MDD *vs* Traditional Software Development: a practitioner subjective perspective

Yulkeidi Martínez [1,*], Cristina Cachero[2],Santiago Meliá[2]

[1]Universidad Máximo Gómez Báez de Ciego de Avila, Cuba
yulkeidi@gmail.com
[2]Universidad de Alicante, Spain
{ccachero, santi}@dlsi.ua.es

Abstract. Context: Today practitioners have a myriad of methods from which to choose for the development of software applications. However they lack empirical data that characterize these methods in terms of usefulness, ease of use or compatibility, all of them relevant variables to assess the developer's intention to use them.
OBJECTIVE: To compare three methods, each following a different paradigm (Model-Driven, Model-Based and the traditional, code-centric, respectively) with respect to its intention to use by junior software developers while developing the business layer of a Web 2.0 application.
METHOD: We have conducted an experiment with 26 graduate students of the University of Alicante. The application developed was a Social Network, which was organized in three different modules. Subjects were asked to use a different method for each one of the three modules, and then answer a questionnaire that gathered their perceptions during its use.
RESULTS: The results show that the method that followed the Model-Driven development paradigm is regarded as the most useful, although it is also regarded as the more difficult to use. They also show that junior software developers feel comfortable with the use of models, and are likely to use them if accompanied by a model-driven development environment.
CONCLUSIONS: Model-driven development methods seem to show a great potential for adoption. However, further experimentation is needed to be able to generalize the results to a different population, different methods, languages and tools, different domains or different application sizes.

Keywords: MDD, MBD, code-centric development, experiment, usefulness, ease of use, compatibility, intention to use

1 Introduction

It is a well-known fact the Software Engineering (SE) community advocates the use of models in order to improve software development practices. Among this community, the Unified Modeling Language (UML) is well established as the standard modeling language. However, UML does not promote any particular development process, and modeling practices may greatly differ from organization to organization. One well-known way of classifying such modeling practices is

*Corresponding author.Address: Universidad Máximo Gómez Báez de Ciego de Ávila. Carretera a Morón km 9 1/2, C.P. 67800, Ciego de Ávila, Cuba.
Tel.: +53 33 217010; Fax: +53 33 266365

according to the extent to which modeling is used to support the development process. In this sense, Fowler [1]describes three different modes in which modeling languages (and UML in particular) can be used: sketch, blueprint and programming language.

- Sketches are informal diagrams used to communicate ideas. They usually focus on a particular aspect of the system and are not intended to show every detail of it. It is the most common use of the UML, and the recommended practice in agile, code-centric frameworks like Scrum [2]. When models are used as sketches, tools are rarely used, the modeling activity being mostly performed in front of blackboards where designers join to discuss complex or unclear aspects of the system.
- Blueprints are diagrams that show most of the details of a system in order to foster its understanding or to provide views of the code in a graphical form. Blueprints are widely used in Model-Based Development (MBD), which is at the core of standard development practices like the ones promoted by the Rational Unified Process (RUP) framework [3].
- Last but not least, fully-fledged models can be used to completely characterize the application. If such is the case, the diagrams may even replace the code, and be automatically transformed into executable binaries. This is the modeling use that lies at the core of the Model-Driven Development (MDD) paradigm.

This classification has led some authors to characterize the modeling maturity level of organizations based on the role of modeling in their software development process [4]. While code-centric development approaches require - at most - an informal use of modeling techniques and languages (such as sketches), both MBD and MDD approaches require a more formal use of models. Furthermore, MDD approaches rely on models that need to be syntactically correct, but also semantically accurate, consistent with each other and complete, so that they can be used as input for model transformations [5]. Table 1 dives into the main differences between these three ways of conceiving the development cycle, and compares, on a discipline (workflow) basis (Model, Implementation and Test), the three paradigms with respect to their reliance on models to help carry out the activities involved. The three disciplines used for comparison are the ones that typically appear in an Agile UP development process [6] during the construction phase.

Discipline	Code-Centric	Model-Based	Model-Driven
Model	Sketch or absent	Blueprint	Fully-fledged (DSL)
Implementation	Manual	Semiautomatic	Automatic or Semiautomatic
Test	Manual	Manual	Semiautomatic or Manual

Table 1: Correspondence between Agile UP disciplines and development paradigms

As we have aforementioned, code-centric approaches do not usually rely on Computer Aided Software Engineering (CASE) tools [7] to represent the *model* discipline. On the contrary, they usually promote the use of, at most, whiteboard sketching that permits to communicate the main ideas among different team members in a quick way. Also, the *implementation* and *test* disciplines are usually tackled manually. When the Model-Based paradigm is followed, its *model* discipline generates a blueprint, usually with the help of a UML tool. These UML tools usually permit to obtain a partial *implementation* (usually plain classes with attributes and empty methods). However, code *testing* is usually carried out in a manual way. Finally, the Model-Driven paradigm typically relies on fully-fledged models that are usually supported by a Domain-Specific Language (DSL). This paradigm requires *modeling* to be carried out by means of a Model-Driven CASE tool that permits to generate, either partially (semiautomatic) or in full (automatic), the final *implementation*. The corresponding *tests* can, depending on the development environment, be partially inferred from models or manually implemented.

As for the CASE tools available for MBD and MDD methods, and depending on the particular paradigm for which they were devised, they may offer not only modeling environments, which assure the syntax correctness of the models, but also model checking and partial or complete software generation capabilities. Well known MBD environments include MagicDraw [8], Poseidon [9], Visual Paradigm [10] and Rational Software Modeler (RSM) [11]. Although all of them offer certain code generation capabilities, they are not able to generate fully-fledged applications, and most MBD approaches do not rely on them for the coding phase. Regarding MDD, some of the best known currently available tools are TrueView Domain Modeller [12], Borland Together [13], OOH4RIA IDE [14] and WebRatio [15], although more tools claiming to provide powerful modeling environments and code-generation capabilities are appearing by the day.

Behind all these efforts, it underlies the assumption that using development methods that rely on models and tools with code generation capabilities improves the global developer experience when developing applications, which in turn improves their intention to adopt the method, particularly as systems become larger and more complex [16,17]. This assumption has been reliably supported by different studies that show MDD economical benefits and advantages over both MBD approaches and traditional, code-centric, approaches [18,19]. Among these advantages we can cite lower entire product life cycle and maintenance costs, higher end-user satisfaction [20], shorter time-to-market and less human resources, short and long term productivity gains, improved project communication and software quality improvements or defect and rework reduction [21,22,23] . Also, these advantages are justified in literature by the higher level of compatibility between systems, the simplified design process, and the better communication between individuals and teams working on the system that the MDD paradigm fosters [24,25].

