A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation

Paulo Eduardo Papotti[1], Antonio Francisco do Prado[1], Wanderley Lopes de Souza[1], Carlos Eduardo Cirilo[1], and Luís Ferreira Pires[2]

[1] Federal University of São Carlos, Computer Science Department, Rodovia Washington Luis, km 235, PO Box 676, Zip Code 13565-905, São Carlos, SP, Brazil
{paulo.papotti,prado,desouza,carlos_cirilo}@dc.ufscar.br

[2] University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science P.O. Box 217, 7500 AE Enschede, The Netherlands
l.ferreirapires@utwente.nl

Abstract. Recent research results have shown that Model-Driven Development (MDD) is a beneficial approach to develop software systems. The reduction of development time enabled by code generation mechanisms is often acknowledged as an important benefit to be further explored. This paper reports on an experiment in which an MDD-based approach using code generation from models is compared with manual coding based on the classic life-cycle. In this experiment, groups of senior students from Computer Science and Computer Engineering undergraduate academic programs implemented a web application using both approaches, and we evaluated in quantitative terms the performance of the groups. The results showed that the development time when code generation was applied was consistently shorter than otherwise. The participants also indicated that they found less difficulties when applying code generation.

Keywords: Experimentation, Code Generation, Model-Driven Development.

1 Introduction

Model-Driven Development (MDD) [1] has been quite popular in the academic community in the last years, and different approaches based on MDD have been proposed. Most recent studies related to MDD focus on the transformation of domain models to different kinds of software applications. These studies show that software development supported by code generation from abstract models provides several benefits, such as, e.g., increase of productivity, facilitated maintenance and documentation, and portability [2–5].

Although the impact of MDD-based approaches has already been established to some extent, there are still some questions that should be answered, like: What is the average time development reduction that can be obtained with code generation from models compared to traditional development? How can code generation be applied to reduce the difficulties faced by developers? What are the practical advantages of using code generation from models perceived by developers? This paper aims at answering these questions by means of experimentation.

In this paper we report on the results of a quantitative study we performed by applying a systematic experimental methodology [6]. This study evaluates the impact of using model-driven code generation on the time spent by groups of students to develop software systems. In this experiment, at a certain time, the groups developed a web application following the disciplines of the classic life-cycle [7] without using any form of code generation, and at another time, the groups developed the same application by using code generation from models. After that, we analysed the data on the time spent by the participating groups that we collected during the experiment. The paper also reports on the results of this analysis.

This paper is further organised as follows: Section 2 presents some concepts related to the principles of experimentation applied in this work, Section 3 discusses our experiment in detail, Section 4 discusses some related work, and Section 5 gives our final conclusions, including some topics for possible future work.

2 Principles of Experimentation

Empirical validation should help establish Software Engineering as a science [8]. Experiments allow the validation in practice of theories developed in a scientific process. Through experimentation, this validation can be performed in a systematic, disciplined, controlled and computable way [9].

In Software Engineering, experimentation enables one to test hypotheses about a particular object of study (e.g., a process, method, tool, feature, model or theory) and observe the effects of adopting these objects in practical situations. From measurements, collected data are analysed and hypotheses can be validated or refuted [6, 9]. Figure 1 shows the general structure of an experiment.

The null hypothesis (H0) is the main hypothesis of the experiment, and indicates that there is no statistically significant relationship between the cause and the effect being investigated. Generally, the goal of an experiment is to refute this hypothesis in favour of one or more alternative hypotheses (H1, H2, H3, ..., Hn).

In an experiment we can identify independent and dependent variables. Independent variables, which are also called *factors*, refer to the elements that are manipulated or maintained during the experiment, so that the causes that affect the outcome of the experiment can be identified. In contrast, dependent variables refer to the elements that are tested when the experiment is performed. The value assigned to a given factor is called a *treatment*. Therefore, treatments should be applied on the objects of study and on the set of participants in order to identify possible results.

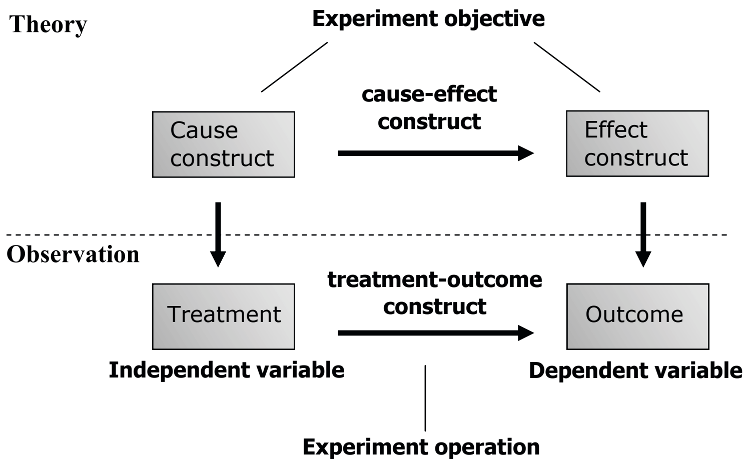

Fig. 1. General structure of an experiment [6]

3 Performed Experiment

Our experiment followed the phases described in Wohlin et al. [6], and was conducted in the second half of 2011. The experiment aimed at performing a comparative analysis of the time spent implementing the CRUD (Create, Retrieve, Update, Delete) functions of web systems from models of the application entity classes described in UML class diagrams. Software development was performed both manually (based on the classic life-cycle) and by using code generation from models.

