Q2	The MDD tool must support MDA in terms of CIM, PIM, PSM	Standarization
Q3	The MDD tool must provide extension mechanisms for modeling languages customization to allow the communication with different tools?	Interoperability
Q4	The MDD tool must provide extension mechanisms that allow the communication/interoperability with different tools	Interoperability
Q5	It is necessary to have a graphical visualization of a model.	Observability
Q6	It is necessary to have a version manager of models.	Observability
Q7	It is necessary to have easy human interaction (such as touch screens) to work with models.	Efficiency
Q8	The MDD tool must provide verification mechanisms of the models	Verification
Q9	The MDD tool must provide automatic defect detection for the models	Verification
Q10	The MDD tool must provide automatic test case generation of the models	Verification
Q11	The MDD tool must provide simulations of the executability of the model	Executability
Q12	The MDD tool must allow redefinitions of the transformation of the models	Flexibility
Q13	The generated code of an MDD tool must have the same efficiency than the code generated with traditional programming	Flexibility
Q14	The MDD tool must allow the selection of different architectural patterns to generate code	Flexibility
Q15	The MDD tool must be able to generate code reviews	Code Generation
Q16	The MDD tool must generate at least the skeleton of the code	Code Generation
Q17	The MDD tool must generate totally executable code	Code Generation
Q18	The MDD tool must support the specification of all views of a system	Requirements
Q19	The MDD tool must save the traceability from requirements to code	Requirements

4 FEATURES FOR MDD TOOLS

This section presents a set of features that any MDD tool designed to work in industrial projects should offer. At this point, it is important to note that (1) the main input of MDD tools is a conceptual model that is independent from the development platform used, and (2) the MDD tool must be able to generate code through the model specified.

4.1 Standardization and Interoperability

Since its inception, the Object Management Group (OMG) has promoted the standardization of different model-based and object-oriented approaches for software development. In this context, the main exponent of OMG standards is UML (OMG, 2010). Other standard defined by OMG is Model-Driven Architecture (MDA) (OMG, 2003), which provides a standard architecture to support MDD developments.

Table 2: Results of the questionnaire.

Id	TV1	TV2	TV3	TV4	Id	TV1	TV2	TV3	TV4
Q1	5	3	3	5	Q11	4	5	3	3
Q2	4	4	2	2	Q12	5	2	1	3
Q3	5	5	5	2	Q13	1	1	3	4
Q4	4	3	5	4	Q14	4	4	3	5
Q5	4	5	3	4	Q15	2	3	3	3
Q6	2	1	5	5	Q16	5	5	4	5
Q7	3	1	4	5	Q17	1	5	3	3
Q8	4	5	4	5	Q18	4	3	3	2
Q9	4	2	2	4	Q19	5	5	4	3
Q10	4	5	1	3					

Standardization promotes significant progress in the development of a MDD tool, allowing to establish an agreement of good practices that facilitate the reuse and interoperability among different tools and modelling approaches, for example by using XMI (OMG, 2007b).

Standardization also promotes specialization of tools in specific domains, leading the development of tools with more concrete and sophisticated features according to different application contexts.

Results of the exploratory study shows that 50% of the subjects are totally agree about the MDD tool must support the standard UML, and the other 50% answers are neutral about this question. For Q2, 50% of subjects are not agree regarding if it is necessary that MDD tools support the MDA architecture in terms of CIM, PIM, and PSM; and the other 50% of the subjects are agree with this question.

Regarding the interoperability, the exploratory study shows that 75% of subjects are totally agree with the affirmation of the MDD tool must provide extension mechanisms for modeling languages customization. Regarding Q4, 75% of the subjects state that MDD tool must provide extension mechanisms that allow the communication or interoperability with different tools.

4.2 Visualization and Management of Models

Due to the main input artifact in a MDD approach is a model, which specifies all the views that allow the complete representation of a system, it is essential that an MDD tool provides a suitable set of management and visualization features for models definition.

The minimum feature to be considered is the possibility of visualizing the models at design time. In particular, it is necessary to provide a graphical interface that facilitates the management of multiple views that are normally used for a system model specification. The exploratory study shows that 75% of the subjects state that it is absolutely necessary to have a graphical visualization of the model.

