Learning Pros and Cons of Model-Driven Development in a Practical Teaching Experience*

Óscar Pastor[1], Sergio España[2], Jose Ignacio Panach[3]

[1]Centro de Investigación en Métodos de Producción de Software - ProS
Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain
opastor@pros.upv.es

[2]Department of Information and Computing Sciences, Utrecht University. The Netherlands
s.espana@uu.nl

[3]Escola Tècnica Superior d'Enginyeria, Departament d'Informàtica, Universitat de València
Avenida de la Universidad, s/n, 46100 Burjassot, Valencia, Spain
joigpana@uv.es

Abstract. Current teaching guides on Software Engineering degree focus mainly on teaching programming languages from the first courses. Conceptual modeling is a topic that is only taught in last courses, like master courses. At that point, many students do not see the usefulness of conceptual modeling and most of them have difficulty to reach the level of abstraction needed to work with them. In order to make the learning of conceptual modeling more attractive, we have conducted an experience where students compare a traditional development versus a development using conceptual models through a Model-Driven Development (MDD) method. This way, students can check on their own pros and cons of working with MDD in a practical environment. Comparison has been done in terms of Accuracy, Effort, Productivity and Satisfaction. The contribution of this paper is twofold: the description of the teaching methodology used throughout the whole course; and the presentation of results and discussions of the comparison between MDD and a traditional development method. Results show that Accuracy, Effort and Productivity are better for MDD when the problem to solve is not easy. These results are shown to students to promote a discussion in the classroom about the use of MDD. According to this discussion, the most difficult part of using MDD is the learnability and the best part is the automatic code generation.

Keywords: Model-Driven, Conceptual Modeling, Teaching Methodology

1 Introduction

We are three teachers of a subject in a master course from Universitat Politècnica de València (Spain) called "Engineering of Information Systems", which deals with the topic of conceptual modelling within a Model-Driven Development (MDD) perspec-

* This work was developed with the support of Generalitat Valenciana-funded IDEO project (PROMETEOII/2014/039)

tive. The main goal of the master course is to teach how to abstractly represent system information through conceptual models (CMs) and how this CMs can be used to obtain the final application in a Conceptual-Schema software development context [1]. The subject has been prepared using an MDD tool (INTEGRANOVA [2]). This way, students can focus all their efforts on building conceptual models, relegating the code generation to automatic model to code transformation rules applied by the tool.

During several previous years we have noticed that, in general, the use of conceptual models as a way to represent all system features is new for students. All previous subjects in the software engineering degree were focused on building UML models that only represent data persistence and part of the system behaviour. These UML models guide the manual implementation of the code at best. Most students build UML models at first stages of the software development but once they start to implement the code, these models are forgotten and in most cases, they are not updated with all the characteristics specified within the code. At the end, UML models are a type of heavy-obsolete documentation that nobody (students and teachers) query, focusing on the implementation and the developed system.

Within that context, the topic of conceptual modelling was not motivating for students that are accustomed to focus on writing code and compiling it. In order to motivate the students, we propose teaching conceptual modelling comparing MDD versus a traditional software development method. This should help to wake up a critical spirit in each student, drawing personal conclusions about the pros and cons of conceptual modelling in practice. There are several works that have studied the advantages and difficulties of MDD, such as [3-4], but we aim to check whether students, which are not familiar with conceptual modelling, perceived the usefulness of MDD. At the end of the course, apart from teaching the students conceptual modelling, we have a set of empirical data that we can use to compare MDD regarding a traditional development. The analysis is based on the study of 4 variables: Accuracy, Effort, Productivity and Satisfaction. The results of analyzing those data statistically are shown to the students as a starting point for a general discussion about the pros and cons of MDD.

We have conducted this teaching experience during two years. The results of the data extracted the first year have been published in [5]. Results showed that MDD obtains better values than a traditional method only when the problem complexity of the system to develop increases. Regarding Effort, Productivity and Satisfaction we did not identify differences. In order to check the idea that the complexity of the problem can affect the results, we increased the complexity of the problems in the second year.

The contribution of this paper is twofold: the description of the teaching methodology used in both years and the summary of results and discussions obtained in the second year. The description of the methodology aims to promote the adoption of future replications in universities different from UPV. The methodology consists in starting with the development of a web application using a traditional method. Next, we have several classes dedicated to learn MDD. Next, students develop another system through MDD. At the end of the course, we compare how much Accuracy, Effort, Productivity and Satisfaction have been experienced by each subject in both treat-

ments, and we conduct a discussion in the classroom. Results of the second year show that when problems to solve are complex, MDD obtains better results not only for Accuracy but also for Effort and Productivity. During the discussion, students claimed that the main cons for using MDD is the high learning curve, while the main pros is the automatic code generation from conceptual models.

