In Search of Evidence for Model-Driven Development Claims: An Experiment on Quality, Effort, Productivity and Satisfaction

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

[1]Escola Tècnica Superior d'Enginyeria, Departament d'Informàtica, Universitat de València
Avenida de la Universidad, s/n, 46100 Burjassot, Valencia, Spain
joigpana@uv.es
[2]Centro de Investigación en Métodos de Producción de Software - ProS
Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain
{sergio.espana, opastor}@pros.upv.es
[3]Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, Campus de Montegancedo, 28660, Boadilla del Monte, Spain
{natalia,odieste}@fi.upm.es

Abstract. Context: Model-Driven Development (MDD) is a paradigm that prescribes building conceptual models that abstractly represent the system and generating code from these models through transformation rules. The literature is rife with claims about the benefits of MDD, but they are hardly supported by evidences. **Objective:** This experimental investigation aims to verify some of the most cited benefits of MDD. **Method:** We run an experiment on a small set of classes using student subjects to compare the quality, effort, productivity and satisfaction of traditional development and MDD. The experiment participants built two web applications from scratch, one where the developers implement the code by hand and another using an industrial MDD tool that automatically generates the code from a conceptual model. **Results:** Outcomes show that there are no significant differences between both methods with regard to effort, productivity and satisfaction, although quality in MDD is more robust to small variations in problem complexity. We discuss possible explanations for these results. **Conclusions:** For small systems and less programming-experienced subjects, MDD does not always yield better results than a traditional method, even regarding effort and productivity. This contradicts some previous statements about MDD advantages. The benefits of developing a system with MDD appear to depend on certain characteristics of the development context.

Keywords: Automatic programming, Methodologies, Programming paradigms, Quality analysis and evaluation,

1 Introduction

Model-Driven Development (MDD) [15] [33] is a paradigm advocating the use of models as the primary software development artefact and model transformations as the main operation. The idea is that all that is needed to develop a system is to build its conceptual model [37]. The conceptual model is the input to a model compiler that automatically generates software code to implement the system, or to a model interpreter that directly executes the model. MDD is the natural continuation of the evolution that gradually raised the abstraction level from assembly languages to third-generation programming languages [40].

Although MDD recommends automating as much code generation as possible, nowadays there is a wide range of approaches to apply the paradigm. Some of the proposals, such as OO-Method [40] WebRatio [6], Genexus [1] and OOHDM [41], generate fully functional systems through automatic transformations. Others generate part of the system. For example, NDT [24] can generate all of the code that supports behaviour and persistency, but most of the user interface needs to be manually implemented.

MDD advocates often claim that it has advantages over traditional software development. For example, Mellor [33] states that the use of models increases productivity, and Selic [42] states that MDD helps to improve productivity and reliability. However, few of these claims have been empirically evaluated. Existent empirical studies focus on measuring time, overlooking other characteristics that MDD is claimed to have, such as quality. There is a lack of empirical evaluations of MDD, probably due to the inherent complexity of comparative evaluations of software development methods and the challenges of adopting MDD in industrial contexts. Staron has reported the following difficulties suffered when applying MDD under conditions of practice [46]: (i) MDD methodological and technological learning curves are high, (ii) there is no development standard, (iii) relations among the multiple views within the conceptual model are unclear, and (iv) the transformations needed to generate code from models are difficult to design.

In this work, we have designed and conducted an experiment to verify some of the claimed advantages of MDD. We aim to contribute to corroborating or refuting some of the claims that have been historically attributed to MDD and widely published in the literature. We have compared an MDD with a traditional method where developers implement the code manually. The experimental tasks are to develop small but fully-functional web applications from scratch. We focus on

evaluating software quality and developer effort, productivity and satisfaction since these are the most popular claims about MDD in the literature. The experimental subjects are last-year master students who have competence in traditional development and no significant previous experience with MDD. As operationalisation of MDD, we have used an industrial tool that can generate fully-functional systems from conceptual models: INTEGRANOVA [2]. MDD is applicable to the development of any system, such as information [40], embedded [47] or cyber-physical systems [27], among others. Our experiment focuses on information systems.

For inexperienced developers, we have observed that there are no significant differences between MDD and a traditional method regarding effort, productivity and satisfaction. However, we have observed that quality in MDD is more stable than a traditional method to variations in problem complexity. These preliminary results clearly contradict the claims that have been accepted as facts (i.e. quality, effort, productivity and satisfaction in MDD are always better) and call for a thorough study and deeper understanding of the conditions under which MDD might be better than other development paradigms. We have analysed some reasons why MDD claims are not satisfied in our experiment. We have identified some variables that appear to influence the suitability of the development paradigm (MDD or traditional) to a project situation, such as problem complexity and developers' background experience with MDD. Results must be interpreted within the context in which the experiment has been run: (i) the subjects are students, (ii) they have previous experience with traditional development and they are learning to develop information systems with MDD, (iii) the systems are developed from scratch and (iv) their size is small. We conclude that further experimental research is required to gain insight and to better understand the conditions under which MDD might be an alternative to traditional software development.