However, to appreciate graphically the model is not the only concern related to visualization that should taking into account when an MDD tool is built. Nowadays, more than ever, it is necessary to provide user interfaces that take into account quality characteristics such as usability (ISO/IEC, 2001); and new characteristics of hardware devices, such as touch panels in desktop and laptop computers (Garber, 2012). In the exploratory study we found that 50% of subjects agree to have an easy human interaction to work with models, and the other 50% answer that it is not necessary.

Regarding versioning of the models, the exploratory study shows that 50% of the subjects state that it is absolutely necessary to manage the versioning of the models. This is the only way to work in collaborative industrial projects, where different members of a development team can work over the same models. However, the remaining 50% of subjects response that it is not mandatory to have a version manager, but it is a desirable feature.

4.3 Verification

In MDD, the modeling language assumes the role of an implementation language due to it achieves a complete and automatic code generation; i.e., the model becomes the new programming code. In this context, the generated code becomes into a black box, rarely reviewed, as is rarely reviewed the machine code generated by a programming language compiler, because this generation is performed using widely accepted standards in the industry.

Thus, the verification of the model should be a mandatory feature of an industrial MDD tool. 100% of subjects response in the exploratory study that the MDD tools must provide verification mechanisms. This means using the MDD tool to find the presence of desirable characteristics and the absence of undesirable characteristics.

Regarding to the absence of undesirable characteristics, different researchers have applied reading techniques or heuristics to identify defects in UML models, such as (Travassos et al., 1999), (Conradi et al., 2003), (Lange and Chaudron, 2004), (Egyed, 2006). Reading the question if it is necessary that the MDD tool provides automatic defect detection of the models, the exploratory study reveals that it is desirable, but with a desactivation possibility because some defects depends on the methodology selected in the project; i.e. it should be good to plug custom defect detection mechanisms to supporting architects' needs.

4.4 Testing

Even though there exist several defect detection techniques proposed by the academia, the most important and most frequently used quality assurance technique applied in industry is testing. Usually, testing of software systems take up to more than 50% of development cost and time (Vos et al., 2010). For this reason, it is very important to perform the testing as early as possible in order to diminish this high cost. Despite there are several tools and methods for testing planning and control, and test case execution and monitoring, there still are a limited amount of approaches for test case design, selection of test data, and test evaluation.

To perform testing of models, there are several

researchers who have applied model-based testing to generate test cases (Dias Neto et al., 2007). These approaches usually create a state transition model to represent the current state of a system and the next state, specifying the events that occur to change the state. In these models, test designs are focused on paths of the execution of the events. However, state transition models only provide a partial view of the final software system (behavioural view). Thus, after the application of test cases generated from these models, it is necessary to manually testing the remaining functionality of the system. In order to reduce the human effort doing testing, MDD tools must provide model-based testing approaches that will be focus in holistic models.

Regarding testing, the exploratory study shows that 50% of the subjects answer that the MDD tool must provide automatic test case generation from models, and the other 50% answer that it could be good, but it is not essential.

4.5 Code Generation and Simulation

The MDD tool must generate at least the skeleton of the code. The exploratory study supports this asseveration with 75% of full agreement and 25% of agreement.

Supporting technologies for automating the model-based operations, such as, model transformations, model validation and verification, generation of software products (model compilation), are essential to achieve the benefits of MDD. In this context, level of generation facilities that a MDD tool provides is of paramount importance. Depending of the tools, the code generation can go from the skeleton or code fragments to the generation of the complete software products. Nevertheless, the exploratory study shows that only 25% of the subjects agreed with the fact that an MDD tool must generate totally executable code. 50% of subjects state that in some cases the total code generation is adequate, arguing that the model that generates the complete executable code has the same complexity than the code generated.

Regarding simulation, MDD tools should allow the execution of models even though they are incomplete (but valid). The main idea is not to wait until the model is finished to see how it looks like the software obtained from the part of the model that is already specified. This allows the correction of the model specification as early as possible. The exploratory study shows that 50% of the subjects agree in that the MDD tools must generate simulations of the models defined.

4.6 Transformation

The transformation of a model corresponds to a set of rules and activities such as refactoring, reverse engineering, application of patterns, among others. Transformations take one or more models as input, and by applying the rules specified, generate one or more output models, including the code of the final software product (implementation model).

