Evaluating the Impact of a Model-Driven Web Engineering Approach on the Productivity and the Satisfaction of Software Development Teams

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

[1] Universidad Máximo Gómez Báez de Ciego de Ávila, Cuba
[2] DLSI. Universidad de Alicante, Spain

Abstract. BACKGROUND: Model-Driven Engineering claims a positive impact on software productivity and satisfaction. However, few efforts have been made to collect evidences that assess its true benefits and limitations.

OBJECTIVE: To compare the productivity and satisfaction of junior Web developers during the development of the business layer of a Web 2.0 Application when using either a code-centric, a model-based (UML) or a Model-Driven Engineering approach (OOH4RIA).

RESEARCH METHOD: We designed a full factorial, intra-subject experiment in which 26 subjects, divided into five groups, were asked to develop the same three modules of a Web application, each one using a different method. We measured their productivity and satisfaction with each approach.

RESULTS: The use of Model-Driven Engineering practices seems to significantly increase both productivity and satisfaction of junior Web developers, regardless of the particular application. However, modeling activities that are not accompanied by a strong generation environment make productivity and satisfaction decrease below code-centric practices. Further experimentation is needed to be able to generalize the results to a different population, different languages and tools, different domains and different application sizes.

1 Introduction

It is a well known fact that the Web Engineering community advocates the use of models in order to improve software development processes for Web applications. However, there are many issues around modeling that are, as of today, cause of controversy and heated debates: to which extent should practitioners model? Which should be the level of detail of these models? Should practitioners strive to maintain the models current, or should these models be disposable? These and others are open questions whose answer currently partly depends on the development culture of the person asked, and partly on the context in which such practices are being adopted. In this respect we claim that, instead, the decision about which is the adequate application of software modeling practices should be answerable based on objective data regarding its impact on well-known process and product quality dimensions. From these dimensions, productivity, defined as a ratio of what is produced to what is required to produce it, outstands, due to its impact during the selection of a development process in industry [1]. Also, satisfaction is an important aspect of quality, since, being software development a human process, the developer's satisfaction is a key factor for the successful adoption of such practices [2].

One quite well-known way of classifying modeling practices in industry is according to the extent to which modeling is used to support the development process. Fowler [3] describes three different modes in which modeling languages (and the 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. 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. They are most useful in code-centric approaches, where the objective is to develop self-explaining code.
- 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) practices, such as the ones promoted by frameworks such as the Rational Unified Process (RUP).
- Last but not least, models can be used to fully characterize the application. If such is the case, the diagrams replace the code, and they are compiled directly into executable binaries. This is the modeling use that lies at the core of Web Engineering Model-Driven Development (MDD) approaches.

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, from manual, code-centric, to full, Model-Driven [4]. While code-centric development methods require - at most - an informal use of modeling techniques and languages, both MBD and MDD require a formal use of models, which mainly relies on the use of Computer Aided Software Engineering (CASE) tools. These tools may offer not only modeling environments - including model checkers that may assure syntactical correctness, semantic accurateness, consistency or completion of models, to name a few desirable model characteristics - but also partial or even complete software generation environments that, in the case of MDD CASE tools, are based on model transformations.

Both MBD and MDD tools work under the assumption that designing models that can generate partial or complete code is much simpler and quicker than actually writing such code. This same view is sustained in MBD and MDD related scientific literature. Such literature claims that the two most outstanding advantages of MDD over code-centric or even MBD approaches are (a) short and long term process productivity gains [5] and (b) a significant external software product quality improvement [6,7]. 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 [8].However, neither the MBD nor the MDD research community have yet been able to provide practitioners with a sufficient body of practical evidence that soundly backs the purported gains of their recommended modeling practices with respect to code-centric approaches [9,10]. Many authors have written about the importance of providing empirical evidence in software engineering [11] but, unfortunately, the percentage of empirical studies - be them surveys, experiments, case studies or postmortem analyses [12]- that provide data to illustrate the impact of MBD and MDD approaches over different quality characteristics (such as productivity or satisfaction) is still very low, which hampers the generalizability of the results. In order to guarantee such generalizability, we also need to take into explicit consideration many factors that may affect these characteristics, such as tool usage, adaptation of the development methodology to the idiosyncrasy of the particular development team, type, complexity and size of the project, and so on. This situation contrasts with other disciplines and even other areas of Software Engineering [13], and it is often mentioned as one of the causes that explain the low adoption level of modeling practices by the practitioner's mainstream [14].