The paper is organised as follows. Section 2 discusses related works. Section 3 describes the teaching methodology used during both years. Section 4 shows the results after analyzing data of the second year. Section 5 shows the results of the discussions in the classroom after showing the results to students. Finally, Section 6 presents the conclusions.

2 Related Works

The topic of techniques to teach conceptual modeling to students has been tackled by several researchers. Muller [6] has elaborated a list of current challenges to teach conceptual modeling. These challenges refer to including conceptual modeling in a multi-disciplinary world to teach that: building systems is an engineering task; customer context and system context is not the same; a system is dynamic; it is important to quantify; systems are not always well-defined; analysts need part of psychology to deal with customers; analysts need a critical attitude. All these challenges show that there is still much work to do in order to better teach conceptual modeling.

Several works explain their experience teaching modeling through Entity-Relationship (ER) schemas, such as Davis [7]. Davis describes that students learn ER models with a combination of individual and team work, including instructor feedback as well as peer interaction. The teacher gives an ER schema and the students must interpret the meaning of constructs used in it; and given a description of requirements, students must design an ER schema. Other similar work has been done by Keberle and Utkin [8], who present a system called “Chen Worlds” to teach ER modeling. The proposal is based on the idea of gaming the environment to accelerate the learning of conceptual models in a course of database information systems. Chen Worlds is a software system for learning, building, visualizing and validating conceptual models in ER notation.

There are other works that have dealt with conceptual models different from ER, such as i*. In this line, Paja et al. [9] have reported their experience teaching i* in a master course. The work concludes that i* analysis allows the students to better understand the activities they perform. This helps to refine models until they are more meaningful and more likely to fulfill their purpose. Other techniques are based on constructivism, such as Zhuoyi et al. [10]. These authors propose a constructive teaching model in a database course where students must learn conceptual modeling under teachers’ guidance. Students explore and find knowledge, construct the meaning and learn to cooperate and communicate with others. According to the authors, constructivism arises enthusiasm of the students and improves the ability of solving problems.

Some works have compared two teaching techniques, such as Kung et al. [11] who have compared top-down versus bottom-up approaches to build conceptual models.

Results show that with proper experience, students can do it better in a bottom-up design. Sedrakyan et al. [12] define a proposal to build models and generate a prototype of application using those models. The proposal has been compared with other traditional techniques that do not generate prototypes. The results show that the proposal improves the understanding of students.

Some authors teach MDD in their courses, such as Akayama et al. [13], who teach conceptual models through MDD. Akayama compared a development using MDD versus another one without MDD. Results showed the effectiveness of MDD. Our proposal is aligned with this idea, as next sections describe.

3 Teaching Methodology

This section describes the teaching methodology that combines MDD with a traditional development (Figure 1). Sessions took 2 hours and there was 1 session per week, with a total of 14 sessions. The course has been designed in such a way that all the students work in pairs for logistic reasons. The three teachers participated at the same time during the course to teach MDD and to report data to analyze the comparison between both methods. Next, we describe each step of the methodology.

In step 1, students fill in a demographic questionnaire to check the level of experience in a traditional development method and in MDD. The course presupposes that most of the students already know how to develop a system through a traditional method but they know nothing about MDD. We must check this idea with the demographic questionnaire. In step 2, we propose a training problem to develop a web application as homework. This training aims to ensure that students are capable of programming a system from scratch. The time to develop the system is 15 days, students can use the development framework for web applications that they better know and they can draw in a paper any conceptual model that they complementary need. Half of the students drew a UML class diagram and the others used no model. During these two weeks, the teacher teaches the basics of MDD. There is not time inside the classroom to develop the system since we assume that every student has enough knowledge to develop the system on his own. Once the period of the training is over, the teacher evaluates the training through a set of test cases on the system. At this point, we can ensure that all the students have enough knowledge to develop a system with a traditional method. In step 3, students must develop another web application from scratch in a period of 4 hours in the classroom under the teacher supervision. In the same way as in the training, students can choose the programming language and draw in a paper any conceptual model they need. We used two problems to avoid that results were dependent on only one problem. Problems were assigned randomly to pairs such a way they are balanced among groups. At the end of this step, the teacher evaluates the developed problem and students fills in a satisfaction questionnaire about the use of a traditional development method.