A suitable MDD tool must offer a number of predefined transformations for assuring a complete model transformation. However, features oriented to define or customize the transformation rules implemented by an MDD tool are supported by 25% of the subjects in the exploratory study.

4.7 Efficiency and Scalability

MDD tools should significantly reduce time and efforts, and simplify the development of final software products. The productivity gained by using MDD tools can be significant when the code generated is quite equivalent in terms of efficiency and scalability to the code manually generated.

Various attempts have been performed in this direction. One prominent effort was computer-aided software engineering (CASE) in the beginning of 80s, which focused on developing software methods and tools that enabled developers to express their designs in terms of general-purpose graphical programming representations, such as state machines, structure diagrams, and dataflow diagrams (Schmidt, 2006). However, CASE tools were not successful in industry due to its inability to handle complex, production-scale systems in a broad range of application domains (Schmidt, 2006).

A relevant efficiency measure that can be considered when evaluating a MDD tool is related to the performance in terms of the volume of information handled, and the amount of memory used. Is expected that the automatically generated code efficiency has a deviation not greater than 10% compared to manually created code (Selic, 2003). One way of evaluating this would be comparing the amount of generated code lines between tools and their equivalent constructed by hand. A more appropriate way to evaluate this would be comparing the functional size measurement of both applications. For measuring the functional size of conceptual models it is possible to apply IFPUG measurement standards (ISO/IEC, 2003) and COSMIC (ISO/IEC, 2011), by applying, for example, the procedure presented in (Marín et al., 2010).

Results of the exploratory study reveals that efficiency in the generated code is not a relevant feature taking into account the increase in the productivity of MDD projects.

4.8 Architectures and Maintenance

Since, MDD tools work with platform independent models; the tool must support the transformation to executable code not only to a variety of languages, but also to different architecture design patterns. For example, if you want to generate in a particular programming language, the MDD tool may allow the selection between a client-server architecture (Berson, 1996) or model view controller (MVC) architecture (Reenskaug, 2003). The exploratory study reveals that 75% of the subjects agree that the MDD tool must allow the selection of different architectural patterns to generate code.

Regarding the generation of code reviews, 75% of the subjects are neutral about the ability of MDD tools to perform partial generation of code from model changes; i.e., it is not necessary to recompile the whole model if a small change is performed. We state that this would facilitate the maintenance of the system to small changes or corrections that may exist during the lifetime of the software created.

4.9 Requirements

Even though there exist standards for requirements documentation (such as IEEE 830 (IEEE, 1984)), the requirements elicitation phase is the less technical by nature. However, it is desirable that the MDD tool supports the traceability from requirement specification to the other views of the system.

The exploratory study shows a 75% of subjects agreed regarding traceability of MDD tools. The remaining 25% answer that it depends on the usage of the tool.

5 GENERAL ANALYSIS

This section presents a general analysis of the features that existing tools have. To do this analysis, a list of products that are compliant with the MDA approach were considered (OMG).

There are 48 MDA tools recognized by OMG (OMG). However, many of these tools are not longer available are not updated for more than one year. From the 48 MDA tools, only 10 (corresponding to 21%) are currently in use and have active support. The remaining 79% were deprecated or acquired by a larger company.

The 10 available tools provide support for different domains, such as real-time or management information systems (MIS). We focus the analysis in MDD tools of the management information systems since it is broadly used in industry. Thus, just three tools were taking into account in the analysis.

In order to make a more representative analysis, open-source tools that are not included in the OMG site were also included. Therefore, to complement the analysis, five open-source tools were added. Thus, eight MDD tools were finally analyzed.

First, a characterization of the selected tools was performed regarding the modeling language used, the system views covered by the tool, the language for the specification of the functional view, and the software product generated (see Table 3).

As Table 3 shows, seven tools support UML

Table 3: Characterization of MDD tools analyzed.