Given this situation, the aim of this paper is to augment the repository of empirical data that contributes to giving a scientific answer to the following research question: *What is the impact of the development method (be it rooted in a code-centric, an MBD, or an MDD paradigm) on the productivity and satisfaction of junior developers while developing Web 2.0 applications?*

In order to answer this question, Section 2 describes some previous studies that center on the impact of MBD and MDD practices on process productivity and satisfaction with respect to traditional code-centric practices. Section 3 outlines our experiment design, and analyzes its results and threats to validity. Finally, Section 4 presents the conclusions and further lines of research.

2 Background

Although still scarce in number and not always systematically performed [15], in the last years we have witnessed an increase in the number of empirical studies that provide empirical data regarding the impact of MBD and MDD practices on the productivity and satisfaction of software development teams.

Regarding MBD, in [5] initial evidence is provided about the use of models and tools as an effective way to reduce the time of development and to improve software development variables such as productivity and product quality.

Regarding MDD, there is a number of experiments where productivity of developers using different methods - some model-driven, others code-centered - was measured [9,16,17,18,19,20]. The conclusion in all these studies is that, as projects grow larger, the use of MDD development practices significantly increases productivity (results ranging from two up to nine or even twenty times, depending on the study). The only evidence contradicting these findings is an industrial experience presented in [14]. In this paper, the authors reported a set of studies that showed contradictory results, with both software productivity gains and losses, depending on the particular study. They explained the found productivity losses by pointing at the use of immature tools and high start-up costs. Also, these studies showed that modeling was considered to be an activity at least as complex as programming with a traditional third generation language.

Last but not least, although most of the aforementioned studies include some kind of global satisfaction score, there are few studies that center on the developers' subjective perceptions while applying different methods. In [21], the author 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 users' satisfaction improves after including FUFs in the software development process. Our experiment does not center on the impact of a method enrichment, but compares different methods with respect to the developer's satisfaction and productivity.

Next, we present the quasi-experiment that we have performed to test the impact of three methods, each one an example of a code-centric, an MBD and an MDD approach respectively, on the productivity and satisfaction of junior software developers.

3 Description of the Experiment

During the months of January and February 2011, a quasi-experiment was conducted at the University of Alicante. A quasi-experiment differs from a true experiment in that subjects are not randomly chosen. Quasi-experiments, although suffering from a lower internal validity, are widely used and deemed useful in the Empirical SE field, since they allow investigations of cause-effect relations in settings such as ours, in which randomization is too costly [22].

3.1 Goals and Context Definition

Following the GQM template [23], our empirical study is aimed at *analyzing* three methods, one representative of the code-centric paradigm, one representative of the MBD paradigm and one representative of the MDD paradigm, *for the purpose of* comparison *with respect to* their productivity and satisfaction *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 2.0 application.

The design of the experiment was based on the framework for experimentation in SE suggested by [12]. The whole data set is included in the replication package available at `http://www.dlsi.ua.es/ ccachero/labPackages/Productivity.v1.rar`.

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

- **RQ1**: Is the team's productivity significantly different among methods, regardless of the particular module being developed?

- **RQ2**: Is the developer's satisfaction significantly different among methods, regardless of the particular module being developed?

These research questions were devised to be answerable by quantitative means.