In the experiment, we used a part of the ProgradWeb[1] system of the Federal University of São Carlos (UFSCar). ProgradWeb is an medium/large size academic system in operation since 1998, which stores all information about students, teachers, registration, certificates, diplomas and other issues related to undergraduate courses at UFSCar.

3.1 Definition

The experiment was defined as follows:

Analyse the use of mechanisms for code generation from models in the construction of web applications;
For the purpose of evaluation;
With respect to efficiency in terms of time;
From the point of view of software developers;
In the context of Computer Science and Computer Engineering undergraduate students.

[1] ProgradWeb - `https://progradweb.ufscar.br/progradweb/`

The experiment was conducted in a university environment with the collaboration of undergraduate students of the Computer Science and Computer Engineering programs at UFSCar. The context of our study corresponds to *multi-test within object study* [6]. Thus, the experiment consisted of various experimental tests, in which different groups of subjects implemented a single application in different ways.

3.2 Planning

After the definition, the experiment was planned according to the steps discussed below.

Context Selection. The experiment was conducted in a university environment, in a teaching laboratory of the Computer Department of UFSCar, within the "Topics in Computer Science 2011" course. The experiment involved the participation of students of the third and fourth academic years in Computer Science and Computer Engineering at UFSCar.

Hypothesis Formulation. In order to determine the effect of the development approach (with and without code generation) in the results, three hypotheses were formulated for the experiment. The following metrics were considered when formulating hypotheses:

τ Total time spent by the groups to develop the Web application with CRUD functionalities.

$\mu\tau$ Average time spent by the groups to develop the Web application with CRUD functionalities

The hypotheses formulated for the experiment are:

Null Hypothesis (H0). In general, there is no difference in efficiency (ε) between the groups when they apply code generation from models, and when they apply manual coding based on the classic life-cycle to build web applications (Equation 1).

$$H0 : \varepsilon_{CodeGeneration} = \varepsilon_{Classic} \Rightarrow \mu\tau_{CodeGeneration} = \mu\tau_{Classic} \quad (1)$$

Alternative Hypothesis (H1). When groups apply code generation from models to build web applications they are generally more efficient than when they apply manual coding based on the classic life-cycle (Equation 2).

$$H1 : \varepsilon_{CodeGeneration} > \varepsilon_{Classic} \Rightarrow \mu\tau_{CodeGeneration} < \mu\tau_{Classic} \quad (2)$$

Alternative Hypothesis (H2). When groups apply manual coding based on the classic life-cycle to build web applications they are generally more efficient than when they apply code generation from models (Equation 3).

$$H2 : \varepsilon_{CodeGeneration} < \varepsilon_{Classic} \Rightarrow \mu\tau_{CodeGeneration} > \mu\tau_{Classic} \quad (3)$$

Variables Selection. In this step we chose the variables that allowed us to analyse the efficiency of the groups when using different development approaches. The variables were chosen as follows:

Independent Variables. We selected the following independent variables in this experiment: developed *application*, the *development approach* used to implement the application, the *development environment*, and the *technologies* used for development. Since the purpose of the experiment is to investigate the impact of the development approach on group efficiency, variable "*development approach*" was chosen as the factor that receives different treatments. The other independent variables remained constant during the execution of experiment, and have been set as follows:

Application. CRUD of part of ProgradWeb system.

Development Environment. Eclipse IDE and MySQL database.

Development Technologies. Java, Hibernate, Java Persistence API (JPA), Java Server Faces (JSF) and XHTML.

Dependent Variable. In accordance with the hypotheses, we chose *group efficiency* as dependent variable in this experiment.

Selection of Subjects. Subjects were selected according to *convenience sampling* [6]. In this non-probabilistic technique, the selected participants were the closest and most convenient to conduct the experiment. Voluntarily, 29 students from the third and fourth years of Computer Science and Computer Engineering undergraduate academic programs of UFSCar participated in the experiment, in the scope of their attendance to the "Topics in Computer Science 2011" course.

Experiment Design. The experiment followed the general design principle of grouping the participants in homogeneous blocks [6], avoiding a direct impact of the experience level of the participants in the treatment outcomes of the *development approach* factor, increasing in this way the accuracy of the experiment.

The participants were divided into 9 homogeneous groups (blocks), where 7 groups contained 3 participants and 2 groups contained 4 participants. The groups were divided so that the average experience level of the groups was as similar as possible.