However, in spite of these results, the purported paradigm shift from pure code-centric approaches to MDD that has been expected in industry for years [26] is still to come [27].

We agree with [27] in that this low level of adoption of MDD approaches may be partly due to the fact that method assessment efforts still mostly revolve around method *technological features* (such as separation of concerns, the availability of

tools or artifacts traceability, to name a few) while paying little attention to the *developers' attitudes and perceptions* of the method. Being software development a human activity, it is these perceptions (which go beyond what technological elements can explain) what determines the final acceptance of the method and, ultimately, of the paradigm. As Moody [28], actual efficacy (whether the method improves performance of a task) is only one of the -at least- two dimensions of success that need to be considered in evaluating software design methods, the other one being adoption in practice (whether the method is likely to be used in practice). Otherwise stated, regardless of its performance, unless a method is used in practice, its benefits cannot be realized. This need to understand the role of people in the method adoption process is widely accepted by the Software Engineering (SE) community [29], and has had it reflection in the use and adaptation of some well-known theoretical technology adoption models in the discipline.

In this paper we present an empirical study that, based on a tailored method adoption model, studies the variables that may impact the actual usage of methods through three representatives of the three aforementioned paradigms (code-centric, MBD and MDD). Contrary to other studies, the intra-subject design used in this study favours the detection of perceived advantages-disadvantages of each approach with respect to the others, putting the measures in context. The fact that, after the experiment, the users had to choose one of the three methods to work with it during the rest of the project mimics the decision process that developers and project managers take on a daily basis at the workplace.

The paper is structured following the reporting guidelines for controlled experiments in SE [30] as follows: Section 2 presents the most relevant theories that may help to explain the software method adoption process. These constitute the theoretical framework of our experiment, and they set the context for the definition of goals, hypotheses and variables in Section 3, together with the experimental design (subjects, instrumentation, operation and data collection mechanisms) and the experiment threats to validity. In section 4 we discuss the main findings of the study, and how they diverge from our original hypotheses. Section 5 presents complementary research to our work. Last, section 6 concludes the paper and outlines some further lines of research.

2 Conceptual Model

As aforementioned, there exist a number of models that establish the determinants of user technology acceptance. Among them, TAM [31], TAM2 [32], PCI [33], TPB [34] or MPCU [35] outstand, since they are theoretically grounded.

A comparative study of the fit of these models, initially devised to assess individual's intention to use a given information technology tool, to predict intention to adopt a given method [36] proved that some of the dimensions defined in those models were useful to predict method adoption (namely Usefulness, Compatibility, Subjective norm and Voluntariness), while others, given the differences between adopting a method and a tool, could be dropped (e.g. Career Consequences, Perceived behavioral control, etc.). This study also dismissed the impact of Ease of Use on

method adoption. This is surprising, since Ease of use is one of the main components of the TAM model and regarded as highly correlated with intention of use in other well-known method adoption theoretical models [27,28,37,38]. Given the controversy of this dimension, we have decided to include Ease of Use as a potentially explainer of method adoption variability. Also, in [27,37,38] tool performance (also called Perceived Tool Maturity) was included as a potentially relevant variable to explain both actual use and future intention of use of MDD methods. However, this dimension was found not significant in their studies, nor was considered relevant in any of the seminal theoretical models studied. For this reason, we have kept it out of our conceptual model.

Summarizing, our conceptual model includes the following explaining variables:

- Usefulness: extent to which the person thinks that using the method will enhance his or her job performance. The more useful a method is regarded by developers, the more likely they are to form intentions to use it.
- Ease of use: the degree to which a person believes that using a particular method would be free of effort. The easier developers believe the method to be, the more likely they are to adopt it.
- Compatibility: degree to which an innovation is perceived as being consistent with the existing values, needs and past experiences of potential adopters. The more compatible a method is with how developers perform their work, the more likely they are to form intentions to use it.
- Subjective norm: degree to which people think that others who are important to them think they should perform the behavior. The more people think that others who are important to them think they should use the method, the more likely they are to form intentions to use it.
- Voluntariness: the extent to which potential adopters perceive the adoption decision to be non-mandatory. The more voluntary the users regard the adoption decision to be, the less likely they are to adopt it.

Figure 1 depicts the components of our conceptual model. In this figure we have stressed the components and influences actually put to test by our experiment, whose context implied a constant subjective norm and voluntariness across the methods used. We will further dive into this issue in Section 3.

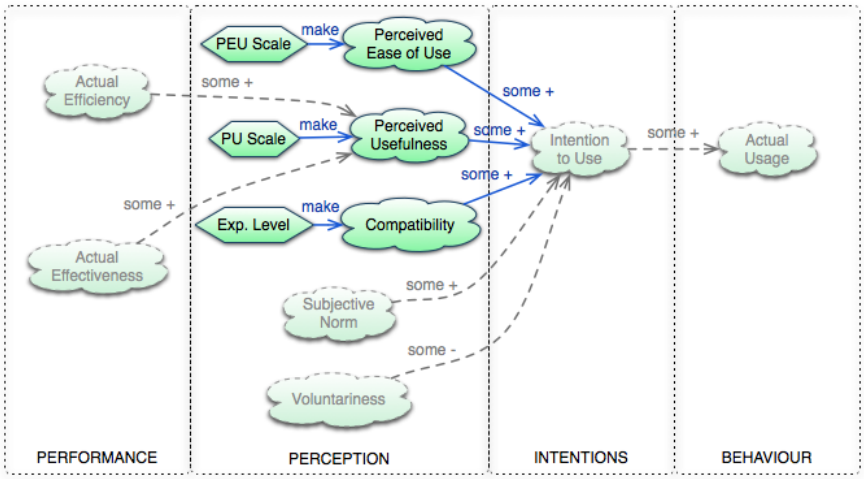

Figure 1: Synthesized method adoption conceptual model

3 Description of the Experiment

During the months of January and February 2011, an experiment was conducted at the Alicante University. The goal was to evaluate the method adoption intentions of users after developing a web application using three different approaches: a code-centric approach (based on C# and the .NET framework), MBD based on UML (supported by the Rational Software Modeler tool) and MDD based on OOH4RIA approach [39] (supported by the OOH4RIA IDE tool).

3.1 Goals and context definition

Following the GQM template [40], our empirical study is aimed at *Analyzing* a code-centric, an MBD and an MDD approach *for the purpose* of evaluating *with respect* to its adoption intentions *from the point of view* of junior software developers. The context of the study was a set of M.Sc. students developing the business layer of a web application.