The paper is organised as follows. Section 2 discusses related work. Section 3 describes the experiment definition and planning. Section 4 presents the outcomes of the study. Section 5 shows the threats to validity identified after running the experiment. Section 6 discusses the interpretation of the results. Finally, Section 7 shows the conclusions.

2 Related Work

We have reviewed the literature in search of statements claiming benefits of MDD. We have generalised similar statements from different works and grouped those statements that refer to the same topic, as seen below:

S1 Improvements in coding and in the resulting code:
- S1.1 Improvement of software code quality [44] [10] [34].
- S1.2 Reduction of flaws in software architecture [4].
- S1.3 Improvement of code consistency [4] [10].
- S1.4 Rapid code generation when the application needs to be deployed on distinct platforms [31] [4] or migrated from one platform to another as technology changes [44].
- S1.5 Automatic application of tested software blueprints and industry-standard patterns [10].
- S1.6 Elimination of repetitive coding for the application [44].

S2 Improvements related to models:
- S2.1 Models are always updated with the code [19].
- S2.2 The model becomes the focus of development effort; it is no longer discarded at the outset of coding [44].

S3 Improvements in maintenance:
- S3.1 Improvement in reuse, development of new versions and maintainability [19].
- S3.2 Reduction of intellectual effort required for understanding the system [42] [34].
- S3.3 Mappings provide interoperability among two or more different platforms [44] [17].

S4 Improvements for developers:
- S4.1 Reduction in developer effort [44] [4] [10] [19] [43] [42].
- S4.2 Improvement of productivity [42] [11] [34].
- S4.3 Enhancement of developer satisfaction [30].

Some of these advantages are embedded in the very definition of MDD and have no need of experimental validation. For example:

- Code generation for distinct platforms: the same model can be used to derive code for different programming languages or platforms [44] [31] [4].
- Improvement of code quality: developers do not need special skills to build a good architecture [4].
- Maintainability: if a programming language evolves, the model compiler can be updated with a new version of the code and the developer can effortlessly update the code from the same model automatically [19].

However, most of the claims listed above can be subject to experimental investigation. There exist some works that have gathered empirical evidences about MDD. Some authors have defined specific frameworks to guide the *evaluation of non-trivial MDD advantages*. For instance, Vanderose and Habra [48] define a framework that explicitly includes the various models used during software development as well as their relationships in terms of quality. Since generic frameworks ([50], [9]) have been widely validated and are frequently used in software engineering to evaluate different types of technology, the benefits of using specific frameworks to evaluate MDD are unclear.

MDD has been adopted by some companies. Some authors have described their *experience of applying MDD in industrial settings*. Baker et al. [8] report on 15 years of applying MDD at Motorola and analyse effort, quality and productivity in automatic code generation and automatic test generation. According to their results, effort is 2.3 times less, defects are between 1.2 and 4 times less, and productivity is between 2 and 8 times greater using co-simulation, automatic code generation and model testing.

Some authors aim to extract the *existent experience with MDD at companies*. For example, Hutchinson et al. [21] focus on understanding which factors lead to a successful adoption of MDD. They interviewed 20 professionals by telephone. The participants were from three different companies: a printer company, a car company, and a telecom company. They found that: MDD requires a progressive and iterative approach; successful MDD adoption depends on organisational commitment; MDD users must be motivated to use the new approach; an organisation using MDD needs to adapt its own processes along the way; MDD must have a business focus, where MDD is adopted as a solution to new commercial and organisational challenges.

Notice that both self-experience and surveys elicit opinions, so their findings are empirical but, by definition, subjective.

Other researchers have conducted *case studies* to identify MDD benefits. Mellegard and Staron [32] performed a case study to compare whether developers expend more effort on modelling in MDD than in a traditional method. They interviewed three project managers and measured effort expended on the development of artefacts. Results show that effort expended on modelling is similar using MDD than using a traditional method. Since MDD automatically generates code, the authors deduce that MDD-compliant methods should always be more efficient than traditional methods. Heijstek and Chaudron [20] conducted a case study to evaluate the effort saved by using MDD in industry. One developer, one lead developer, two project leaders and one estimation and measurement officer were interviewed. The authors studied a project to develop a system for a large financial institution. The study focuses on model size, model complexity, model quality and effort to build the models. Results show that, for the project under study: large and complex models built with MDD change more often but do not necessarily contain more defects than smaller models; MDD achieves better results with regard to effort, quality and development complexity. The research also report on the subjective opinions of development team members about benefits of MDD: increase in productivity, consistent implementation, and improvement of the overall quality. Kapteijns et al. [26] performed a case study to analyse the productivity of MDD applied to small middleware applications. The research focuses on the development of a middle-sized system for managing satellite data. The study observes one developer and analyses productivity. They found that productivity was 2.6 times greater when MDD was applied for this specific system.