Subjects and Application. The initial group of subjects were 30 students of the Web Applications Developer Master at the University of Alicante. These students were divided into six teams of 4 to 6 people. From them, the data corresponding to Team 6 had to be dropped due to some of their components abandoning the Master for work reasons soon after the experiment had started. Therefore, the final set of observations corresponds to the observations of the remaining five groups (26 subjects). Since the abandonment of the group had nothing to do with the application being developed, the treatments that the group were 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 women, of whom 75% had more than 2 years of professional experience as developers of 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. Regarding the subjects' level of knowledge with respect to the different technologies and methods used during the experiment, a questionnaire showed that 81% knew UML, and that another 12% considered that they had a high-level of knowledge of UML. It should also be noted that 76% of the subjects had previously programmed with VS and .NET during their degree courses, although only 12% had applied them in industry. Finally, the subjects acknowledged no previous practical knowledge of MDD, although 56% of them were aware of the existence of the paradigm. By the time the experiment took place, the subjects had received additional training in all three methods. Such training consisted in 30 hours of training in programming in C# using Visual Studio 2010, 20 hours of training in UML modelling with RSM and 10 hours of training in modelling with the OOH4RIA tool.

Each of the five groups developed a social media application for a different domain:

- Trips: the social network for this domain is focused on establishing relationships between people who want to reduce travel costs by sharing their own cars.
- Events: the social network for this domain is centred on organized social events.
- Hospitals: the social network for this domain aims at improving the communication between physicians and patients.
- Academics: the social network for this domain focuses on connecting and sharing teaching contents among teachers and students.
- Facework: the social network for this domain helps workers to share information about different tasks, resources and goals of the company.

All the applications shared the same complexity, which was controlled by defining a set of functional features that all the applications had to support, regardless of the domain. From them, the three functional features 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., depending on the particular application being developed).
- Support for the organization of events (from now on Events) where people can invite their friends or colleagues to attend to a place where the event is realized (the particular event being a celebration, a work meeting, etc. depending on the particular application being developed)
- Support for an organizational community (from now on Organization) where subjects (be them companies, celebrities, etc., depending on the particular application) can publish content, photos, etc. in a unidirectional way to the social network.

Each one of these functional features was designed as a module. In order to further control the complexity of each module, we strictly defined their architecture, which was based on four main layers: the Business Objects layer (BO), the Data Access Objects layer (DAO), the Data Transfer Objects layer (DTO) and the Database layer (DB). In this way, it was possible to standardize to a certain point the code that had to be developed and facilitate its measurement. Although we are conscious that such strict architecture may hamper the external validity of the experiment, this factor was kept constant across the three treatments, in order to preserve the comparability of the results. The subjects were asked to implement each module following a different method. The time assigned for the implementation of each module was two weeks.

In order to develop the different projects, the students had to follow the Agile Unified Process (Agile UP) methodology [24], a streamlined approach to software development that is based on the IBM's Rational Unified Process (RUP) [25]. 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, are presented in Table 1. All the students had previously developed at least one application with Agile UP, and they had an additional 10-hour training period to refresh the main concepts.

Table 1. Degree of automation of Agile UP disciplines by development paradigm

Discipline	Code-Centric(.NET)	Model-Based(RSM)	Model-Driven(OOH4RIA)
Model	Sketch or absent	Blueprint	Fully-fledged (DSL)
Implementation	Manual	Semi-automatic	Automatic
Test	Manual	Manual	Semiautomatic

Implementation Language and Case Tools. The development environment for the experiment was set up as follows:

- Code Development Environment: NET framework and NHibernate (Object-Relational Mapping).
- IDE (Integrated Development Environment) Development Tool: Visual Studio 2010.
- Languages: C# and the Extensible Application Markup Language (XAML).
- MBD Modeling Environment: Rational Software Modeler (RSM)
- MDD Modeling Environment: OOH4RIA IDE
- Other tools: Subversion 1.6 (SVN), Jira (Issue Tracking) and Survey Monkey (for questionnaires).

The code-centric treatment relied solely on the development enviroment provided by the Visual Studio 2010 and the use of external tools that permit to manage the collaborative work (Subversion). On the other hand, the MBD treatment required the students to work with the UML class diagram of the RSM tool. Last but not least, for the MDD treatment the students worked with the OOH4RIA approach [26].