The design of this experiment was based on the framework for experimentation in SE research suggested by [40]. The whole data set is included in the replication package available at http://www.dlsi.ua.es/~ccachero/labPackages/MethodIntention2Adopt.v1.rar. This study is based on the theoretical method adoption model presented in Section 2, and compares three methods, one representative of the code-centric paradigm, one representative of the MBD paradigm and one representative of the MDD paradigm.

The research questions addressed in this study were formulated as follows:

- RQ1: Is the developer's perceived usefulness of the method significantly different among methods, regardless of the particular application?
- RQ2: Is the developer's perceived ease of use of the method significantly different among methods, regardless of the particular application?
- RQ3: Is the developer's compatibility with the method significantly different among methods, regardless of the particular application?
- RQ4: Are the Perceived Usefulness (PU), Perceived Ease of Use (PEU) and Perceived compatibility (PC) measures correlated?
- RQ5: After the experience using the three approaches to development (code-centric, MBD and MDD), do the developer's adoption intentions significantly differ with respect to them, regardless of the particular application being developed?
- RQ6: Which are the main perceived advantages/drawbacks of each method?

The first four research questions were devised to be answerable by quantitative means, while the fifth research question was exploratory (no formal scale for measuring the IA was devised) and the sixth one, qualitative in nature, was aimed at gathering some possible explanations for the quantitative results.

Subjects and application

The subjects were 30 students of the Web Applications Developer Master at the University of Alicante. These students were divided in six groups of 4 to 6 people. From them, one group did abandon the experiment due to two of their components abandoning the Master for work reasons, so the final set of observations corresponds to the observations of the remaining five groups (26 subjects). Since the abandonment of the experiment had nothing to do with the treatments that the group was applying to his project nor the particular order in which they were applying them, we can assume that the results of the experiments have not been compromised.

The final sample comprised 25 men and 1 woman, of whom 75% had more than 2 years of experience developing web applications. The mean age of the participants was 25,6 years old and all of them were Computer Engineering graduates of the University of Alicante.

Each group developed a module for a different domain (travel, events, hospitals, academics and facework), although all the applications shared the application type (a social network) and the complexity (which was controlled by defining a range of functional requirements that all the applications had to support regardless of the domain). From them, the three functional requirements that were included in our experiment were:

- Support for the establishment of a community of users (from now on Group) to create contents and relationships among people of different environments (professional, personal, etc.).
- Support for the organization of events (from now on Events) where people can invite their friends or colleagues to attend to a place where is realized a celebration, a work meeting, etc.

- Support for an organizational section where companies, celebrities, etc. can publish content, photos, etc. in a unidirectional way to the social network community (from now on Organization).

Each one of these functional requirements was designed as a module. The subjects were asked to implement each module following a different method. The order in which students applied each method was randomized to avoid order effects. After the experiment, the subjects were asked to choose their preferred approach out of the three that they had used during the experiment in order to develop the remaining modules (functional requirements) of the project.

In order to develop the different projects, the students had to follow the Agile Unified Process (Agile UP) methodology [6] a streamlined approach to software development that is based on the IBM's Rational Unified Process (RUP). The Agile UP lifecycle is serial in the large, iterative in the small, and delivers incremental releases over time. Specifically, our experiment was situated in the construction phase of Agile UP, which is focused on developing the system to the point where it is ready for pre-production testing. The construction phase is made up of a set of disciplines or workflows that groups different tasks of this process. These disciplines, together with the impact of modelling practices on each of them depending on the paradigm, were presented in Table 1.

Implementation Language and CASE Tools

The development environment for the experiment was set up as follows:

- Development framework: .NET framework, Silverlight 4.0 and NHibernate (Object-Relational Mapping).
- IDE (Integrated Development Environment) Development Tool: Visual Studio 2010.
- Languages: C# and XML Mapping (ORM mapping of NHibernate).
- Other tools: The set of questionnaires filled in by each developer were published in http://www.surveymonkey.com/

The code-centric treatment relied solely on the coding tools provided by this environment. The MBD treatment also required the students to work with RSM. Last but not least, for the MDD treatment the students worked with the OOH4RIA IDE. Both RSM and OOH4RIA IDE are based on the Eclipse Modeling Project [41]. Eclipse is a development of open source software whose main purpose is to provide a highly integrated platform tools [42]. According to [43], Eclipse has contributed to the successful implementation of the Model Driven Architecture (MDA, which is the OMG standard for MDD [44]) providing an open source platform and a whole implementation of the MDA specifications. The University of Alicante uses both RSM and OOH4RIA IDE in SE undergraduate and graduate studies, and therefore all the students were previously familiarized with them.

To standardize the code that had to be developed, the subjects had to implement/generate four specific files: the Business Entities Component file (BEC), the Data Access Component file (DAC), the Data Transfer Component file (DTC) and the Data Base file (DB).

3.2 Experiment Planning

As shown in Figure 1, the idiosyncrasy of the experiment made some of the dimensions of our initial theoretical model non relevant; namely, voluntariness and subjective norm did not apply to our course environment, where the three paradigms were equally valued and no obligation whatsoever was made about the method the students had to use once the experiment was finished. This lack of relevance has been outlined in Figure 1 by showing the corresponding dimensions in a lighter shadow of grey.

Given the evidences gathered by our theoretical model and the research questions presented in Section 2, we have defined the following Independent (experimentally manipulated) Variables (IV) or factors:

- Meth: Method, a categorical variable with three levels: code-centric, MBD, MDD.
- App: Application, a categorical variable with five possible values: Travels, Hospitals, Events, Academics, Facework.

The dependent (measurable) variables (DV) are:

- PU: Perceived usefulness of each method, an interval measure based on a 7-point Likert scale.
- PEU: Perceived ease of use of each method, an interval measure based on a 7-point Likert scale.
- PC: Perceived compatibility of each method, an interval measure based on a 7-point Likert scale.
- IA: Intention to Adopt a given method: a nominal measure with three possible values: code-centric, MBD, MDD.