In summary, MDD case studies mainly focus on measuring effort and find that MDD always scores higher than a traditional method. Notice that the strength of case studies is that they investigate a phenomenon in its real context. The weaknesses are a low level of control (which means that there is no way of knowing what caused the observed results) and low level of generalisation (since the findings are true only for the case under study).

Some researchers have conducted *experimental investigations* of MDD. Anda and Hansen [7] conducted an experiment to analyse the advantages of MDD in legacy systems. The research focused on a project at a large company to develop a safety-critical system. There were 28 experimental subjects, where 14 developed parts of the system from scratch and 14 enhanced existing components. The variables measure the ease of constructing diagrams, use of diagrams and utility of diagrams. Results show that the use of diagrams is beneficial in testing and documentation, and there is a need for more methodological support on the use of diagrams. Krogmann and Becker [28] performed an experiment computing two projects with students. One project used a traditional method where a team of 11 subjects developed a system. While in the other project one subject developed another system using MDD. The studied variable is effort. They found that development effort for a system with MDD is lower than for a traditional method. According to the outcomes, MDD reduces effort by 11%. Notice that time is not measured, it is estimated by developers. Martínez et al. [30] compare three different development paradigms in a controlled laboratory experiment: code-centric development (developers focus only on the code, models are rarely used), model-based development (developers build models that are used to produce small chunks of code) and MDD (developers build models that are used to automatically generate code for part or all of the system). The 26 subjects are divided into five teams. Each team used the three development paradigm but in different order. Each team developed a different web application but all of them shared the same complexity and context (social media applications). The subjects do not build a fully functional web application; they build chunks of code (modules). The studied variables are: perceived usefulness, perceived ease of use, compatibility of each paradigm, and intention to adopt. Results show that the MDD method has the highest perceived usefulness, perceived ease of use and intention to adopt. MDD was the least compatible with developers' current practices but the most useful in the long run. Some of the cons of MDD highlighted by users based on their subjective perceptions are: greater learning curve, lower compatibility of MDD with the other two paradigms and lower reliability of the results. Papotti et al. [38] conducted a controlled laboratory experiment developing part of a medium-sized academic system used at the University of Sao Carlos. The analysis is only focused on the time to develop CRUD (Create, Retrieve, Update, Delete) operations. MDD is compared with a traditional method applied by 29 undergraduate computer engineering students. The variable measured is effort. Results show that MDD reduces development time by 90%, and reduces developers' difficulties by 57% of the subjects.

Table 1. Empirical studies on MDD in the literature

Author	Type of Study	Site	Size	Data Collection	Variables	Results	Limitations
Baker et al. [8]	Self-Experience	Industry	15 years	Interviews	- Effort - Quality - Productivity	- MDD reduces effort in coding and testing - MDD increases quality (1.2-4 times) - MDD improves productivity (2-8 times)	Results are based on subjective opinions of authors.
Hutchinson et al. [21]	Survey	Industry	20 subjects 3 companies	Interviews	Factors for a successful adoption of MDD	Successful MDD use depends on: -Progressive adoption -Integration with existing processes -Organisational commitment	Conclusions extracted from interviews are subjective.
Mellegard and Staron [32]	Case Study	Industry	3 subjects 1 project	Interviews	Effort invested to build models	Model building efforts are similar in MDD and the traditional method	Empirical evaluation is confined to model building effort and omits code generation.
Heijstek and Chaudron [20]		Industry	4 subjects 1 project	Interviews	-Model size -Model complexity -Model quality -Effort to build models	MDD increases: - Model size - Model quality MDD reduces: - Development complexity - Effort	Results extracted from non-quantitative analysis of interviews.
Kapteijns et al. [26]		Industry	1 subject 1 project	Measures	Productivity	MDD improves productivity (function point/hour)	Results only valid for middleware applications.
Anda and Hansen [7]	Experiment	Industry	28 subjects 1 project	Measures	-Ease of constructing diagrams -Use of diagrams -Utility of diagrams	Diagrams improve testing and documentation but there are few diagramming methods	Experiment focuses on diagrams. Benefits of generating code from these diagrams have not been evaluated empirically.
Krogmann and Becker [28]		Academia	11 subjects 2 projects	Measures	Effort	MDD reduces the effort by 11%	-Results focus on development time. -Time logging is not used; analysed times are rough estimates.
Martínez et al. [30]		Academia	26 subjects 5 projects	Measures	-Perceived usefulness -Perceived ease of use -Compatibility -Intention to adopt	MDD has: -The highest perceived usefulness -The highest perceived ease of use -The highest intention to adopt -The least compatibility and reliability	Experiment does not study the development of fully functional systems from scratch. The final product is small chunks of code.
Papotti et al. [38]		Academia	29 subjects 1 project	Measures	Effort	MDD reduces: -Development time -Developer difficulties	-Results focus on development time. -Experiment is limited to CRUD operations.
Bunse et al. [12]		Academia	45 subjects 1 project	Measures	-Reuse -Effort -Quality	MDD reduces effort MDD improves: - Reuse - Quality	Results are only valid for components, not full applications developed from scratch.