MDD tool	Modeling Language	System Views	Functional View Language	Products Generated
AndroMDA	UML by using MagicDraw	Structural and Dynamic Views	-	Structure of the system
OpenMDX	UML	-	POJO	-
Acceleo	UML2 by using EMF	Structura, Dynamic, and Functional Views	OCL	Code Skeleton.
TopCased	UML2 by using EMF	Structura, Dynamic, and Functional Views	OCL	Code Skeleton. Documentation. Allows complete code generation by using plugins.
StarUML	UML 2.0	Structura, Dynamic, and Functional Views	-	Code Skeleton and requirements and implementation documents
Integranova	OO-Method	Structural, Dynamic, Functional, and Presentation Views	OASIS	Complete fully working generation code. Documentation. Fucntional size Measurement.
IBM Rational Rose	UML2.1	Structural and Dynamic Views	OCL	Complete fully working generation code
Blu Age	UML 2.1	Structural, Dynamic, Functional, and Presentation Views	OCL	Complete fully working generation code

(OMG, 2007a) (OMG, 2010) and one tool support OO-Method (Pastor et al., 2001) as modeling language. UML is the standard the facto to be used in industry, and OO-Method starts from the UML class diagram and adds semantic information to allow the generation of fully-executable code. Using UML or UML-based modeling languages reduces the learning curve of the tool and facilitates the integration with different project management tools.

In order to avoid or diminish defects and faults of the generated applications, it is very important that the MDD tool provides support to the holistic representation of a system in a conceptual manner, which includes the static, dynamic, functional, and presentation views. More details of these views can be found in (Marín et al., 2013). From the eight tools analyzed: one tools does not detail the different views supported; two tools support the structural and the dynamic views; four tools support the structural, dynamic and functional views; and only two tools supports structural, dynamic, functional, and presentation views.

Regarding the language for the specification of system functionality, four tools uses OCL (OMG, 2006), one tool uses OASIS (Pastor et al., 1992), and one tool uses Plain Old Java Objects (POJO).

Regarding the product generated, one tool does not specify the final product generated, four tools generate the skeleton of code, and just three tools generate the code completely.

An analysis performed to the features established in Section 3 shows that regarding to standardization, most of the tools use the UML modeling language. Only one tool does not support UML, but it supports an extension of UML called OO-Method. However, not all the tools support the same version of UML, which will alleviates the exportation of the models to other supported formats and promotes the interoperability of tools.

Regarding the visualization of the models, the analyzed tools provide graphical visualization or connections with graphical tools (such as MagicDraw or Eclipse EMF). However, there not exist tools that take advantage of new interaction features that could increase the productivity of software engineers.

Regarding the verification of the model, some tools offer verification of syntactical defects in the models and verification of the consistency of the different views supported. Nevertheless, it is also necessary that MDD tools offer verification of semantic defects in order to prevent faults when the generated system is executed.

Regarding testing and simulation of the models, analyzed MDD tools do not offer options to generate tests or simulation artifacts in order to properly validate the models, rather than testing the code once the system is generated.

Regarding the transformation, even though the tools analyzed offer refactoring and reverse engineering features, they do not provide facilities to customize transformations in particular situations.

With respect to efficiency and architecture, commercial tools such as Rational or Integranova can export to a number of different architectures, but at open-source tools this option is available. In addition, none of the eight tools provides support to code optimization.

Finally, with respect to requirements traceability, the tools analyzed do no provide mechanisms to keep the traceability from requirements to code.

6 CONCLUSIONS

This paper presents a set of features that an MDD tool should have for successful application and adoption of the Model-Driven Development paradigm in industry. An exploratory study was performed to validate this set of key features for MDD tools.

In addition, an analysis of available MDD tools has been presented to evaluate the adoption of existing tools regarding the features proposed. To perform this analysis, the current OMG catalogue of tools was considered. However, almost the 80% of these tools are deprecated, which dramatically reduce the set of MDD tool that met with the features proposed and are aligned with the needs of software development projects.

The remaining tools in the OMGs list have reached a level of maturity in which it is possible to generate solutions from a model, however, none of these tools support all the features presented in this paper. Thus, an interesting challenge is to collaborate with the existing MDD tool providers to analyze in deep the features proposed and to develop a tool (or a suite of tools) that be aligned with all these features.

This work is part of a research agenda that aims to develop MDD tool for systems in the domain of information systems management. In addition, this agenda also include the development of techniques to semantically verify the models and generate test cases automatically from conceptual models, and empirical studies that validate these techniques.