Students were scheduled to work on these three modules during six weeks along the months of January and February 2011. By this time of the year, the students had already gone through most of the topics of the master, and had gathered a substantial amount of experience with the different tools and the development environments. The experiment defined a tight timetable of deliverables, one every two weeks. Each deliverable consisted of a set of source files and a domain model. The source code had to contain four specific file types: the Business Object files (BO), the Data Access Object files (DAO), the Data Transfer Object files (DTO) and the Database files (DB). The teams were continuously monitored by a Master instructor whose role in the experiment was to look after the quality of the data gathered, both in class and off-line through the Jira and the SVN report systems.

3.2 Experiment Planning

Given the low number of development teams available, and in order to facilitate the detection of method impact by controlling the variability among subjects, the experiment was conceived as an intra-subject design. The combination team-module-approach was defined using a Factorial Design [27,28] (see Table 2). This kind of design avoids order effects and can provide some unique and relevant information about how variables interact or combine in the effect they have on the dependent variables. Also, this design eliminates any possible order effect. Teams were randomly assigned to each treatment order.

In order to answer the research questions presented in Section 3.1, we have defined the following independent (experimentally manipulated) variables (IV) or factors:

Table 2. Experiment design: a factorial, intra-subject design. Group marked with(*) did not finish the experiment.

Team/Module	Application	Group	Events	Organization
1	Travel	code-centric	MBD	MDD
2	Events	code-centric	MDD	MBD
3	Hospital	MBD	MDD	code-centric
4	Academics	MDD	MBD	code-centric
5	Facework	MBD	code-centric	MDD
6*	Automobile	MDD	code-centric	MBD

– Meth: Method, a categorical variable with three levels: code-centric, MBD, MDD. It is important to note that, in this experiment, when we refer to method we are in fact talking about a compound variable (method*tool), due to the coupling of these two variables in our experimental settings.
– Mod: Module, a categorical variable with three possible values: Groups, Events, Organization.

The Dependent (measurable) variables (DV) are:

– P(Meth, Mod), a ratio variable that measures the productivity of the team with each method and module
– S(Meth,Mod), an interval variable, based on a 7-point Likert scale, that measures the satisfaction of the developers with each method and module

The DV have been measured through the following collection procedures:

1. To measure Productivity we measured both the development time and the size of the modules developed by each team.
   – Development time: The student had to document the time of each development activity through the JIRA tool.
   – Module size: We measure the size of the code produced by students in source lines of code (SLOC). SLOCs come in handy to express the size of software among programmers with low levels of experience [29]. We automated the obtention of this measure through the Line Counter [30] tool.
2. To measure Satisfaction we defined a satisfaction scale (SS) made up of eleven items, where each one was based on a 7-point Likert rating scale.

These measures have been used to test the following testable hypotheses, which are based on the research questions and the existing empirical evidence presented in Section 2:

– Productivity Hypothesis (PH): Prod(MDD)>Prod(MBD)>Prod(code-centric). Developer teams are significantly more productive with the MDD method, followed by the MBD method, followed by the code-centric method.
– Productivity-Module Interaction Hypothesis (PMIH): $P(Module*Meth) < 0.05$. The effect on P of the particular module to which the method is applied is insignificant compared to the effect of the method.

– Satisfaction Hypothesis (SH): Satisf(MDD)>S(MBD)>S(code-centric). Developers are significantly more satisfied with the MDD method, followed by the MBD method, followed by the code-centric method.
– Satisfaction-Module Interaction Hypothesis (SMIH): S(Module*Meth)<0.05. The effect on S of the particular module to which the method is applied is insignificant compared to the effect of the method.

3.3 Instrumentation

Besides the instructional materials, all the students received three booklets:

– Modules Description Booklet: a requirements document describing the functional and non functional requirements of the three modules included in the experiment. This booklet was divided in three parts, and it was the same regardless of the order in which the treatments were to be applied.
– Jira Time Reporting Booklet: a document explaining the time reporting conventions that were to be used during the experiment
– Subject Instruction sheet: a set of instructions to the students to correctly perform the experiment.