To determine the experience level of each participant, we used a *Participant Characterization Form.* The participants answered multiple-choice (closed) questions to assess their knowledge regarding the topics covered by the experiment (e.g., Java, Hibernate, JSF, UML), allowing the experience level of each participant according to the technologies used in experiment to be quantified. Figure 2 displays the individual experience levels of each participant P_i and the average experience level of each group G_j, according to the information given by the students in their characterization forms. These levels were obtained by calculating the average of knowledge degree in a 5-point scale for each question of the form answered by the participants.

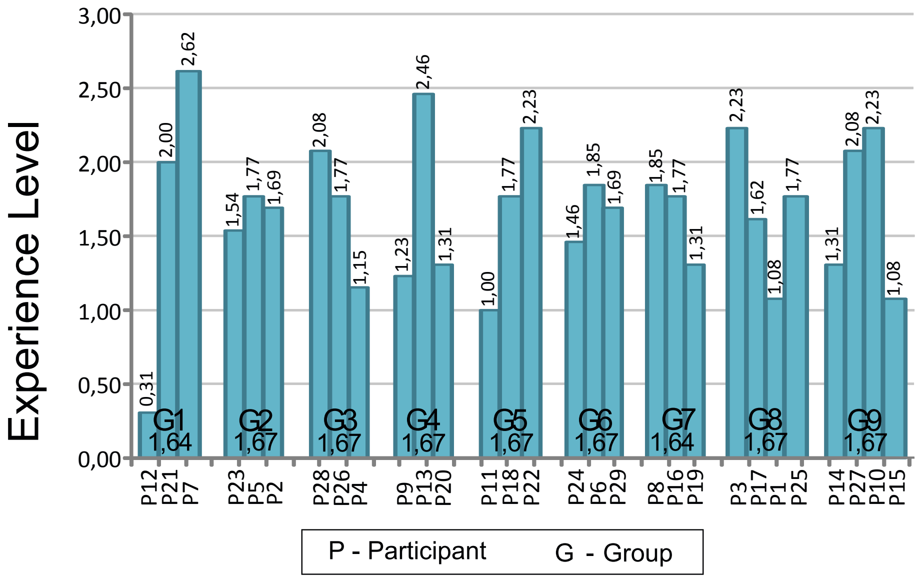

Fig. 2. Distribution of participants in the experiment

With respect to the design type, the experiment consisted of a paired comparison [6] of one factor (development approach) with two treatments (code generation and manual coding). In this design type, each subject (or a group of subjects) uses both treatments on the same object of study. The order in which the participants apply the treatments is set at random. Therefore, in our experiment, the application was developed by all participating groups using both code generation and manual coding. Table 1 shows the randomly selected order in which the treatments have been assigned to the groups, where '1' and '2' indicate which treatment was applied first and second, respectively.

Instrumentation. All the material required to support the participants throughout the experiment was made available beforehand, including the preparation of

Table 1. Assignment of treatments to groups

Group	Code Generation	Manual Coding
G1	1	2
G2	1	2
G3	2	1
G4	1	2
G5	2	1
G6	2	1
G7	2	1
G8	1	2
G9	1	2

objects, instruments and guidelines for data collection used during the execution of the experiment.

3.3 Experiment Execution

Once the experiment was defined and planned, we executed the experiment according to the following steps: preparation, operation and validation of the collected data.

Preparation. At this stage, the students got committed and involved with the experiment and were made aware of its purpose and research objectives. During the preparation of the experiment, all participants were informed and accepted the terms regarding the confidentiality of the provided data, which would be only used for academic purposes, and their freedom to withdraw, by signing a *Consent Form.*

At this stage, the following objects were produced to be used in the experiment:

Participant's Characterization Form. Questionnaire in which the participants assess their knowledge on the technologies and concepts used in the experiment.

Consent Form. Document signed by the participants, stating the objectives and confidentiality of the experiment, and their freedom to withdraw.

Task Description. Document describing the task to be performed in the experiment.

Support Material. Roadmap describing the steps to be used in the development of applications.

Data Collection Form. Document containing empty spaces to be filled in by the participants to record the start time and end time of each activity performed during the experiment.

Evaluation Form. Questionnaire containing multiple-choice questions for the participants to evaluate and compare the two development approaches used in the experiment.

In order to avoid the interference of the time spent in learning the approaches to code generation and manual coding, a training with a duration of two weeks was planned and provided to all participants, so that everybody was able to develop the kind of application proposed in the experiment.

The platform adopted for developing both applications consisted of Java as implementation language, the JSF framework to support the MVC pattern implementation, JPA and Hibernate to support persistence in a MySQL[2] database and the Eclipse IDE[3] as development environment. This platform was installed and configured beforehand in the computers where the experiment was conducted.