Bunse et al. [12] performed an experiment on a component-oriented approach using MDD in embedded software systems. A total of 45 subjects divided into three-member teams develop a car-mirror control system. The experiment focuses on comparing MDD with the Unified Process and an agile approach. The studied variables are: reuse, effort, and quality. The results show that using MDD in a component-oriented approach has a positive impact on reuse, effort, and quality.

The results of most experiments show that it takes less effort (measured as time) to develop a system with MDD than using a traditional method. Notice that only two experiments [28] [38] investigate development of a fully functional system from scratch. Generalisation of results from [28] is troublesome since: the response variable is not measured but it works with participants estimation; each treatment is applied on a different problem/project and compares the development performed by one person with the one performance by a team of 11. [38] is a full randomised laboratory experiment but it develops only CRUD operations.

Other variables such as quality, productivity or developer satisfaction have not been studied in experiments yet. If the focus is on effort only (measured as time), there is no guarantee that the code generated with MDD fulfils end-user expectations nor the MDD is satisfactory for developers. Moreover, most existing experiments focus on one project, which is an obstacle to generalisation to other projects. Only Martínez et al. [30] included the study of different projects (problems) as a factor in the experiment.

Table 1 shows a summary of existent empirical works that have studied the benefits provided by MDD. Most studies compare MDD with a traditional method for developing small chunks of code rather than a full system from scratch. The studies [26] [28] [38] [12] that do deal with a full system use subjective metrics for the comparison [12] or compare productivity [26] and development time [28] [38].

We aim to conduct an experimental investigation to test the most cited benefits (quality, productivity, effort, satisfaction) of MDD. The experimental subjects need to develop a fully functional system from scratch using MDD or a traditional software development method, where code is written manually.

3 Experiment Definition and Planning

The following is a description of the experiment setting according to Juristo and Moreno [25].

3.1 Goal

The goal of this experiment is to compare the MDD paradigm with traditional software development methods for the purpose of filling the existing gap in empirical evidence about MDD. The focus is placed on the differences that appear when building a system from scratch. Of all the existing differences, we focus on product attributes, as well as on developer comfort and workload. The experiment is conducted from the perspective of researchers and practitioners interested in investigating how much better MDD is than traditional software development methods.

3.2 Experimental Subjects

The study *participants* are master students with some professional experience. The subjects participating in the experiment are master students from the Universitat Politècnica de València (UPV, Spain) who have previously taken two software engineering courses. The experiment was performed as part of a MDD course. We recruited 26 students. They all had previous knowledge of the object-oriented paradigm, but few knew anything about MDD before the course. In order to characterise the population, the subjects filled in a demographic questionnaire before running the experiment. Table 2, Table 3 and Table 4 summarise the main characteristics of participants and their background.

Table 2. Job experience at software companies

None	1 month	1-3 months	3-12 months	1-3 year	More 3 years
15	1	0	4	4	2

Table 3. Types of jobs performed by the students and time in the job

		Junior programmer	Senior programmer	Developer	Tester	Manager
Number of students with role		15	1	4	4	4
Time spent in the job (months)	Avg.	11.5	12	24	9	34
	Min.	4	12	12	6	6
	Max.	24	12	36	12	72

Table 4. Experience with MDD and models

Experience with	None	I have heard about it	I took lessons	I have worked with it	I have used it regularly at work
MDD	4	11	8	2	1
Entity-Relationship Diagram	1	3	11	7	4
Class Diagram	0	1	14	7	4
State-Chart Diagram	3	2	11	9	1
Activity Diagram	2	7	10	7	0
Sequence Diagram	2	3	13	7	1
Web Applications Development	6	8	9	0	3

Table 2 focuses on development experience measured by the number of months or years that subjects have worked in industry. Table 3 shows the type of role and the (average, minimum and maximum) time spent in the respective role. Note that some participants have played more than one role. Table 4 shows the previous experience in MDD and several other types of models. Most experiment participants had no work experience, and a few had worked with models. So, our experiment sample can be considered to be representative of a population of novice developers. In the experimental investigation, subjects worked in pairs (mainly for logistic reasons); therefore, when we use the term "pair" in this paper, we mean a couple of participants working together.