Also based on the research questions, we defined the following testable hypotheses:

- HPU: PU(MDD)>PU(MBD)>PU(code-centric): Perceived usefulness of MDD methods is greater than perceived usefulness of MBD methods, which in turn is perceived as more useful than code-centric approaches for development tasks. This fact holds regardless of the actual application developed. (RQ1).
- HPEU: PEU(MDD)<PEU(MBD)<PEU(code-centric): Perceived ease of use of MDD methods is lower than perceived ease of use of MBD methods, which in turn is perceived as more difficult to use than code-centric approaches for development tasks. This fact holds regardless of the actual application developed. (RQ2).
- HPC: PC(MDD)<PC(MBD)<PC(code-centric): Compatibility of MDD methods is lower than Compatibility of MBD methods, which in turn is perceived as less compatible than code-centric approaches for development tasks. This fact holds regardless of the actual application developed. (RQ3).
- HCorr: PC is positively correlated with PU and PEU (RQ4)

In order to test the hypotheses, we defined the following measuring instruments:

- The PU, PEU and PC DV were measured through questionnaires.
- The perceived usefulness PU(Meth) was assessed through a 7-point Likert scale that consisted of four items: subjective developer's throughput (with

respect to an expert), subjective developer's efficiency, subjective utility of the method and subjective reliability of the results obtained from applying the method.

- The perceived ease of use PEU(Meth) was assessed through a semantic-differential scale that required developers to judge the development method on 8 pairs of adjectives describing their experience. Four adjectives were formulated in positive and four in negative to control a possible acquiescence bias. Developers could modulate their evaluation on 7 points (after recoding of reversed items 1 = very negative, 7 = very positive).
- To measure PC(Meth) we defined a two-item scale, made up of a 7-point rating of familiarity (1 = very odd, 7 = very familiar), and of level of previous experience (1=very low 7=very high) with the techniques and tools involved in code-based, MBD and MDD development respectively.

The IA DV was measured indirectly through a decision, made by the developers, on which method to use for the rest of the project, once the experiment was finished. Last but not least, the perceived advantages/disadvantages were asked through three open questions, one for each method.

3.3 Data Analysis and interpretation of results

The statistical analysis was carried out with the PASW (Predictive Analytics SoftWare) Statistics software [45].

Prior to the assessment of the hypotheses, we checked the reliability of the PU, PEU and PC scales in the context of our experimental settings. We applied the Alpha of Cronbach test, which revealed the following results:

- For the PU scale, all the items showed a correlation higher than 0.3, while the global Cronbach alpha was 0.817, giving proof of a sufficient internal consistency among the PU items. Subsequently we can calculate the mean and consider this mean as a global rating of PU with each one of the three treatments (code-centric, MBD, MDD).
- For the PEU scale, all the items showed a correlation higher than 0.3, while the global Cronbach alpha was 0.896, giving proof of a high reliability of the scale. Again, this means that we can calculate the mean and consider this mean as a global rating of PEU with each one of the three treatments (code-centric, MBD, MDD).
- For the PC scale, we found very low levels of correlation (although significant) between the developers' perception of a method as 'odd' and their level of experience reported with such method. Therefore, for the measurement of the PC on subsequent analyses, we opted to rely solely on the level of experience reported, since, from our point of view, it more clearly reflects the definition of Compatibility given in our theoretical model.

RQ1: Perceived usefulness of the development approaches

To test the HPU hypothesis (concerning the existence of significant differences in the perceived usefulness of the different methods), we applied a 3*5 Mixed Design ANOVA [46], in which the application (Trips, Events, etc.) was the between-subjects variable, and the PU ratings for each method was the within-subjects variables.

In order to assure that applying this statistical method made sense, we first checked the homogeneity of covariance among groups with the Box's M test (F=1,137, p=0,295). Also, we verified that the principle of spherity was not violated by applying the W Mauchly's test (W=0,909, p=0,387).

The results showed that MDD produced the highest PU (M = 4,90), followed by code-centric (M=4,51) and then MBD (M=3,48). The results also showed that the interaction application*method was not significant (F(8,42) = 1,919, p>0.05). We can then safely examine the main effects of the two independent variables (application and method) on these means without needing to qualify the results by the existence of a significant interaction. The main effect of application did not attain significance (F(4,21)=1,126, p=0,371), but the main effect of method did reach significance, (F (2, 42)=19,411, p<=0,01), that is, *the differences in PU are significantly affected by the method used, regardless of the particular application being developed.*

Given the significance of the method, the last step of the analysis consisted on studying the pair wise differences among methods through a matched T-test. In order not to augment the risk of a type-1 error, a Bonferroni adjustment was applied. This means reducing the significance threshold to 0.0167 (p = 0.05 / 3 = 0,0167). With this adjustment, the PU differences between MBD and both code-centric (t=4,05, p<0,001) and MDD approaches (t=5,24, p<0,001) were significant, while the difference between code-centric and MDD PU scores was not.

We can graphically observe these results in Figure 2. The fact that the particular application was not significant is reflected in the five lines more or less overlapping. The Method variable influence is reflected in the acute ups and downs of the lines. Finally, the method*application interaction lack of significance is reflected in the lines being more or less parallel (all of them showing the same tendency with each method). The same graphical clues hold for the remaining graphics.

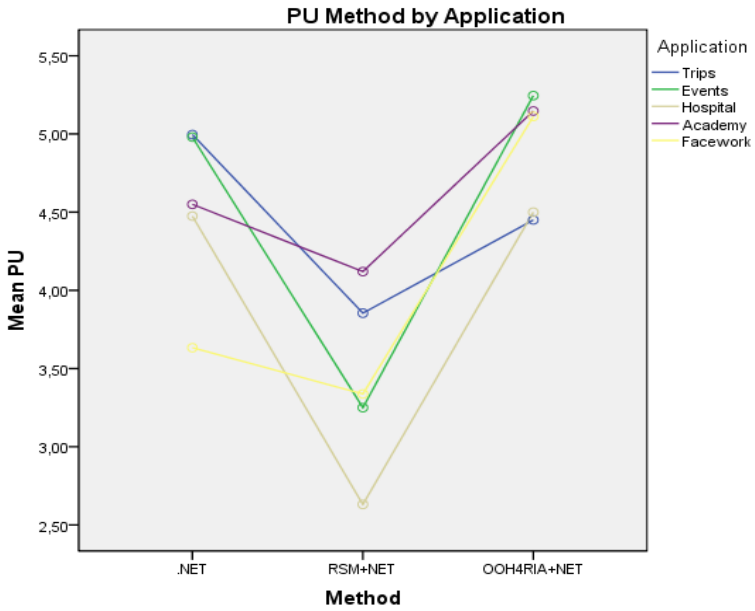

Figure 2: PU means by method. Each line corresponds to one of the five applications that were developed as part of the experiment.

RQ2: Perceived ease of use of the development approaches

We then tested the HPEU hypothesis related to the degree of perceived ease of use of the different methods. Again, the chosen procedure was a 3*5 Mixed Design ANOVA [46], in which the application (Trips, Events, etc.) was the between-subjects variable, and the PU ratings for each method was the within-subjects variables.