[2] MySQL - `http://www.mysql.com/`

[3] Eclipse IDE - `http://www.eclipse.org/`

Operation. On the day in which the experiment was performed, the tools and materials used in experiment were stored in a package that was made available to the groups. This package contained the following elements:

– Eclipse IDE configured with the Papyrus UML[4] plug-in.
– Library files (Hibernate, JSF and MySQL Connector).
– Folder with the Apache Tomcat Server to run the developed application.
– Class diagram of the developed application.
– All the documents described in experiment preparation step.

In addition, when following the code generation approach, the groups received a code generator previously developed for the experiment. This generator consisted of the M2C transformations to generate Java code from UML class diagrams.

The groups performed the experimental task as defined in the *Task description*. All groups implemented the proposed application with all required functions. The applications resulting from using both approaches were equivalent, and had the same quality and performance levels.

During the experiment, the groups recorded on the *Data Collection Form* the start time and end time of each activity performed to implement the application. Table 2 summarises the collected data, showing the time spent on the implementation activities. Table 3 gives a description of the acronyms used in Table 2.

Table 2 shows that the data divided into two blocks, for the manual coding and code generation approaches. Table 2 also shows average and total values.

Data Validation. In general, we observed that the groups developed the proposed tasks satisfactorily, and the collected data was within the expected limits. This means that the treatments were executed correctly and in accordance with the planning. Therefore, we can claim that the obtained data was valid to conduct the proposed evaluation.

3.4 Results Analysis and Interpretation

Table 2 shows that when using the manual coding approach, the groups spent an average of 2 hours and 2 minutes to complete the proposed activities, whereas when following the code generation approach this time dropped to an average of 11 minutes (reduction of 90.98%). Therefore, it is fair to conclude that the groups completed the implementation task more quickly when following the code generation approach than when following the manual coding approach. This difference is justified by the model-to-code (M2C) transformations, which automatically generated a substantial amount of source code from a predefined model.

In the case of applications with more classes and attributes in each class in their models than the application used in our experiment, the amount of time and effort required to perform the implementation activities following the manual coding approach, tends to be even longer. Although not all the necessary

[4] Papyrus UML - `http://www.eclipse.org/papyrus/`

Table 2. Time spent on the experiment activities

		TProj	TEntity Classes	TPersistence	TManaged Beans	TConverters	TWeb Pages	Total
Manual Coding	G1	0 h 14 min	0 h 32 min	0 h 7 min	0 h 25 min	0 h 6 min	0 h 44 min	2 h 8 min
	G2	0 h 6 min	0 h 38 min	0 h 3 min	0 h 13 min	0 h 4 min	0 h 59 min	2 h 3 min
	G3	0 h 7 min	0 h 59 min	0 h 3 min	0 h 8 min	0 h 3 min	0 h 37 min	1 h 57 min
	G4	0 h 8 min	1 h 8 min	0 h 8 min	0 h 14 min	0 h 6 min	0 h 15 min	1 h 59 min
	G5	0 h 7 min	0 h 49 min	0 h 8 min	0 h 15 min	0 h 6 min	0 h 40 min	2 h 5 min
	G6	0 h 8 min	0 h 41 min	0 h 13 min	0 h 25 min	0 h 7 min	0 h 33 min	2 h 7 min
	G7	0 h 13 min	0 h 33 min	0 h 7 min	0 h 32 min	0 h 8 min	0 h 22 min	1 h 55 min
	G8	0 h 5 min	0 h 30 min	0 h 10 min	0 h 10 min	0 h 2 min	1 h 2 min	1 h 59 min
	G9	0 h 8 min	1 h 7 min	0 h 9 min	0 h 8 min	0 h 6 min	0 h 29 min	2 h 7 min
	Average	**0 h 8 min**	**0 h 46 min**	**0 h 7 min**	**0 h 16 min**	**0 h 5 min**	**0 h 37 min**	**2 h 2 min**
Code Generation	G1	0 h 7 min	0 h 4 min	0 h 4 min				0 h 15 min
	G2	0 h 8 min	0 h 3 min	0 h 1 min				0 h 12 min
	G3	0 h 1 min	0 h 7 min	0 h 1 min				0 h 9 min
	G4	0 h 3 min	0 h 4 min	0 h 1 min				0 h 8 min
	G5	0 h 2 min	0 h 1 min	0 h 1 min				0 h 4 min
	G6	0 h 8 min	0 h 2 min	0 h 3 min				0 h 13 min
	G7	0 h 10 min	0 h 2 min	0 h 1 min				0 h 13 min
	G8	0 h 6 min	0 h 2 min	0 h 1 min				0 h 9 min
	G9	0 h 9 min	0 h 6 min	0 h 3 min				0 h 18 min
	Average	**0 h 6 min**	**0 h 3 min**	**0 h 1 min**				**0 h 11 min**

Table 3. Description of the acronyms used in Table 2

Acronym	Description
TProj	Time spent in creating the Web project on Eclipse IDE
TEntityClasses	Time spent in implementing the entity classes using JPA
TPersistence	Time spent in implementing session beans to access the database
TManagedBeans	Time spent in the implementation of managed beans
TConverters	Time spent in the implementation of converters
TWebPages	Time spent in the implementation of Web pages using JSF

application code is automatically generated when using the code generation approach, this approach can already provide a significant productivity gain, which justifies the effort and time spent on building generators based on model-to-code transformations, as the one used in this experiment. This also means that any application that uses the same platform and technologies as the ones for which these generators are built can be developed with much less effort than if they would be implemented using manual coding.