Next, in step 4, the teacher explains MDD during 12 hours using INTEGRANOVA [2] as tool based on UML models that supports MDD. In step 5, we use a training problem similar to the training problem used in step 2. The students have 15 days to

develop the problem as homework using INTEGRANOVA. We use 6 hours in the classroom to support this development. At the end of this development, the teacher evaluates the result running test cases. At this point, we can ensure that students have enough knowledge to work with MDD. In step 6, students have to develop a system from scratch using MDD without any help from the teacher. We swap the problems used in step 3 such a way students do not develop the same problem they developed manually. At the end, the teacher evaluates the system running test cases and students fill in a satisfaction questionnaire about the use of MDD.

Finally, the teachers analyze statistically the data obtained throughout all sessions and show the results to students. In the last session, there is a discussion in the classroom where, taking as starting point the results of the analysis, each student gives an opinion about pros and cons of MDD according to this practical experience.

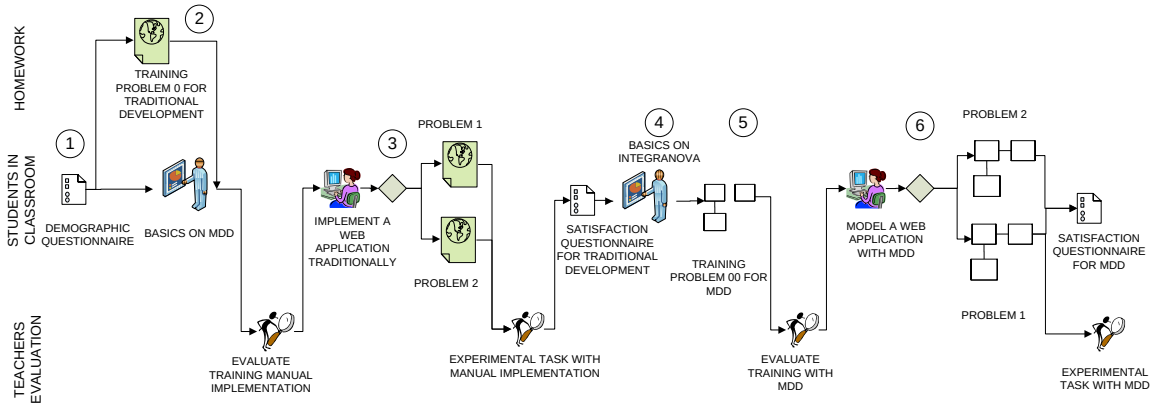

Fig. 1. Schema of the teaching methodology

3.1 Design of the Practical Experience

Next, we describe the design used in the teaching methodology. In order to arise ideas for and against MDD, we designed a practical experience based on four research questions: (**RQ1**) Is software accuracy affected by MDD? ; (**RQ2**) Is developer effort affected by MDD?; (**RQ3**) Is developer productivity affected by MDD?; (**RQ4**) Is developer satisfaction affected by MDD?. All these questions have been extracted from works that claim MDD benefits, such as [14-15]. The idea is that every student checks all these claims on his own to extract conclusions about the use of MDD. The teacher collects data throughout all the experience to answer the four research questions. At the end of the course, results are shown to discuss in the classroom the pros and cons of MDD.

Research questions are written as null **hypothesis** that the teacher must check statistically: (**H_{01}**) The software accuracy of a system built using MDD is similar to software accuracy using a traditional method; (**H_{02}**) The developer effort to build a system using MDD is similar to effort using a traditional method; (**H_{03}**) The developer productivity using MDD to build a system is similar to productivity using a traditional

method; ($\mathbf{H_{04}}$) The developer satisfaction using MDD to build a system is similar to satisfaction using a traditional method. We work with the **factor** Method, with two levels: MDD and a traditional method; and 4 **response variables**: Accuracy, Effort, Productivity and Satisfaction. The developed problem is a block variable since we are not interested in studying its effect.