In order to assure that applying this statistical method made sense, we first checked the homogeneity of covariance among groups with the Box's M test (F=0,768, p=0,78). Also, we verified that the principle of sphericity was not violated by applying the W Mauchly's test (W=0,877, p=0,269).

The results showed that MDD produced the highest PEU (M = 4,87), followed by code-centric (M=3,98) and then MBD (M=3,55). The results also showed that the interaction application*method is significant ($F(8,42) = 2,801$, $p<0.05$). This means that the results vary differently depending on the particular application being developed.

Also, the results showed that, while the main effect of application did not attain significance ($F(4,21)=1,033$, $p=0,414$), the main effect of method did reach significance, ($F(2,42)=24,704$, $p<0,01$), that is, *the differences in PU are significantly affected by the method used, regardless of the particular application being developed. This result, however, must be interpreted cautiously, because the effect of the method is different in the different applications.*

We can graphically observe these results in Figure 3. Here, it can be observed that the significance of the method by application interaction is due to the Academic application, whose MBD PEU follows a completely different trend than the remaining applications; that it can be explained by the previous experience of the subjects, in the use of RSM, in their degree studies.

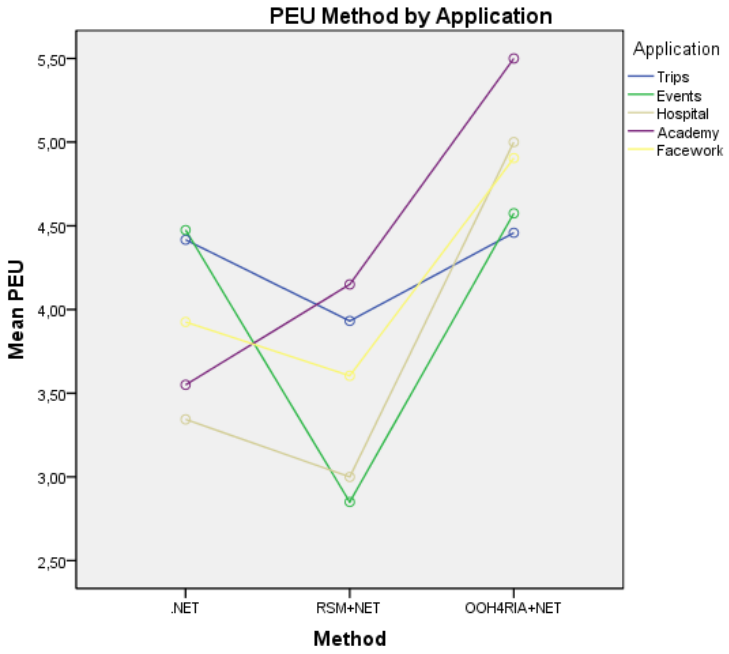

Figure 3: PEU means by method. Each line corresponds to one of the five applications that were developed as part of the experiment.

Given the significance of the Meth variable, a study of the follow-up comparisons among methods was performed through a matched T-test with the Bonferroni adjustment. The results show that the PEU differences between MDD and both code-centric ($t=-4{,}029$, $p<0{,}001$) and MBD approaches ($t=-5{,}67$, $p<0{,}001$) were significant, while the difference between code-centric and MBD PEU scores was not ($t=1{,}949$, $p>0{,}05$).

RQ3: Perceived compatibility of the development approaches

To test the HPC hypothesis (concerning the existence of significant differences in the perceived compatibility of the different methods), we dropped the inter-subject factor and applied a one-way RM Anova, under the premise that, having all the applications the same set of functional requirements and belonging to the same domain, the method compatibility (the degree to which the method is perceived as

being consistent with the existing values, needs and past experiences of developers) did not depend on the particular application being developed.

On a 3-point scale (1=low, 2=medium, 3=high), developers rated the method based on MBD as slightly the most compatible (M=2,12) followed by the code-based method (M=2) and, finally, the method based on MDD (M=1,54). This compatibility was exclusively based on their previous experiences with the methods.

A check of the spherity threw a significant W (W=0,471, p<0,001), which made us test the significance of the differences with a conservative Greenhouse-Geisser F. The results showed that the main effect of method did reach significance (F(1.3,50)=15,492, p<0,001), that is, *the differences in PC are significantly affected by the method used*.

A follow-up pair wise analysis with a t-test with a Bonferroni adjustment showed that MDD is perceived as significantly less compatible than code-based (t=3,33, p=0,003) and MBD (t=5,09, p<0,001), while differences between code-based and MBD are not significant (t=-1,806, p>0,05).

RQ4: Correlation among conceptual model dimensions

In order to study the correlation among the three conceptual model dimensions included in our study, anon-parametric Sperman correlation analysis was performed. The results of this analysis can be seen in Table 2.

		PU_Global	PEU_Global	PC_Global
PU_Global	Rho Spearman	1,000	,663**	-,278*
	P	.	,000	,014
	N	78	78	78
PEU_Global	Rho Spearman	,663**	1,000	-,288*
	P	,000	.	,011
	N	78	78	78
PC_Global	Rho Spearman	-,278*	-,288*	1,000
	P	,014	,011	.
	N	78	78	78

Table 2: Correlations among theoretical model dimensions (PU, PEU, PC)

The results show how PC is negatively correlated with both PEU and PU, which means that, in this experiment, *developers regarded the development methods with which they were less familiar as more useful and easy to use*. These results are hard to explain and contradict the theoretical model, and need further investigation with a more elaborated PC measurement scale.

Also, we can observe how, according to our data, PU and PEU are strongly correlated: the more useful the method is perceived, the more easy to use (and vice versa). Being the two variables dependent variables in our experiment, we cannot say anything about whether there is a causality nor in which direction. More specific research needs to be carried out in order to ascertain these aspects of the theory, and, ideally, come up with more elaborate constructs that are independent from each other.

RQ5: Intention to adopt analysis

When, after the experiment, our subjects were asked to choose a given method to go on with the project, 20 subjects out of 26 (that is, 76,9% of the sample) decided to use the MDD approach.

RQ6: Perceived advantages and disadvantages of methods

For a better understanding of previous objective results, we complemented our study with an opinion survey in which developers reported in a free-form format their main perceived advantages and disadvantages of each method. After gathering the results, and in order to organize the natural language responses, we classified each advantage/disadvantage under a common epigraph (see the first column of Table 3). Moreover, for a given method, these features could be reported either as an advantage or a drawback.