Descriptive Statistics. Descriptive statistics deals with the presentation and processing of a numerical data set [6]. Once an experiment is performed,

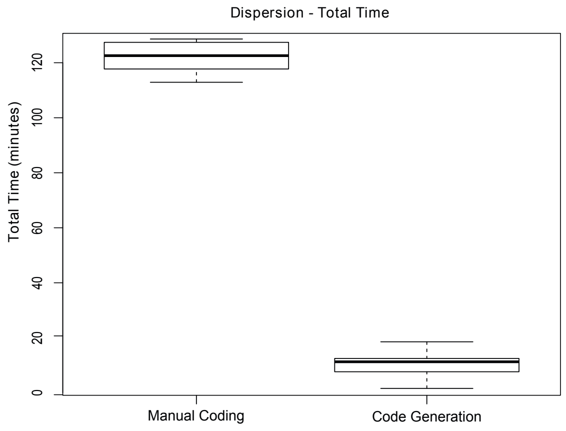

Fig. 3. Dispersion of the total time spent by the groups

descriptive analysis allows the collected data to be analysed, grouped and graphically presented, to view the results from different perspectives.

Figure 3 shows the dispersion of time spent by the groups that participated in the proposed experiment in a boxplot chart. Figure 3 shows clearly that when the groups used code generation they were faster than when they applied manual coding.

Furthermore, the normality of the data set was verified using the Shapiro-Wilk nonparametric test [10]. The test results confirmed that the sample sets obtained with both approaches consisted of normally distributed sets, as shown in Figure 4.

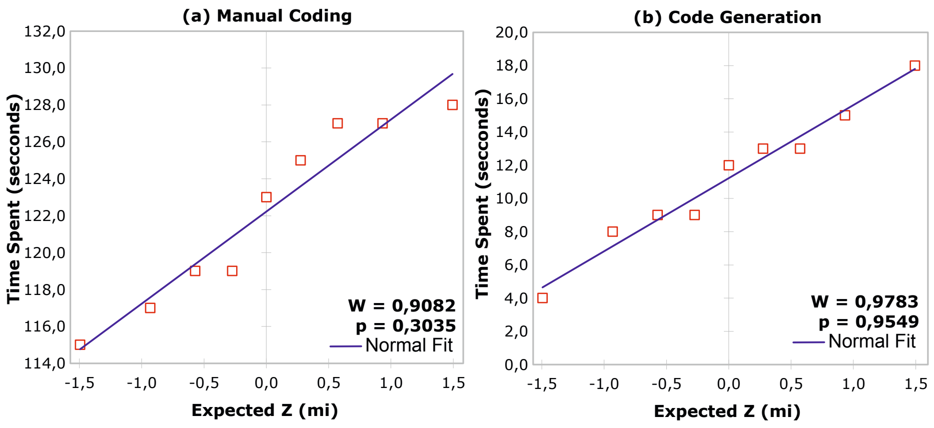

Fig. 4. Normality of the sample data sets

Data Set Reduction. Before applying statistical methods, it is necessary to check the quality of the input data [6]. The representation and correctness of data have a direct impact on the conclusions that can be drawn from the results, since if incorrect data is used in the statistical analysis, wrong conclusions will probably be drawn. Incorrect data sets can be obtained due to systematic errors or the presence of *outliers*, which are data values that are much higher or much lower than expected when compared with the remaining data.

After a careful analysis of the forms filled by the participants, we have not found any systematic errors (e.g., transcription errors by the participants) nor outliers in the input data set. Therefore, there was no need to reduce the data set, and the data were considered valid for applying statistical methods of data analysis.

Hypothesis Testing. When analysing the data produced by the groups that participated in the experiment, we noticed that when groups followed the code generation approach they were more efficient than when they followed the manual coding approach. In order to verify and quantify the actual efficiency gain from the collected data, we applied the paired t-test [11] (Equation 4).

$$t_0 = \frac{\bar{d}}{S_d/(\sqrt{n})} \tag{4}$$

where:

n Number of paired samples.
$\bar{d}$ Difference of averages of each samples set.
S_d Standard deviation of the differences of the samples (Equation 5).

$$S_d = \sqrt{\frac{\sum_{i=1}^{n}(d_i - \bar{d})^2}{n-1}} \tag{5}$$

where:

d_i Difference between each pair of sample.

Assuming a two sided paired t-test, we can reject H_0 if $|t_0| > t_{\alpha/2,n-1}$. In this case, $t_{\alpha,f}$ is the upper α percentage point of the t-distribution with f degrees of freedom.