These instruments are included in the replication package available at `http://www.dlsi.ua.es/~ccachero/labPackages/Productivity.v1.rar`.

The experiment had the following structure:

1. Subject instruction sheet.
2. Pre-experiment questionnaire: it included demographic questions as well as questions about subjects' previous experience with Web application development, Web programming and application modeling, etc.
3. Project work: For each treatment, the students spent two weeks working on the corresponding module with the assigned methodology.
4. Post-experiment questionnaire: it included a semantic-differential scale that required developers to judge each method on 11 pairs of adjectives describing the developer's overall satisfaction with such method.

At the end of each module, the students delivered both the Domain Models and the Source Code (BO, DAO, DTO and DB files).

3.4 Data Analysis and Interpretation of Results

The statistical analysis was carried out with PASW (Predictive Analytics SoftWare) Statistics [31].

Prior to the assessment of the hypotheses, we checked the reliability of the Satisfaction scale in the context of our experimental settings. For the satisfaction scale, all the items showed a correlation higher than 0.3, while the global Cronbach alpha was 0.892, giving proof of a high internal consistency among the scale items. This high correlation has led us to calculate the scale mean for each method, and consider this mean as a global rating of satisfaction with each one of the three treatments (code-centric, MBD, MDD).

RQ1: Impact of Method on Team Productivity. The data gathered to accept/reject the PH and PMIH hypotheses (see section 3.2) are graphically presented in Fig. 1.

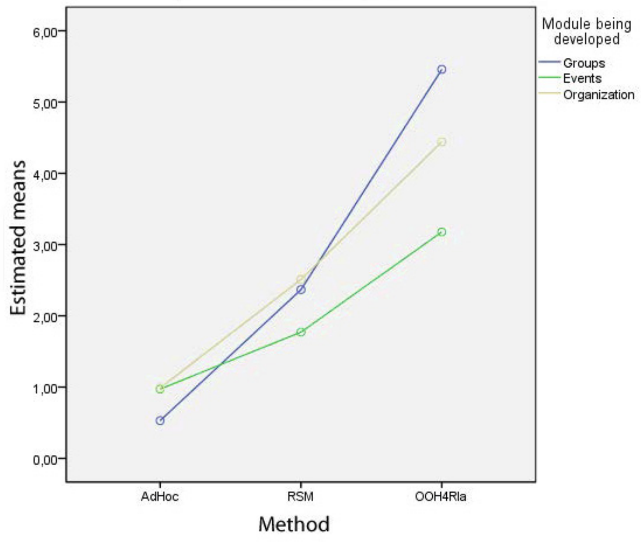

Fig. 1. Productivity: SLOC/Hours

To test the PH and PMIH hypotheses, we applied a 3*5 Mixed Design ANOVA , in which the module (Groups, Events, Organization) was the between-subjects variable, and the calculated P ratings for each method were the within-subjects variables. In order to assure that applying this statistical method made sense, we verified that the principle of spherity was not violated by applying the W Mauchly's test (W=0,005, p>0,05) [32]. The results showed that MDD produced the highest P (M = 4,60, SD=1,17), followed by MBD (M=2,30, SD=1,10) and then Code-centric (M=0,80, SD=0,29), and that these differences were significant (F=25,395, p=0,001).

The results also showed that the interaction Mod*Meth was not significant (F= 3,009, p>0.05). We can then safely examine the main effects of the two independent variables (Mod and Meth) on these means without needing to qualify the results by the existence of a significant interaction. The main effect of module did not attain significance (F=0,538, p>0.05), while the main effect of method did reach significance, (F=25,39, p<0,01), that is, the differences in P are significantly affected by the method used, regardless of the particular module being developed. The last step of the analysis consisted on studying the pairwise differences among methods through a one-way RM Anova with pairwise comparisons. 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 differences in productivity between the Code-centric and the MBD method did not attain significance (t=-2,69, p=0,054) but the differences between Code-centric and MDD (t=-6,029, p=0,004) and MBD and MDD (t=-6,031, p=0,004) did.