ACKNOWLEDGEMENTS

This work was funded by FONDECYT project TESTMODE (Ref. 11121395, 2012-2015).

REFERENCES

Balogh, A. & Varró, D. 2006. Advanced Model Transformation Language Constructs in the VIATRA2 Framework. *ACM Symposium on Applied Computing – Model Transformation Track (SAC 2006)*. Dijon: ACM Press.

Berson, A. 1996. *Client/server architecture*, McGraw-Hill.

Conradi, R., Mohagheghi, P., Arif, T., Hegde, L. C., Bunde, G. A. & Pedersen, A. 2003. Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment. *17th ECOOP*. Springer.

Dias Neto, A. C., Subramanyan, R., Vieira, M. & Travassos, G. H. 2007. A survey on model-based testing approaches: a systematic review. *1st ACM international workshop on Empirical assessment of software engineering languages and technologies (WEASELTech '07)*. NY, USA: ACM.

Egyed, A. 2006. Instant Consistency Checking for the UML. *28th ICSE*. Shangai, China: ACM.

Epsilon. 2010. http://www.eclipse.org/gmt/epsilon (Online).

Garber, L. 2012. Tangible User Interfaces: Technology You Can Touch. *IEEE Computer,* 45**,** 15-18.

IEEE 1984. IEEE 830 Guide to Software Requirements Specifications.

ISO/IEC 2001. ISO/IEC 9126-1, Software Eng. – Product Quality – Part 1: Quality model.

ISO/IEC 2003. ISO/IEC 20926, Software Engineering – IFPUG 4.1 Unadjusted Functional Size Measurement Method – Counting Practices Manual.

ISO/IEC 2011. ISO/IEC 19761, Software Engineering – COSMIC – A Functional Size Measurement Method.

Kitchenham, B. & Pfleeger, S. 1996. Software Quality: The Elusive Target. *IEEE Software,* 13**,** 12-21.

Lange, C. & Chaudron, M. 2004. An Empirical Assessment of Completeness in UML Designs. *8th Conf. on Empirical Assessment in Software Eng. (EASE)*. IEEE.

Marín, B., Pastor, O. & Abran, A. 2010. Towards an accurate functional size measurement procedure for conceptual models in an MDA environment. *Data & Knowledge Engineering,* 69**,** 472–490.

Marín, B., Pereira, J., Giachetti, G., Hermosilla, F. & Serral, E. 2013. A General Framework for the Development of MDD Projects. *1st International Conference on Model-Driven Engineering and Software Development - MODELSWARD 2013*. Barcelona - Spain: SciTePress.

OMG. *MDA Products and Companies* [Online]. Available: http://www.omg.org/mda/committed-products.htm.

OMG 2003. MDA Guide Version 1.0.1. *In:* JOAQUIN MILLER & MUKERJI, J. (eds.).

OMG 2006. Object Constraint Language 2.0 Specification.

OMG 2007a. UML 2.1.2 Infrastructure Specification.

OMG 2007b. XMI 2.1.1 Specification.

OMG 2010. UML 2.3 Superstructure Specification.

Paige, R. F. & Varró, D. 2012. Lessons learned from building model-driven development tools. *Software & Systems Modeling,* 11**,** 527-539.

Pastor, O., Gómez, J., Insfrán, E. & Pelechano, V. 2001. The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. *Information Systems,* 26**,** 507–534.

Pastor, O., Hayes, F. & Bear, S. 1992. OASIS: An Object-Oriented Specification Language. *Int. Conference on Advanced Information Systems Engineering (CAiSE)*. Manchester, UK.

Reenskaug, T. 2003. The Model-View-Controller (MVC), Its Past and Present. University of Oslo.

Schmidt, D. 2006. Model Driven Engineering. *IEEE Computer,* 39**,** 25-31.

Selic, B. 2003. The Pragmatics of Model-Driven Development. *IEEE Software,* 20**,** 19–25.

Travassos, G., Shull, F., Fredericks, M. & Basili, V. 1999. Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality. *OOPSLA' 99*. Denver, CO, USA.

Vos, T. E., Baars, A. I., Lindlar, F., Kruse, P. M., Windisch, A. & Weneger, J. 2010. Industrial Scaled Automated Structural Testing with the Evolutionary Testing Tool. *Third International Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society.