Next, we describe the **metrics** used to check the null hypotheses. We measure Accuracy as the percentage of acceptance test cases that are successfully passed. We used 4 different metrics for Accuracy, from more restrictive to less restrictive:

- All or nothing (AN)**:** we consider that a test case is satisfied only if every item is passed.
- Relaxed all or nothing (RAN): we consider that a test case is satisfied when at least 75% of items are passed.
- Weighted items (WI): we assign a weight to each test item depending on the complexity of its functionality. When test cases are run, we add the weights of passed items.
- Same weight for all items (SW): we assign the same weight to each item within a test case (independently of complexity) in such a way that the addition of all the weights of the items is 1 per test case. When test cases are run, we add the weights of passed items.

Effort is measured as the time taken by each pair to develop the web applications from scratch. Productivity is measured as the ratio Accuracy/Effort. Finally, satisfaction is measured as the positive attitude towards the use of the development method through a questionnaire based on a Likert scale. Metrics for Satisfaction are based on Moody's proposal: Perceived Usefulness (PU), Perceived Ease of Use (PEOU), and Intention to Use (ITU) [16].

Problems used in the experience are a system to manage a company of electrical appliance (Problem 1) and a system to manage a photography agency (Problem 2). Complexity of both problems is similar, Problem 1 has 40 function points and Problem 2 has 35. Complexity of problems used in both training sessions is also similar to complexity of Problem 1 and Problem 2. Since training problems were not analyzed statistically, we do not describe them.

The **design** is a paired design blocked by problems, since we apply both treatments (MDD and a traditional development) to each subject and we are not interested in studying the effect of the problem in the response variables.

We have conducted this experience during two courses. The results of the first course (the baseline experience) was published in [5]. Results of that preliminary study showed that MDD gets better Accuracy the more complex the problem to solve is. We did not identify differences between Effort, Productivity and Satisfaction comparing both methods. In order to study in depth the idea that MDD seems to be more robust to higher complexities, we replicated the experience in a second course with the same procedure, but in this case we used a more complex version of both problems (Problem 1 and Problem 2). Next sections focus on explaining the results obtained after analyzing response variables and the discussion of students in the replication of the second course.

4 Results

This section describes the results from the data extracted through the experience. These results were shared with the students to promote the critical thinking regarding MDD. The teachers have used a Mixed Model [17] to check whether null hypotheses can be rejected or not. In those cases where null hypotheses are rejected, we have calculated the effect size to know the degree of differences between both treatments. The effect size has been calculated through Cohen's Delta [18]. More than 0.8 is a large effect; between 0.79 and 0.5 is a moderate effect; between 0.49 and 0.2 is a small effect.

Table 1 shows the p-values and the effect size of our response variables. **Accuracy** obtains significant results for the four metrics (AN, RAN, WI and SW) since all p-values are less than 0.05. Since effect sizes are around 0.6, we can state that differences between MDD and a traditional method are moderate. So, we can reject $H_{01,}$ which means that software accuracy of a system built using MDD is not similar to software accuracy using a traditional method. MDD obtains better averages in Accuracy than a traditional method independently of the used metric.

Efficiency obtains also significant results, since p-value is less than 0.05. Effect size is 0.77, which means that differences between MDD and a traditional method are moderate. So, we can reject $H_{02,}$ which means that the developer effort to build a system using MDD is not similar to effort using a traditional method. MDD obtains better averages in Efficiency than a traditional method

Productivity obtains significant results considering the four metrics for Accuracy (AN, RAN, WI and SW), since all p-values are less than 0.05. Effect sizes are around 0.6, which means that differences between MDD and a traditional method are moderate. So, we can reject H_{03}, which means that the developer productivity using MDD to build a system is not similar to productivity using a traditional method. MDD obtains better averages in Productivity than a traditional method.

Satisfaction does not obtain significant results in any of the three metrics (PEOU, PU, and ITU), since p-values are higher than 0.05. So, effect sizes have not been calculated. We cannot reject H_{04}, which means that the developer satisfaction using MDD to build a system is similar to satisfaction using a traditional method.

Table 1. p-values and effect sizes.

	Accuracy				Efficiency	Productivity				Satisfaction		
	AN	RAN	WI	SW		AN	RAN	WI	SW	PEOU	PU	ITU
p-value	.00	.01	.00	.00	0.00	.00	.02	.01	.01	.56	.27	.85
Effect size	.66	0.6	.62	.61	0.77	.64	.58	.6	.59	-	-	-

According to these results we can state that the problem complexity affects positively MDD. In the baseline experience, we only got significant results for Accuracy when the problem to solve was complex. In this replication, where problem complexity has been increased regarding the baseline, we obtain better results for MDD in Accuracy, Efficiency and Productivity. This leads to think that the higher complexity

we try to solve with MDD, the better results we got. Note that even though problems were implemented significantly better with MDD, students did not feel a better satisfaction. All these results were shown to the student in the classroom for discussion.