RQ2: Impact of Method on Developer's Satisfaction. The data gathered to accept/reject the SH and SMIH hypotheses (see section 3.2) are graphically presented in Fig. 2.

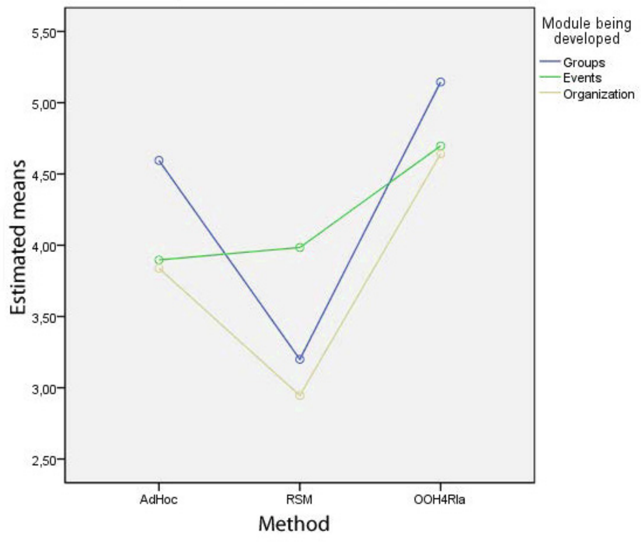

Fig. 2. Satisfaction: Likert Scale

To test the SH and SMIH hypotheses we applied a 3*5 Mixed Design ANOVA , in which the module (Groups, Events, Organization) was the between-subjects variable, and the S ratings for each method were the within-subjects variables. In order to assure that applying this statistical method made sense, we first checked that the principle of spherity was not violated by applying the W Mauchly's test (W=0,838, p=0,142) [32]. The results showed that MDD produced the highest S (M = 4,76, SD=0,73), followed by code-centric (M=4,17, SD=0,72) and then MBD (M=3,48, SD=0,96). The results also showed that the interaction Mod*Meth is not significant (F = 1,768, p>0.05). If we examine the effects of the two independent variables (module and method) we can observe how the Mod inter-subject influence did not attain significance (F=0,167, p>0,05), while the main effect of method did reach significance, (F=18,04, p<0,01), that is, the differences in S are significantly affected by the method used, regardless of the particular module being developed. The last step of the analysis consisted on studying the pairwise differences among methodologies through a one-way

Anova with pairwise comparisons. 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 ($p = 0.05 / 3 = 0{,}0167$). Even with this conservative adjustment, all the pairwise S differences were significant ($p<0{,}05$), which means that, in our experiment, subjects rated significantly differently the three approaches, being MDD the best method rated and MBD the worst.

3.5 Threats to Validity

The analysis of the threats to validity evaluates under which conditions our experiment is applicable and offers benefits, and under which circumstances it might fail. For the classification of these threats, we have followed the classification proposed by Cook and Cambell [33]: internal, external, construction and conclusion.

Threats to conclusion validity refer to the relationship between the treatment and the outcome. In order to minimize the threats, we have strived to automatically capture as many measures as possible, with the help of well-known tracking systems such as JIRA or SVN. Additionally, statistical tests have been chosen conservatively, without making any kind of assumption on variable distributions. However, the fact that the students self-reported the measures, together with the duration of the experiment (six weeks) and the low number of subjects hamper the conclusion validity.

Threats to internal validity 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. All groups applied all the treatments to different modules at different times, what minimizes many internal threats such as selection, history, maturation or social threads such as compensatory rivalry or resentful demoralization. However, being an intra-subject design, carry-over effects may have occurred.

Threats to construct validity refer to the relationship between theory and observation. In this sense, both the treatments and the measures used to assess productivity and satisfaction have been previously widely used in literature. This notwithstanding, there remains the possibility of an interaction of testing and treatments: the need to self-report certain measures may have changed the behavior of the students. We believe that the fact that the experiment took over six weeks minimizes this risk, since it is very difficult to maintain a 'potentially abnormal' behavior over such a long period of time without it being detected. Also, the hypothesis of the experiment (that is, a higher productivity of MDD environments) was quite easy to guess, so students may have felt bound to report less time when using MBD or MDD. Anyway, the experiment observers took special care not to disclose this hypothesis to the students. Additionally, the experiment suffers from a restricted generalizability across constructs: we have checked a positive outcome between productivity and MDD, but we cannot assure that this does not come at the expense of other characteristic of the developed software, such as modularity, reusability, or any other quality attribute.