3.3 Research Questions and Hypothesis Formulation

There are so many claims about the pros and cons of MDD that it is impossible to evaluate them all in a single experiment. Next, we analyse claims made about MDD in the literature (according to the list we show in Section 2) to select which ones can be studied in this experiment.

Statements focused on code and coding improvements (see S1 in Section 2) depend on the tool that implements the model to code transformation rules, such as statements that refer to reduction of architecture flaws, code consistency, code generation for different platforms, interoperability among platforms, application of automatic tests, and lack of repetitive code. Since our experiment aims to be as independent as possible of any code generation process, we have avoided to analyse the architecture of the generated code, which is generally tool-dependent.

There are model-related statements (S2), such as models are always updated and models are not discarded. These statements have no need of an empirical evaluation because a method that does not guarantee both statements is not MDD compliant.

There has to be an existing system in order to evaluate statements about improved maintenance (S3). This is the case of statements about MDD benefits regarding reuse, development of new versions of a system, maintainability, ease of understanding of the system and portability among different platforms. Since we aim to compare MDD with a traditional method for building a system from scratch, these statements were not considered in our experiment.

The statements that best tie in with our goal deal with product attributes and developers' viewpoints, do not deal with the architecture of the generated code (since architecture is tool-dependent), are not trivial and can be analysed on a system developed from scratch. Of all the MDD advantages stated in the literature, the statements that share these characteristics are statements S1.1 and those of S4: *improvement of software code quality*, *reduction in developer effort*, *improvement of productivity* and *enhancement of developer satisfaction*. The research questions that we have formulated to study these statements are:

- **RQ1:** *Is software quality affected by MDD?* We refer to quality in a broad sense and we adopt the definition by IEEE [22], i.e. the degree to which a system, component or process meets specified requirements. We measure quality as the percentage of end-user requirements satisfied by the developed system. The null hypothesis tested to address this research question is: H_{01}*: The software quality of a system built using MDD is similar to software quality using a traditional method*.
- **RQ2:** *Is developer effort affected by MDD?* Effort is defined as the number of labour units required to complete a schedule activity or work breakdown structure component, usually expressed as person-hours, person-days or person-weeks [22]. We measure effort as the time taken per pair to build a system. The null hypothesis tested to address this research question is: H_{02}*: The developer effort to build a system using MDD is similar to effort using a traditional method.*
- **RQ3:** *Is developer productivity affected by MDD?* Productivity is defined as the ratio of work product to work effort [22]. We operationalise productivity as the amount of quality work done per effort. The null hypothesis tested to address this research question is: H_{03}*: The developer productivity using MDD to build a system is similar to productivity using a traditional method.*
- **RQ4:** *Is developer satisfaction affected by MDD?* Satisfaction is defined as the contentedness with and positive attitudes towards product use [22]. We operationalise satisfaction as how at ease developers are as they develop a system. The null hypotheses being tested to address this research question is: H_{04}*: The developer satisfaction using MDD to build a*

system is similar to satisfaction using a traditional method.

3.4 Factors and Treatments

We now define factors and their levels to operationalise the cause of our experiment construct. Factors are variables whose effect on the response variables we want to understand [25]. The experiment studies one factor: development method. The control in this experiment is *a traditional method* while the treatment is *MDD*.

Regarding the control treatment, there are different approaches for operationalising a traditional development method where developers implement the code manually. Following [30], we have chosen two main approaches that experimental subjects might want to use: code-centric and model-based development. In code-centric development, developers focus mainly on producing the code and seldom use conceptual models. In model-based development, developers build some conceptual models to abstractly represent the system prior to implementation. We aim to compare MDD with each subject's preferred traditional method. This way, MDD is compared under conditions more closely resembling reality (where practitioners have experience using certain paradigms). Since we cannot guarantee that all subjects prefer the model-based paradigm (despite having received training in previous courses), we allowed participants to use their preferred, code-centric or model-based, method. Subjects choose the paradigm they are more familiar with and towards which they have better expectations. Pairs that used the model-based paradigm were free to choose whichever conceptual models they thought best.