Based on the samples, $n = 9$ and $d = \{113, 111, 108, 111, 121, 114, 102, 110, 109\}$ (in minutes). The average values of each data set are $\mu_{Classic} = 122,222$ and $\mu_{Generation} = 11,222$. So, $\bar{d} = 122.222 - 11.222 = 111$, which implies that $S_d = 5.099$ and $t_0 = 65.30667$.

The number of degrees of freedom is $f = n - 1 = 8$. We take $\alpha = 6.8 \cdot 10^{-12}$. In the table of statistical probability of the *Student's t-distribution*, we could verify that $t_{3.4 \cdot 10^{-12},8} = 65.2154$. Since $|t_0| > t_{3.4 \cdot 10^{-12},8}$ **the null hypothesis $\mathbf{H_0}$ can be rejected with a significance level of** $6.8 \cdot 10^{-12}\%$.

To perform the calculations mentioned above we used the software R[5], with RKWard[6] as a graphical user interface.

3.5 Discussion of Results

After we performed hypothesis testing and we verified the possibility of rejecting the null hypothesis (H_0), we could draw some conclusions regarding the influence of independent variables on the dependent variable, the validity of the experiment and the treatment of the validity threats (in Section 3.7).

Concerning the rejection of the null hypothesis, we can conclude that the differences in development times have statistical significance, as evidenced by the samples provided by the groups that followed the manual coding and code generation approaches. This indicates that the difference in development time spent by the groups was due to the development approach they used during the experiment, and not by any accident or mistakes in the collection of samples.

Table 2 shows that when the groups used code generation, the average total time was consistently lower than when they used manual coding ($\mu\tau_{Generation} < \mu\tau_{Classic}$). This is an evidence that hypothesis H_1 can be validated instead of hypothesis H_2. Therefore, we can safely argue that groups that use code generation from models usually spend less time in software development. This result is within our expectations for this experiment, since we expected that automated code generators would speed up parts of the development task. This expectation was confirmed in practice by our experiment.

Furthermore, the conclusions on the results of this study are limited to the scope of CRUD web applications implemented by software developers in a university environment, since the experiment was performed *in vitro* and under controlled conditions. In our data set, only the time spent on implementation activities was collected, which means that the time spent on other steps of software development (e.g., modeling and testing) was not covered by the experiment reported in this paper. To extend our results to a broader context, we would have to perform new experiments with an increased number of subjects, and performed in *in vivo* environments, in which we could compare the use of code generation from models with other approaches for software development beyond manual coding according to the classic life-cycle, as addressed in this study.

Replication of our experiment in an industrial environment could increment the validation of our work, as it would extend the results to new contexts, where model-driven approaches could be compared with other development approaches (e.g., software product lines and agile methods), also considering factors that do not play a role in an academic environment. In addition, other effects related to software development may be studied, such as, e.g., effectiveness with respect to faults during development. In this case, empirical evaluations with users and inspection tests should be planned in order to collect relevant metrics for assessing

[5] R - `http://www.r-project.org/`
[6] RKWard - `http://rkward.sourceforge.net/`

the degree to which the developed applications reach the goals of effectiveness, efficiency and subjective satisfaction from the point of view of their end users.

A package containing the tools, materials and more details about the experiment steps is available at `www.dc.ufscar.br/∼paulo.papotti/EXPERIMENT.zip` and can be used by researchers and practitioners to help them perform new experiments related to our study.

3.6 Participants' Opinion

We analysed the participant's opinion in order to evaluate the impact of using the approaches considered in the experiment. After the experiment operation, all students received two evaluation forms with multiple-choice questions with empty spaces for them to report on their perception of the manual coding and code generation approaches.

After the participants filled in both questionnaires, the answers were analysed and some interesting results were obtained. When asked if they encountered difficulties in the development of the proposed tasks when they followed the manual coding approach, 54% of the participants reported having difficulties, 32% mentioned partial difficulties and only 14% had no difficulties. In contrast, when asked the same question with respect to the code generation approach, 71% reported not having any difficulty, 11% mentioned partial difficulties, and only 18% of all participants had total difficulty in completing the tasks. Figure 5 shows charts that visualise the levels of difficulty faced by the participants.

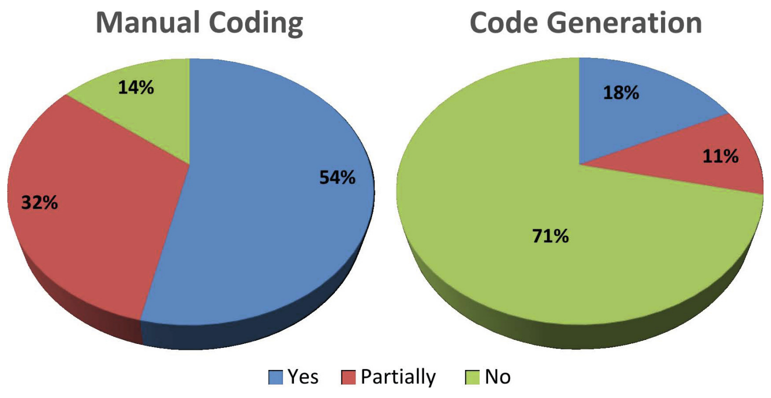

Fig. 5. Perceived difficulty of participants for both approaches

Figure 5 shows a decrease in the perceived difficulty when the participants used code generation. When following the manual coding approach, 86% of the participants had some kind of difficulty (total or partial), while for the code generation approach this value fell to 29%. Therefore, we believe that the mechanisms for code generation were essential to facilitate the task of the participants, since there has been an increase of 57% in the percentage of participants who developed the proposed task without any difficulties.