Last but not least, external validity is concerned with generalization of the results. In this group of threats we have identified a lack of sample representativeness (M.Sc. students), academic environment, a strict architecture and a restricted domain and complexity. Also, we are conscious of the existing coupling between method and tool: all the methods were accompanied by tools. Although we tried to choose well-known development environment and -when possible- use standards (e.g. UML for the modeling activity in MBD and MDD), we are conscious that the different tools add a different 'flavor' to the methods. Therefore, this experiment needs to be replicated in order to make sure that it is the method used and not the tool what causes the observed differences.

4 Conclusions

During the last years the Web Engineering community has claimed how the use of modeling practices in MDD and MDB approaches significantly improves the productivity and satisfaction of Web applications with respect to code-centric development approaches. However, up to now, the quality and quantity of the empirical analyses that demonstrate the true impact of these modeling techniques over the final developer's productivity or satisfaction are still very low. In this paper, we have presented a rigorous analysis of a quasi-experiment carried out in a controlled environment. The data gathered shows that the productivity and the satisfaction of junior Web developers are significantly affected by the development method but they are independent from the particular module being developed. The main conclusions of our study (that still need to be corroborated with further replications) are:

– The MDD approach seems to significantly increase the productivity of developers with respect to both the MDB and the code-centric approach.
– The MDD approach satisfies the most the expectations of juniors developers, followed by code-centric and, in third position, the MDB approach.

These results are well aligned with with the assumption that model-driven techniques improve the productivity and also the satisfaction among developers when they are accompanied by a generation environment. However, the productivity and the satisfaction can decrease even below code-centric practices when the modeling activities are used exclusively as blueprints to improve the understanding, and the developers must implement manually almost all the final code.

Our study of the impact of MDD on the productivity and satisfaction is just the beginning of a family of experiments in which we want to replicate the same analysis with practitioners in industry and also with more complex Web application client-side models. Moreover, further experimentation is needed to separate the effect of methods from the effect of their accompanying development environments (Visual Studio 2010, RSM or the OOH4RIA tool) and to be able to generalize the results to a different population, different methods and languages, different application types or different application sizes.

Acknowledgements. This paper has been co-supported by the DLSI, the Spanish Ministry of Education, and the University of Alicante under contracts TIN2010-15789 (SONRIA) and GRE10-23 (DISEMRIA). The authors also wish to thank their students to take the time to participate in this empirical study. Besides we would like to thank to Jose Javier Martinez and Juan Antonio Osuna, who contributed to the development of the OOH4RIA Tool.