Regarding the treatment level, there are several sound MDD methods such as NDT [24], WebRatio [6], OOHDM [41], and so on. Of all the existing methods, we chose OO-Method [40] and its associated tool INTEGRANOVA [2]. OO-Method was chosen for three reasons. First, INTEGRANOVA is one of the few tools being used successfully in industry [2]. Other tools have been exercised in an academic context but not used in industry. Second, INTEGRANOVA is one of the few tools that can generate fully functional systems without writing a single line of code. Other MDD tools require some code to implement part of the functionality or system interfaces that are not supported by models. This makes INTEGRANOVA fully compliant with the MDD paradigm, which advocates focusing the entire developer effort on building conceptual models and relegates code generation to automatic transformations. Third, INTEGRANOVA can generate code in different languages using the same conceptual model as input, such as C# and Java. The tool can generate code for desktop and web systems from the same model too. All these features make OO-Method and its tool the perfect choice for embodying the treatment level to be used by participants to generate a fully operational system from scratch. Appendix A contains a brief description of the models used by INTEGRANOVA.

3.5 Response Variables and Metrics

Response variables are the effects studied in the experiment caused by the manipulation of factors [25]. **RQ1** requires a variable to measure quality. According to ISO 9126-1 [3], quality is composed of several characteristics: functionality, reliability, usability, efficiency, maintainability, portability. We have chosen functionality since it is more focused on satisfying end-user expectations. More specifically, we study the sub-characteristic **Accuracy**, which is defined in ISO 9126-1 as "the capability of the software product to provide the right or agreed results or effects". We measure accuracy as the percentage of acceptance test cases that are successfully passed.

We have defined a set of acceptance test cases that cover each web application. Each requirement has one associated test case. Once the development process was complete, we executed the set of acceptance test cases in the presence of the pairs. This way, the developer pairs could help us setup the system (the setup process is outside the scope of our experimental investigation). Each test case is defined as a sequence of steps; we consider each one of these steps as an item that needs to be satisfied. We used four aggregation metrics to decide whether a test case passes or not:

- **All or nothing:** we consider that a test case is satisfied only if every item is passed (100% items passed). If one item failed the whole test case failed. The test case is seen as a black box with two possible values: success or failure (1 or 0). To aggregate the results of all test cases in a test suite, we calculate the percentage of test cases run successfully.
- **Relaxed all or nothing:** we consider that a test case is satisfied when at least 75% of items are passed. The test case is again seen as a black box with two values: success (1) or failure (0). To aggregate the results of all test cases in a test suite, we aggregate the percentage of test cases run successfully.
- **Weighted items:** we assign a weight to each test item depending on the complexity of its functionality. The complexity of functionalities is decided by the problem designer. Weights are directly proportional to the complexity of the functionality. Weights are assigned in such a way that the addition of all weights is 1 for each test case. When test cases are run, we add the weights of passed items. The test case returns a value between 0 (no item has passed the test) and 1 (every item has passed the test), including decimals. This is the percentage of passed items in a test case. To aggregate the percentages of all the test cases, we calculate the average.
- **Same weight for all items:** we assign the same weight to each item within a test case (independently of complexity) in such a way that the addition of all the weights of the items is 1 per test case. This avoids the subjectivity of the weighting item metric. When test cases are run, we add the weights of passed items. The test case returns a value be-

tween 0 (no item has passed the test) and 1 (all items have passed the test), including decimals. This shows the percentage of passed items in a test case. The aggregation of all test cases is calculated in the same way as for the weighting item aggregation metric, that is, to aggregate the percentages of all the test cases, we calculate the average.

Table 5 shows how many test cases were used to evaluate each problem and how many items there are in each test case. Test cases for both Problem 1 and Problem 2 are shown in Appendix B and Appendix C respectively. Some items appear in more than one test case. In an invoicing system, for example, if we aim to evaluate whether or not the system automatically calculates the amount of the invoice using the product price, we need to add more than one product to the invoice, repeating items to insert products. Table 5 also shows how many items appeared more than once throughout all the test cases. Problem 1 has more repeated items since the "Create invoice" test case needs a lot of information to have been previously saved in the system. Repeated evaluation items are considered only once (first time they are tested). This results in 21 items for Problem 1 and 22 items for Problem 2. Since problem complexities are similar, the number of items is also similar.

Table 5. Test items of each problem

Problem 1		
Test case	**Items**	**Repeated Items**
1. Create customer	8	0
2. Create repair card	28	22
3. Create invoice	7	0

Problem 2		
Test case	**Items**	**Repeated items**
1. Create application	12	0
2. Create application for the same photographer	6	4
3. Approve application	3	0
4. Promote photographer	5	0

RQ2 needs a dependent variable that measures effort. Since effort is the ratio of time to develop a system per developer [22], we measure **effort** as the time taken by each pair to develop the web applications from scratch.