The most common difficulties pointed out by the participants when they followed the manual coding approach are: (1) too much effort spent on coding; (2) problems related to the language; and (3) mistakes they made due to lack of attention. In contrast, the most common difficulties faced by participants when using code generation are: (1) lack of practice with the use of generators; and (2) poor integration between the generators and the IDE used for development.

Since all participants stated that the code generation approach assisted them in performing the development task, they were also asked to mention the advantages of this approach when compared with manual coding. Figure 6 summarises the results of this enquiry, showing that 92.86% of the participants mentioned the generation of part of the application code as the biggest advantage of the code generation approach. Other advantages that have been mentioned are the increased focus on modeling (71.43%), the ease to perform future maintenance (53.56%) and others (7.14%).

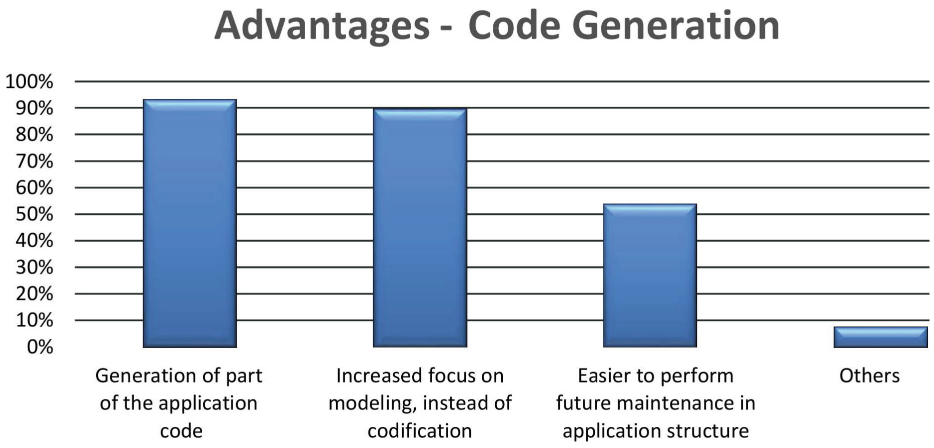

Fig. 6. Advantages of code generation pointed out by participants

3.7 Validity Threats

Whenever an experiment is performed, the validity of its results should be assessed. An experiment may have its results put at risk and invalidated for different reasons depending on the way it was conducted. Therefore, the conditions for validity should be considered since the initial stages of the experiment. The main types of validity that should be considered are: conclusion validity, internal validity, construct validity and external validity. All these validities were considered in our experiment.

Conclusion Validity. Different precautions were taken to ensure the conclusion validity of the experiment. We used a parametric statistical test (paired t-test), which is suitable for evaluating the factor "*development approach*" with treatments "*code generation*" and "*manual coding*".

The normality of the collected data was confirmed before using the paired t-test through the Shapiro-Wilk normality test (Figure 4) with a significance level of at least 5%. However, to reinforce and ensure the validity of results, we applied the Wilcoxon non-parametric test, an alternative test to the paired t-test, which does not require that the collected data are normally distributed. In this case, the results pointed to the same direction as the paired t-test, confirming the rejection of the null hypothesis.

Furthermore, to increase the validity of the conclusion, the data collected by participants (hour and minutes) do not dependent on human judgement and are quite reliable, even though they were collected by the participants themselves.

Finally, the heterogeneity of the experience level of the participants was treated by grouping them in blocks (groups) with similar average experience level.

Internal Validity. The experiment was conducted by undergraduate students of the Computer Science and Computer Engineering academic programs at the final stage (third and fourth academic years) of their study, who are well acquainted with software development. Therefore, we are confident that these students are representative for the population of software developers.

Furthermore, a questionnaire was used to characterize the participants of the experiment by capturing their profile and their average experience level. This information was used to assign them to homogeneous groups of similar average experience level.

Construct Validity. The goal of the proposed experiment was to compare two different approaches to software development and their impact on the time spent on the development task by groups of participants. The data on the time spent by the groups were collected during the development of a sample application using both approaches in order to perform this comparison.

In order to avoid any interference in the behaviour of the participants, the metrics and calculations that were used on the collected data were not disclosed to the participants, so that they would keep their focus on the development task in the most spontaneous way as possible, instead of seeking opportunities to get results that would favour or harm the experiment.