5 Discussion in the Classroom

The subsequent discussion performed in the classroom arose a set of significant aspects. Figure 2 shows a summary of all the discussed aspects of MDD and the number of students who supported them. The main "pros" are the code generation and the quick software development; while the main "cons" are the difficult deployment and the learnability.

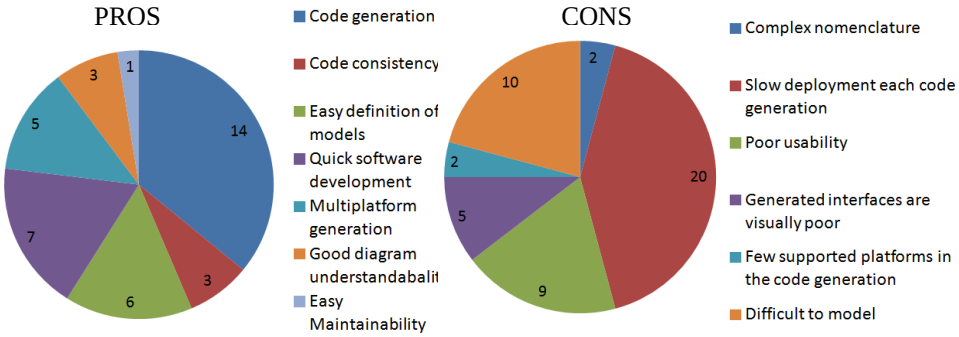

Fig. 2. Pros and Cons of MDD extracted from the discussion session

In the discussion there was a major agreement on a basic fact: the more complex a problem is, the bigger MDD improvements become. Students understood that a "real" conceptual-model compiler can provide a much more efficient and effective software development environment, as capturing the problem complexity in a conceptual (higher-level) model is easier than to do it at a pure (lower-level) programming level. To make true that "the model is the code" instead of "the code is the model", a conceptual programming [19] environment must make possible to specify the full problem complexity in a conceptual model, and this conceptual model must be executable. A main challenge for industrial MDD tools is then to make real this conceptual model compiler goal. The tool used in our experiment –INTEGRANOVA- is a very appropriate example of such a kind of MDD tool.

Another interesting discussion thread was related to know why satisfaction did not improve with MDD. It was again a major agreement in one relevant aspect: using MDD is not at a simple task. It was somewhat assumed that modeling should be easier than programming. The reality was that most of the students had serious problems to switch to a conceptual modeling-based mental strategy to face the problem solution. Conceptual modeling capabilities need to be prepared and to be practiced. While students clearly showed to have a good programming profile, largely practiced during their undergraduate learning experience, their conceptual modeling profile was not at

all so good. Current Computer Science degrees mainly focus on programming, not on modeling. Since conceptualizing is an essential task for a software engineer, this lack of conceptual modeling expertise in many curriculas can be seen as a serious handicap to make MDD practices become widely and correctly used.

In any case, a final significant reflection was that after practicing programming for years, practicing MDD only for a few weeks was enough to obtain even better results in 3 of the 4 response variables, while for the fourth one the difference was not significant. A final aspect that we want to explore in future experiments is how personal abilities of the student correlate with the results. We suspect that student that outperform with programing, also do it with modeling. Measuring what precise improvement is achieved in these cases when using MDD will be part of our future analysis.

6 Conclusions

This paper proposes a teaching methodology to teach conceptual models comparing a development from scratch using a traditional method versus a development from scratch using MDD, which is based on conceptual modeling. This way, students can experience on their own the pros and cons of working with MDD. Teachers report data about Accuracy, Effort, Productivity and Satisfaction throughout all the classes. Results conclude that Accuracy, Effort and Productivity are better working with MDD when the problem to solve is complex. These results are shown to the students and a discussion is done in the classroom. According to this discussion, the main "pros" is the quick code generation, while the main "cons" of MDD is the learning curve. These results agree with a previous baseline experience we conducted the previous year on the same subject.

Note importantly that there are some characteristics in our proposed teaching method that might affect the results. First, the limitation of 4 hours to the development of problems results in a maximum level for Effort. So, we do not know whether differences in time would have been higher without that limitation. Second, we use complex problems to analyze variables, but these problems are still toy problems. Maybe, more differences between MDD and a traditional method might have arisen if we had worked with real complex problems. Third, all the students knew to develop a web application with a traditional method, but the level of knowledge was not the same. Results might have been different if we had worked with professionals.