RQ3 requires a dependent variable that measures developer **productivity**. Since productivity is the ratio of quality work to effort [22], we measure productivity as the accuracy to effort ratio (accuracy/effort).

RQ4 requires a dependent variable that measures developer **satisfaction**. Since satisfaction is the positive attitude towards the use of the development method [22], we measure satisfaction using a 5-point Likert scale questionnaire. The instrument used to measure this variable is a satisfaction questionnaire built using the framework developed by Moody [36]. Moody defined a framework (based on the work by Lindland [29]) in order to evaluate model quality in terms of perceived usefulness (PU), perceived ease of use (PEOU), and intention to use (ITU). This framework has been previously validated and is widely used [36]. According to [36], we defined eight questions to measure PU, six questions to measure PEOU and two questions to measure ITU. We defined a questionnaire for each treatment (MDD and traditional method). Even though the meaning of each question was the same for both levels, each questionnaire includes terms specific to the level that we aim to measure. For example, the statement *"I will definitely use MDD to develop web applications"* is used to measure the satisfaction with MDD, whereas the statement *"I will definitely use a traditional method to develop web applications"* is used to measure the satisfaction with the traditional method.

Note that all four response variables are to be measured after developing an executable web application from scratch. By executable we mean a system that can be used by an end user. This is independent of the percentage of functionality that the system provides with regard to its specification. Table 6 shows a summary of the research questions, hypotheses and response variables used to test these hypotheses.

Table 6. Summary of RQs, hypotheses and response variables

RQs	Hypotheses	Response Variables	Metric
RQ1	$H_0 1$	Accuracy	Test cases passed
RQ2	$H_0 2$	Effort	Time
RQ3	$H_0 3$	Productivity	Accuracy/effort
RQ4	$H_0 4$	Satisfaction	PU, PEOU, ITU

3.6 Experiment Design

This section discusses the design alternatives that we considered to limit the threats to validity brought about in this domain. Design names used here follow Juristo and Moreno [25]. Notice that design names depend on the discipline; designs with different names in different disciplines may actually be equivalent.

Before selecting a specific design for the experiment, we considered several alternatives, which were ruled out based on a trade-off between threats to validity and context characteristics. We share this information hoping to provide insights to researchers undertaking similar experiments in the future and to clarify the rationale for our experimental design. The designs we analysed and discarded due to their threats are the following:

- **One factor, two treatments:** This design divides units into two groups: one group (G1) develops the system with MDD and the other group (G2) develops the system with a traditional software development method. Each group solves the same problem (P1) (Table 7.a). The pros of this design are that the conditions for both treatments are identical (same session, same problem). The main threat of this design is that the number of experimental units is divided by two. We do not have a large enough number of experimental units (in this case, pairs of subjects) to get powerful results using this design. Besides, from the training perspective, pairs would only practice one software development method (MDD or traditional method), which is not a viable alternative in a MDD course.
- **Paired design:** In this design, pairs are not divided into groups so as not to reduce sample size. The experiment is performed during two sessions. In the first session, all pairs solve a problem with a traditional software development method. In a second session, all pairs solve the same problem with MDD (Table 7.b). The pros of this design are that the comparison conditions are still much the same (a single problem), we use the largest possible sample size with the available number of experimental units and pair variability does not affect levels since the same units are used in both levels (subjects act as their own control, eliminating variability due to differences in subject behaviour). This design has two main threats: pairs may learn the problem in the first session and the results of the second session may be influenced or biased by a problem learning effect; treatment and order are confounded since one level is always exercised first and the other second.

Table 7 Discarded designs

(a) One factor, two treatments design

		P1
Session 1	**MDD**	G1
	Traditional	G2

(b) Paired design

		P1
Session 1	**Traditional**	G1
Session 2	**MDD**	G1

- **Paired with two experimental objects:** This design aims to avoid the possibility of the learning effect in the paired design. One problem is used with a traditional software development process and a different problem with MDD (Table 8.a). The pros of this design are that it uses the largest possible sample size, the between-session learning effect problem is removed and variability among pairs does not affect levels. This design has one main threat: the problem (P1 and P2) to be solved and the treatment (MDD and a traditional method) are confounded.
- **Cross-over with two experimental objects:** To avoid the threat of confounding treatment and order, we considered a cross-over design where experimental units are divided into two groups (G1 and G2). In the first session, G1 solves problem P1 with MDD, while the other group (G2) solves the same problem with a traditional software development method. In a second session, both groups solve another problem (P2) using the other development method (Table 8.b). The pros of this design are that both treatments are applied in each session (avoiding the confusion between session or order and treatment), it uses the largest possible sample size, there is no problem learning effect, the problem is not confounded with levels, and there is no variability due to differences in average subject responsiveness. This design has one main threat: it is not easy in the context of a MDD course to justify the use of the traditional method as a treatment in session 2 once subjects have used MDD in session 1.