External Validity. Our experiment was conducted in a computerized laboratory, equipped with the items necessary to perform the development task, including the tools and technologies used in software development in industrial environments, such as Java tools and the Eclipse IDE. The experiment was completely performed in a period of about 3 hours, so that the results have not been affected by excessive fatigue or boredom of the participants.

4 Related Work

A study of the impact of MDD adoption in large scale projects is presented in [12], in which different characteristics of a large industrial project were investigated in which MDD was applied in its pure form. The study focused on the analysis of the size and complexity of models produced in the project, considering metrics related to quality and modeling efforts. Similarly, three case studies related to the implementation of the MDE in a industrial environment are presented in [13]. The study highlights the importance of social, technical and organizational requirements for the successful adoption of model-driven software development in practice. The authors used questionnaires, interviews and observation to collect data from industrial professionals. Challenges and barriers related to the adoption of MDD were discussed in [14], based on the authors' experience in different industrial and academic projects. A quantitative analysis with preliminary results from the adoption of a tool intended for software maintenance in the MDD context is reported in [15].

Our work is closely related to the work on practical experiments with model-driven software development mentioned above, and especially the ones related to the use of code generation mechanisms. However, unlike most related work, our study has a strong focus on quantitative statistical analysis of real data collected by the participants of the experiment. Furthermore, the results of our work have a more limited scope, and are specifically related to the use of code generation from models, aiming at a deeper analysis within the proposed scenario.

5 Final Remarks

This paper presented the results of a quantitative analysis of an experiment in which the use of mechanisms for code generation from models in software development has been evaluated. The population of participants in the experiment was composed of undergraduate students of the last years from UFSCar who developed an application based on ProgradWeb system, following both a development approach based on manual coding, as with code generation from models. We collected data about the time spent on the development task in both cases. From the data collected, the analysis revealed an average of 90.98% reduction in development time, considering the conditions of the experiment. In addition, the code generation approach contributed to reduce the difficulties encountered by 57% of the participants.

The results achieved in this paper reaffirm the benefits of approaches that use model-driven code generation, especially for the development of CRUD web applications. Productivity increase and difficulties reduction during development were the main points we explored in our experiment. Our study directly evaluates the benefits of MDD and quantifies the achieved gains. Although the study was conducted in a university environment, the results are valid and may be used in other research projects (respecting our adopted conditions) that aim at making further progress in the related research areas.

Finally, some topics for further work include: perform the experiment described in this paper again in an industrial environment; perform other experiments to compare code generation from models with other approaches that are not model-based; and evaluate the effectiveness of development by analysing the faults found by the groups during the experiment.

Acknowledgments. We thank the National Counsel of Technological and Scientific Development (CNPq) for sponsoring our research in the context of the INCT-MACC.

References

1. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: 2007 Future of Software Engineering, FOSE 2007, pp. 37–54. IEEE Computer Society, Washington, DC (2007)
2. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
3. van Deursen, A., Klint, P.: Little languages: little maintenance. Journal of Software Maintenance 10, 75–92 (1998)
4. Bhanot, V., Paniscotti, D., Roman, A., Trask, B.: Using domain-specific modeling to develop software defined radio components and applications. In: Proceedings of the 5th OOPSLA Workshop on Domain-Specific Modeling, San Diego, CA, USA (2005)
5. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37, 316–344 (2005)
6. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering: an introduction. Kluwer Academic Publishers, Norwell (2000)
7. Pressman, R.: Software Engineering: A Practitioner's Approach, 6th edn. McGraw-Hill, Inc., New York (2005)
8. Conradi, R., Basili, V., Carver, J., Shull, F., Travassos, G.: A pragmatic documents standard for an experience library: Roles, documen, contents and structure (2001)
9. Travassos, G., Gurov, D., Amaral, E.: Introdução à engenharia de software experimental. UFRJ (2002)
10. Shapiro, S., Wilk, M.: An analysis of variance test for normality (complete samples). Biometrika 52(3/4), 591–611 (1965)
11. Montgomery, D.: Design and analysis of experiments. Wiley (2008)
12. Heijstek, W., Chaudron, M.: 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 (August 2009)
13. Hutchinson, J., Rouncefield, M., Whittle, J.: Model-driven engineering practices in industry. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE), pp. 633–642. ACM (2011)
14. Mohagheghi, P., Fernandez, M.A., Martell, J.A., Fritzsche, M., Gilani, W.: MDE adoption in industry: Challenges and success criteria. In: Chaudron, M.R.V. (ed.) MODELS 2008. LNCS, vol. 5421, pp. 54–59. Springer, Heidelberg (2009)
15. Ricca, F., Leotta, M., Reggio, G., Tiso, A., Guerrini, G., Torchiano, M.: Using unimod for maintenance tasks: An experimental assessment in the context of model driven development. In: 2012 ICSE Workshop on Modeling in Software Engineering (MISE), pp. 77–83 (June 2012)