Starting with the traditional code-centric approach (see second column of Table 3), we can observe that 50% of subjects perceived the feeling of control that they experienced when working with code as one important advantage of the code-centric approach, while none regarded that as a disadvantage. In an order of decreasing size, the rest of advantage responses are distributed into lower Learning curve (5), higher Personalization (4) and higher Compatibility with respect to their previous experiences (3). At the other end of the spectrum, according to subjects, the main drawback of code-centric is that it requires a high development effort (with 20 subjects giving responses falling in that category) and has a low maintainability (5 subjects).

Regarding the MBD approach using RSM (see third column of Table 3), the advantages of the method greatly varied. The most important ones were a reduced development effort (10), a better maintainability (6) and a reduced learning curve (5). However, these advantages are not universally appreciated as can be seen is we observe how development effort and reduced learning curve were signaled as disadvantages by 5 and 3 subjects, respectively. Also, subjects noted that the code generation capabilities provided by RSM were clearly insufficient (5 subjects).This noted, the main MBD reported drawback was a significant lack of perceived reliability of the approach (12 responses), due to some detected errors in the RSM code-generation process.

Last but not least, we asked the same question regarding the OOH4RIA approach as an exponent of the MDD paradigm. The main reported disadvantages of the method were a higher learning curve (6), a low compatibility with their previous

experience (5), probably due to the adaptation of a new development style, and a lack of reliability (7), which may have been due to some bugs of the OOH4RIA IDE tool that appeared during the development phase. However, an overwhelming majority of students (24 out of 26) recognized an important increase in development speed and an important effort reduction with repetitive tasks. Also maintanability was regarded as an advantage by 3 subjects.

Reported Feature		Code-based	MBD	MDD
Development Effort	Advantage	1	10	24
	Drawback	20	5	1
Feeling in control	Advantage	13	-	-
	Drawback	-	-	3
Learning Curve	Advantage	5	5	2
	Drawback	1	3	6
Compatibility	Advantage	3	-	-
	Drawback	-	2	5
Maintainability	Advantage	-	6	3
	Drawback	5	-	-
Personalization	Advantage	4	-	-
	Drawback	-	1	-
Reliability	Advantage	-	-	-
	Drawback	-	12	7

Table 3: Reported advantages/drawbacks of each method

3.4 Threats to Validity

The threats to validity evaluate under which conditions our experiment is applicable and offers benefits, and under which circumstances it might fail. Four families of threats to validity are reported by Cook and Cambell [47]: internal, external, construction and conclusion.

3.4.1 Threats to Conclusion Validity

They refer to the relationship between the treatment and the outcome. All the statistical analyses have been preceded by tests that assured that the assumptions of the statistical procedure were not being violated. The alpha has been adjusted following the most conservative approach (Bonferroni), which also contributes to the

conclusion validity. The intra-subject design also protects the results against aleatory heterogeneity of subjects, since all subjects were presented with the three methods. This notwithstanding, given the duration of the different treatments (each subject was working with each method for over a fifteen days) random irrelevancies in the experimental setting might have occurred that may have affected the data. We can assume however that such irrelevancies have affected all the levels of the treatment equally.

3.4.2 Threats to Internal Validity

They are concerned with the possibility of hidden factors that may compromise the conclusion that it is indeed the treatment what causes the differences in outcome. In order to increase the internal validity of the design, subjects received similar training in all three methods. Since all treatments were applied by all subjects in random order(with only one order being left of the experiment due to a group dropping the master soon after the experiment began), there was no reason for compensatory rivalry, equalization or demoralization. All the applications and modules on which the subjects worked were of similar complexity. The lack of influence of the particular application on the results was statistically tested. This notwithstanding, given the relatively small sample used (26 subjects), and the fact that the sample was not random, but chosen based on their participation in a master degree, our experiment still presents a threat to internal validity that can only be overcome with future replications. Also, even if we counterbalanced the design by randomizing the order in which the different subjects used the different methods, subjects still applied the three methods in a row. This makes possible the appearance of carry-over effects (e.g. subjects could benefit as the experiment takes place of an increased familiarity with the experiment setting, the development environments, etc.) or, on the contrary, fatigue effects (e.g. through a loss of interest in the registration of the experiment data). Also, it is possible the diffusion of treatment imitation effect, that is, users learning from the previous method and imitating the practices when applying the next method assigned. In order to diminish such risks, we supervised the whole process, and maintained the amount of information that needed to be provided by the users to a minimum. We also automated the data gathering process to prevent coding errors.

Threats to Construct Validity

They refer to the relationship between theory and observation. During the experiment, both the theoretical model and the hypotheses were carefully kept from the subjects. No special emphasis was made over the pros and cons of any of the methods until after the experiment was finished and the opinions gathered. The particular technologies used to test each development approach are widely used in practice.

However, to our knowledge extent there are not standard instruments to measure the components of the theoretical model applied in this article. Although the reliability of the scales has been checked to increase the construct validity, the

number of items of some of the used scales should be definitely increased and validated through further research to become more robust.

Threats to External Validity

Last but not least, threads to external validity are concerned with the generalization of the results. The type of application used for the experiment, far from being a toy example, is a real application, defined based on true client requirements. The subjects are graduate students, many of them already working as developers and therefore true representatives of junior developers. This notwithstanding, we have used similar size applications, all of the same type (social networks). Therefore we cannot generalize the results to applications of different sizes or different application types without more replicas.

4 Discussion

This study was designed under the premises that the use of models improved the developers' perceived usefulness of the method, although it somehow made the development process more difficult. Also, it aimed at confirming/refuting that code-centric approaches were regarded by developers as more compatible with their current practices.

It has been therefore somehow surprising to check the extent to which using models in the context of an MBD environment is regarded as useless and difficult, even if the subjects did not show a great degree of incompatibility with the approach (probably due to all of them being junior developers, with an extensive training in modeling). We think that the data gathered in this study should be regarded as a warning for software engineering trainers, who should aim at teaching modeling techniques in the context of MDD environments if they aim at increasing the perceived value of the modeling activities.

As expected, the MDD method was regarded as the more difficult to grasp but the more useful in the long run. This data seems to back the claim made in [36], where it was assessed that usefulness had a much greater impact in intention to use a method than ease of use. In fact, the experiment showed how 20 out of the 26 subjects chose the MDD method to continue with the project development, although we cannot split the effect of the paradigm from the effect of the particular MDD approach (OOH4RIA) that was used in the experiment. This coupling between method and tool makes especially necessary the replication of the experiment with other, both commercial and academic, environments, to assess whether these results can be generalized.