As future work, we plan to conduct more replications of the same teaching methodology during several years. It would be interesting to get more replications from different universities with other MDD tools.

7 References

1. Olive, A.: Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In: Oscar Pastor, J.F.e.C. (ed.): Proceedings of the 16th Conference on Advanced Information Systems Engineering, Lecture Notes in Computer Science, Vol. 3520. Springer-Verlag, Porto, Portugal (2005) 1-15

2. INTEGRANOVA Technologies: http://www.integranova.com.
3. Selic, B.: The Pragmatics of Model-Driven Development. IEEE software **20** (2003) 19-25
4. Hailpern, B., Tarr, P.: Model-Driven Development: the Good, the Bad, and the Ugly. IBM Syst. J. **45** (2006) 451-461
5. Panach, J.I., España, S., Dieste, Ó., Pastor, Ó., Juristo, N.: In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction. Information and Software Technology **62** (2015) 164-186
6. Muller, G.: Challenges in Teaching Conceptual Modeling for Systems Architecting. In: Jeusfeld, A.M., Karlapalem, K. (eds.): Advances in Conceptual Modeling: ER 2015 Workshops AHA, CMS, EMoV, MoBID, MORE-BI, MReBA, QMMQ, and SCME, Stockholm, Sweden, October 19-22, 2015, Proceedings. Springer International Publishing, Cham (2015) 317-326
7. Davis, K.C.: Teaching Conceptual Design Capture. In: Parsons, J., Chiu, D. (eds.): Advances in Conceptual Modeling: ER 2013 Workshops, LSAWM, MoBiD, RIGiM, SeCoGIS, WISM, DaSeM, SCME, and PhD Symposium, Hong Kong, China, November 11-13, 2013, Revised Selected Papers. Springer International Publishing, Cham (2014) 247-256
8. Keberle, N., Utkin:, I.V.: Teaching Conceptual Modeling in ER: Chen Worlds. ICTERI (2012) 222-227
9. Paja, E., Horkoff, J., Mylopoulos, J.: The Importance of Teaching Systematic Analysis for Conceptual Models: An Experience Report. In: Jeusfeld, A.M., Karlapalem, K. (eds.): Advances in Conceptual Modeling: ER 2015 Workshops AHA, CMS, EMoV, MoBID, MORE-BI, MReBA, QMMQ, and SCME, Stockholm, Sweden, October 19-22, 2015, Proceedings. Springer International Publishing, Cham (2015) 347-357
10. Zhuoyi, C., Na, L., Hongjie, Z.: Exploration of teaching model of the database course based on constructivism learning theory. Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on (2012) 1808-1811
11. Kung, H.-J., Kung, L., Gardiner, A.: Comparing Top-down with Bottom-up Approaches: Teaching Data Modeling. Information Systems Educators Conference, Information Systems Educators Conference (2012)
12. Sedrakyan, G., Snoeck, M., Poelmans, S.: Assessing the effectiveness of feedback enabled simulation in teaching conceptual modeling. Computers & Education **78** (2014) 367-382
13. Akayama, S., Hisazumi, K., Hiya, S., Fukuda, A.: Using Model-Driven Development Tools for Object-Oriented Modeling Education. MODELS (2013)
14. Borland: Keeping your business relevant with model driven architecture (MODEL-DRIVEN ARCHITECTURE). (2004)
15. Singh, Y., Sood, M.: Model Driven Architecture: A Perspective. Advance Computing Conference, 2009. IACC 2009. IEEE International (2009) 1644-1652
16. Moody, D.L.: The method evaluation model: a theoretical model for validating information systems design methods. In: Ciborra, C.U., Mercurio, R., Marco, M.d., Martinez, M., Carignani, A. (eds.): European Conference on Information Systems (ECIS 03), Naples, Italy (2003) 1327-1336
17. West, B.T., Welch, K.B., Galecki, A.T.: Linear mixed models: a practical guide using statistical software. CRC Press (2014)
18. Cohen, L.: Statistical power analysis for the behavioral sciences. Lawrence Earlbaum Associates (1988)
19. Embley, D.W., Liddle, S., Pastor, Ó.: Conceptual-Model Programming: A Manifesto. Handbook of Conceptual Modeling. Springer (2011) 3-16