References

1. CMU/SEI: CMMI Product Development Team, CMMI for Development verion 1.2 (2006)
2. Moore, G.C., Benbasat, I.: Development of an instrument to measure the perceptions of adopting an information technology innovation. Information Systems Research 2(3), 192–222 (1991)
3. Fowler, M.: UML distilled: a brief guide to the standard object modeling language, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2004)
4. Kleppe, A.G., Warmer, J., Bast, W.: MDA explained: the model driven architecture: practice and promise. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
5. Bruckhaus, T., Madhavii, N.H., Janssen, I., Henshaw, J.: The impact of tools on software productivity. IEEE Software 13(5), 29–38 (2002)
6. Genero, M., Manso, M.E., Visaggio, A., Canfora, G., Piattini, M.: Building measure-based prediction models for UML class diagram maintainability. Empirical Software Engineering 12(5), 517–549 (2007)
7. Abrahão, S., Iborra, E., Vanderdonckt, J.: Usability evaluation of user interfaces generated with a model-driven architecture tool. Maturing Usability, 3–32 (2008)
8. Mellor, S.J., Clark, T., Futagami, T.: Model-driven development: guest editors' introduction. IEEE Software 20(5), 14–18 (2003)
9. Heijstek, W., Chaudron, M.R.V.: Empirical investigations of model size, complexity and effort in a large scale, distributed model driven development process. In: 35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, pp. 113–120. IEEE (2009)
10. Mohagheghi, P.: An Approach for Empirical Evaluation of Model-Driven Engineering in Multiple Dimensions. In: C2M:EEMDD 2010 Workshop- from Code Centric to Model Centric: Evaluating the Effectiveness of MDD, pp. 6–17. CEA LIST Publication (2010)
11. Kitchenham, B., Budgen, D., Brereton, P., Turner, M., Charters, S., Linkman, S.: Large-scale software engineering questions-expert opinion or empirical evidence? IET Software 1(5), 161–171 (2007)
12. Wohlin, C., Runeson, P., Höst, M.: Experimentation in software engineering: an introduction. Springer, Netherlands (2000)
13. Zelkowitz, M.V.: An update to experimental models for validating computer technology. Journal of Systems and Software 82(3), 373–376 (2009)
14. Mohagheghi, P., Dehlen, V.: Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 432–443. Springer, Heidelberg (2008)
15. Abrahão, S., Poels, G.: A family of experiments to evaluate a functional size measurement procedure for Web applications. Journal of Systems and Software 82(2), 253–269 (2009)

16. Afonso, M., Vogel, R., Teixeira, J.: From code centric to model centric software engineering: practical case study of MDD infusion in a systems integration company (2006)
17. Krogmann, K., Becker, S.: A Case Study on Model-Driven and Conventional Software Development: The Palladio Editor. Software Engineering, 169–176 (2007)
18. Staron, M.: Transitioning from code-centric to model-driven industrial projects–empirical studies in industry and academia. Model Driven Software Development: Integrating Quality Assurance (2008)
19. Kapteijns, T., Jansen, S., Brinkkemper, S., Houët, H., Barendse, R.: A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare. From code centric to model centric software engineering: Practices, Implications and ROI, 22 (2009)
20. Mellegård, N., Staron, M.: Distribution of Effort among Software Development Artefacts: An Initial Case Study. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Ukor, R. (eds.) BPMDS 2010 and EMMSAD 2010. LNBIP, vol. 50, pp. 234–246. Springer, Heidelberg (2010)
21. Panach, J.: Incorporación de mecanismos de usabilidad en un entorno de producción de software dirigido por modelos. Tesis doctotal, Universidad Politécnica de Valencia (2010)
22. Kampenes, V., Dyba, T., Hannay, J., Ksjoberg, D.: A systematic review of quasi-experiments in software engineering. Information and Software Technology 51(1), 71–82 (2009)
23. Perry, D.E., Porter, A.A., Votta, L.G.: Empirical studies of software engineering: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering, pp. 345–355. ACM (2000)
24. Ambler, S.: Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. Wiley (2002)
25. Kruchten, P.: The rational unified process: an introduction. Addison-Wesley Professional (2004)
26. Meliá, S., Gómez, J., Pérez, S., Díaz, O.: Architectural and technological variability in rich internet applications. IEEE Internet Computing 14(3), 24–32 (2010)
27. Montgomery, D.C.: Design and analysis of experiments. John Wiley & Sons Inc. (2008)
28. Plonsky, M.: Psychological Statistics (2009)
29. Gollapudi, K.: Function points or lines of code?–an insight. Global Microsoft Business Unit, Wipro Technologies (2004)
30. Seato: Counting Lines of Code in C# (2004)
31. SPSS Inc. an IBM CompanyHeadquarters: PASW Statistics 18 - Content Guide (2009)
32. Mauchly, J.W.: Significance test for sphericity of a normal n-variate distribution. The Annals of Mathematical Statistics 11(2), 204–209 (1940)
33. Cook, T.D., Campbell, D.T., Day, A.: Quasi-experimentation: Design & analysis issues for field settings. Houghton Mifflin, Boston (1979)