If we take a look at the reported advantages/disadvantages that may explain this decision, it seems that, for MDD, a lower development effort compensated some perceived disadvantages such as greater learning curve, lower compatibility or even lower reliability, due to code generation errors. This in turn may suggest that, in our theoretical model about intention to use, perceived usefulness should be weighted as

having more importance than perceived ease of use or compatibility. However, much more data is necessary to corroborate or dismiss this preliminary evidence.

Last but not least, the correlation analysis performed on the dimensions of the theoretical model open up several interesting further lines of research. Notably, the fact that PC is negatively correlated with PU and PEU poses some doubts about the relevance of compatibility in a software development method adoption model as an independent dimension, at least in the context of junior software developers. Are software developers technology geeks, and therefore attracted rather than repelled by new, unknown development environments and practices? Are modeling practices, at least within a MDD paradigm, experienced as so advantageous that they shadow the possible effect of this variable? All these are doubts that require from further experimental research. Also, these correlations suggest that perhaps the theoretical model needs to be re-structured to include more independent dimensions that can increase the reliability of further analyses methods such as linear regression models or Bayesian networks, to name a few.

5 Related Work

In the last year we have witnessed how the number of empirical studies regarding the subjective perceptions while applying different methods has increased. Navarrete & Ignacio [48] empirically assessed the satisfaction of an MDD method (called MIMAT) that includes Functional Usability Features (FUFs) in an MDD software development process. The study concluded that the user's satisfaction improves after including FUFs in the software development process. Our experiment does not center on the enrichment of a given method with a new artifact/technique, but compares different methods with respect to not only the developer's perceived ease of use but also the perceived usefulness and compatibility.

Closer to our experiment, Arisholm et al. in [49] reports on controlled experiments, spanning across two locations: Oslo (Norway) and Ottawa (Canada), which investigate the impact of UML documentation on software maintenance. Results show that, for complex tasks and past a certain learning curve, the availability of UML documentation may result in significant improvements of the functional correctness of changes as well as their design quality. Another recent experiment [50] compares three treatments or levels in the use of UML models (no modeling conventions, with modeling conventions and tool-supported modeling conventions). The experiment was performer with 106 MSc students organized in 35 teams. The findings shows that development effort with tool-supported modeling conventions is approximately half of the effort invested when using traditional development (*code-centric*). It also showed that there is no significant correlation between class-count and the effort spent in modeling. Our experiment does not center on effort but on intention to adopt the method.

There are also studies that center on the set of tools that accompany each paradigm: Pelechano *et al.* [51] performed an empirical comparison of Eclipse Modeling Plug-ins and Microsoft DSL Tools on the basis of its utility and satisfaction from the point

of view of developers. The result showed that both tools are very useful and can be used in future projects. Eclipse users are 100% faithful to this environment; however 60% of the DSL Tools users would migrate to Eclipse. Also [52] performs a similar comparison between Microsoft DSL Tools and Eclipse EMF/GEF/GMF Frameworks. Results show that the MS/DSL Tools metamodel designer is more usable than the EMF metamodel designer. Our study is less focused on tools and more focused on development paradigms.

6 Conclusions and further lines of research

This study makes three main contributions. On the one hand, it presents a tailored theoretical method adoption model that, being based on well known evidence from different fields, can be used as a starting point to study the adoption possibilities of existing or new development methods. On the other hand, it presents an empirical comparison of the perceived usefulness, ease of use and compatibility of three methods that in turn are clear exponents of the three mainstream development paradigms nowadays: code-centric, MBD and MDD. In this sense, the study concludes that MDD approaches are the more difficult to use but, at the same time, are regarded as the more useful in the long run. Also, it shows that junior developers feel that modeling techniques are compatible with their background and experiences. Last but not least, to our knowledge extent, this study provides the first set of evidence on the perceived strengths and weaknesses of the different paradigms with respect to their adoption intent.

All contributions open new lines of research. Regarding the theoretical model, only three out of the five initial components have been studied. Also, the correlations found among the three components suggest that it may be advisable to refine the model and the measurement instruments to diminish the co-linearity of the variables and be able to fine-tune the prediction power of the model. Such prediction power has not been formally tested, and remains another line of work. Regarding specific methods, further experimentation is needed to be able to generalize the results to a different population, different methodologies (e.g. agile) and languages, different application types or different application sizes. The perceived advantages and disadvantages of the methods and its impact in the final decision to adopt them give some clues to make decisions about the prioritization of improvements that need to be made to the development environments.

References

[1] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2004.

[2] K. Schwaber and M. Beedle, *Agile software development with Scrum*, Prentice Hall Upper Saddle River, NJ, 2002.

[3] I. Jacobson and S. Bylund, *The road to the unified software development process*, Cambridge Univ Pr, 2000.

[4] M. Eichberg, M. Monperrus, S. Kloppenburg, and M. Mezini, "Model-Driven Engineering of Machine Executable Code," *Modelling Foundations and Applications*, 2010, pp. 104-115.

[5] R. Picek, D. Grabar, N. Vrcek, D. Plantak Vukovac, B. Kos, M. Bratko, A. Lovrenčić, and others, "Suitability of Modern Software Development Methodologies for Model Driven Development," *Journal of Information and Organizational Sciences*, vol. 33, 2009.

[6] S.W. Ambler, "Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process," *John Wiley&Sons*, 2002.

[7] S. Lahtinen and J. Peltonen, "Enhancing usability of UML CASE-tools with speech recognition," *Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on*, 2004, pp. 227-235.

[8] No Magic Inc., "MagicDraw UML," *(Last update april 26, 2010)*, 2006.

[9] Gentleware, "Poseidon for UML Community Edition 8.0," 2009.

[10] VisualParadigm, "Visual Paradigm for UML," *http://www.visualparadigm.com/*, 2011.

[11] IBM Corporation, "Rational Software Modeler," *(Last upadate September 2, 2008)*, 2008.

[12] evolving-software, "TrueView Domain Modeller," *http://www.evolving-software.co.uk/*, 2011.

[13] Borland Together, "Together - Visual Modeling for Software Architecture Design," *http://www.borland.com/*, 2011.

[14] S. Meliá, J.J. Martínez, S. Mira, J. Osuna, and J. Gómez, "An Eclipse Plug-in for Model-Driven Development of Rich Internet Applications," *Web Engineering*, 2010, pp. 514-517.

[15] M. Brambilla, S. Comai, P. Fraternali, and M. Matera, "Designing web applications with WebML and WebRatio," *Web Engineering: Modelling and Implementing Web Applications*, 2008, pp. 221-261.

[16] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, "Developing applications using model-driven design environments," *Computer*, vol. 39, 2006, pp. 33-40.

[17] R. Picek and V. Strahonja, "Model Driven Development-future or failure of software development," *IIS*, 2007, pp. 407-413.

[18] P. Mohagheghi and V. Dehlen, "Where is the proof? - A review of experiences from applying MDE in industry," *European Conference on Model Driven Architecture--Foundations and Applications (ECMDA 2008)*, 2008, pp. 432-443.

[19] M. Guttman and J. Parodi, *Real-life MDA: solving business problems with model driven architecture*, morgan kaufmann, 2006.

[20] T. Stahl, M. Völter, J. Bettin, A. Haase, S. Helsen, and K. Czarnecki, "Model-Driven Software Development: Technology, Engineering, Management," *John Wiley and Sons, ISBN:978-0-470-02570-3*, 2006, pp. 970-978.

[21] E. López, M. González, M. López, and E. Iduñate, "Proceso de Desarrollo de Software Mediante Herramientas MDA," *Revista Iberoamericana de Sistemas, Cibernética e Informática*, vol. 3, 2006, pp. 6-10.

[22] H. Gustavsson, B. Lings, B. Lundell, A. Mattsson, and M. Beekveld, "Integrating proprietary and open-source tool chains through horizontal interchange of XMI models," *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, 2007, pp. 521-522.

[23] W. Heijstek and M.R. Chaudron, "Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process," *Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on*, 2009, pp. 113-120.

[24] S.J. Mellor, T. Clark, and T. Futagami, "Model-driven development: guest editors' introduction.," *IEEE software*, vol. 20, 2003, pp. 14-18.

[25] J. Muñoz and V. Pelechano, "MDA vs Factorías de Software," *Actas del II Taller sobre Desarrollo de Software Dirigido por Modelos, MDA y Aplicaciones (DSDM 2005)*, 2005, p. 1.

[26] A.G. Kleppe, J. Warmer, and W. Bast, *MDA explained: the model driven architecture: practice and promise*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[27] S. Walderhaug, M. Mikalsen, I. Benc, and S. Erlend, "Factors affecting developers' use of MDSD in the Healthcare Domain: Evaluation from the MPOWER Project," *From Code Centric to Model Centric Software Engineering: Practices, Implications and ROI. Workshop at European Conference on Model-Driven Architecture*, 2008.

[28] D.L. Moody, "The method evaluation model: a theoretical model for validating information systems design methods," *11th European Conference on Information Systems (ECIS 2003), Naples, Italy*, 2003, p. 79.

[29] S.L. Pfleeger, "Understanding and improving technology transfer in software engineering," *Journal of Systems and Software*, vol. 47, 1999, pp. 111-124.

[30] A. Jedlitschka and D. Pfahl, "Reporting guidelines for controlled experiments in software engineering," *2005 International Symposium on Empirical Software Engineering, 2005.*, 2005, p. 10.

[31] F.D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS quarterly*, vol. 13, 1989, pp. 319-340.

[32] V. Venkatesh and F.D. Davis, "A theoretical extension of the technology acceptance model: Four longitudinal field studies," *Management science*, vol. 46, 2000, pp. 186-204.

[33] G.C. Moore and I. Benbasat, "Development of an instrument to measure the perceptions of adopting an information technology innovation," *Information systems research*, vol. 2, 1991, pp. 192-222.

[34] I. Ajzen, "The theory of planned behavior," *Organizational behavior and human decision processes*, vol. 50, 1991, pp. 179-211.

[35] R.L. Thompson, C.A. Higgins, and J.M. Howell, "Personal computing: toward a conceptual model of utilization," *MIS quarterly*, vol. 15, 1991, pp. 125-143.

[36] C.K. Riemenschneider, B.C. Hardgrave, and F.D. Davis, "Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 28, 2002, pp. 1135-1145.

[37] H. Fujita and I. Zualkernan, "Evaluating Software Development Methodologies based on their Practices and Promises," *New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Seventh Somet_08*, vol. 182, 2008, p. 14.

[38] P. Mohagheghi, "An Approach for Empirical Evaluation of Model-Driven Engineering in Multiple Dimensions," *C2M:EEMDD 2010 workshop- from Code Centric to Model Centric: Evaluating the Effectiveness of MDD*, CEA LIST Publication, 2010, pp. 6-17.

[39] S. Meliá, J. Gómez, S. Pérez and O. Díaz,. *A model-driven development for GWT-based rich Internet applications with OOH4RIA*. In proceedings of the ICWE 2008, Washington, DC, USA, IEEE Computer Society, 2008, pp. 13–23.

[40] C. Wohlin, P. Runeson, and M. Höst, *Experimentation in software engineering: an introduction*, Springer Netherlands, 2000.

[41] R.C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, Addison-Wesley Professional, 2009.

[42] F. Budinsky, *Eclipse modeling framework: a developer's guide*, Prentice Hall Ptr, 2004.

[43] A. Fernandez, S. Abrahão, and E. Insfran, "Towards to the validation of a usability evaluation method for model-driven web development," *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2010, pp. 1-4.

[44] O.M. OMG, *Model Driven Arquitecture*, 2003.

[45] SPSS Inc. an IBM CompanyHeadquarters, "PASW Statistics 18 - Content Guide," 2009.

[46] J. Pallant, *SPSS survival manual*, Open Univ. Press, 2005.

[47] T.D. Cook, D.T. Campbell, and A. Day, *Quasi-experimentation: Design & analysis issues for field settings*, Houghton Mifflin Boston, 1979.

[48] P. Navarrete and J. Ignacio, "Incorporación de mecanismos de usabilidad en un entorno de producción de software dirigido por modelos," 2010.

[49] E. Arisholm, L.C. Briand, S.E. Hove, and Y. Labiche, "The impact of UML documentation on software maintenance: An experimental evaluation," *IEEE Transactions on Software Engineering*, vol. 32, 2006, pp. 365-381.

[50] M. Chaudron, *Effective UML Modelling Does Software ModelingPay ? - empirical studies in UML*., 2011.

[51] V. Pelechano, M. Albert, J. Muñoz, and C. Cetina, "Building tools for model driven development comparing microsoft DSL tools and eclipse modeling plug-ins," *Proceedings of the 11th Conference on Software Engineering and Database (JISBD'06)*, 2006.

[52] T. Özgür, "Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the context of the Model-Driven Development," *School of Engineering. Ronneby, Sweden, Blekinge Institute of Technology*, 2007, p. 56.