Table 8. Discarded designs

(a) Paired design with two experimental objects

		P1	P2
Session 1	**Traditional**	G1	-
Session 2	**MDD**	-	G1

(b) Cross-over design with two experimental objects

		P1	P2
Session 1	**MDD**	G1	-
	Traditional	G2	-
Session 2	**MDD**	-	G2
	Traditional	-	G1

After discarding all the above designs for our experimental investigation, we finally chose **a paired design blocked by experimental objects** [25] shown in Table 9. We divided the pairs into two groups (G1 and G2). Both groups used a traditional software development method in the first session and MDD in the second session. We blocked by problem to avoid the learning effect of using the same problem in both sessions.

Note that the design that we use (Table 9) avoids most of the threats: experiment findings are not fully problem dependent (since we use two problems); it provides the largest possible sample size with the number of available units; there is no between-session problem learning effect; pairs cannot share their solutions with members of other groups since all units work at the same time and they cannot talk each other during the session; all units are used in both levels avoiding variability among units; from an educational viewpoint, it makes sense to begin the course with a hands-on reminder of the traditional development method and finish with the hands-on exercise using MDD.

The design applied in our experiment also has some intrinsic threats: the context of the experiment in session 1 might not be the same as in session 2 (boredom, noise in the room, tiredness, etc.); since MDD is always applied in the second session, subjects might carry over something from the traditional development to MDD. However, since the traditional-

MDD order is the regular order practitioners will use (they know a traditional paradigm and move to MDD), we think this experiment design matches the reality under study.

Table 9. Design used in the experiment: paired design blocked by experimental objects

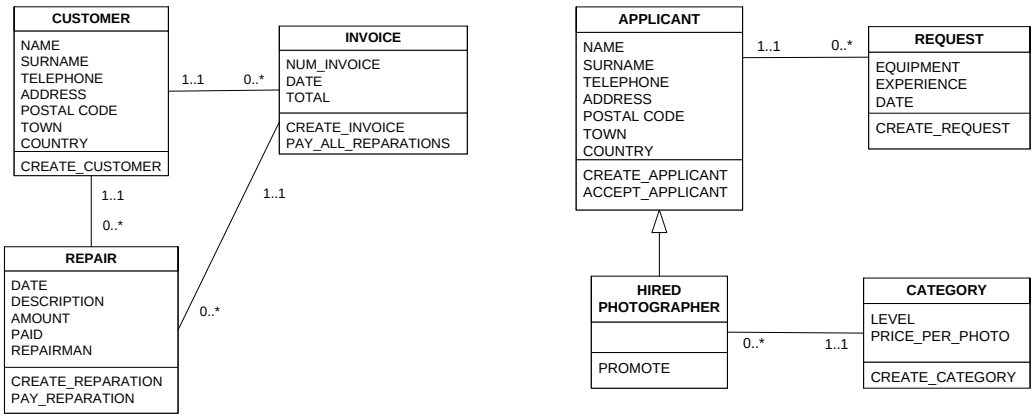
		P1	P2
Session 1	**Traditional**	G1	G2
Session 2	**MDD**	G2	G1

3.7 Experimental Objects

The objects used in the experimental investigation are two requirements specifications created for this experiment. They both are information systems required to be implemented as web applications. According to the lessons schedule during which the experiment was run, we had four hours (two sessions) to apply each treatment. Using a real web application to obtain outcomes applicable to real systems would be the best case. However, if we had defined a complex real problem for each treatment, participants would have not been able to finish the implementation of the problem due to time constraints. It would then have been impossible to execute and run test cases on the web applications created by subjects, which we set out to do in order to evaluate MDD in full system operating conditions. To ensure that every pair produces an executable system, we defined simple problems that could be tackled in four hours. The domain and complexity of both problems is analogous to remove or mitigate the effect of problems on the results.

Both requirements are specified according to IEEE standard 830-1998 [45]. One problem is to manage a company of electrical appliance repair and the other is a photographer agency management system. Even though the problems are small, they exploit the most significant modelling primitives of MDD: several classes with attributes, methods and relationships; conditions to be satisfied before triggering methods; transactions (methods composed of several methods following the principle of atomicity); and formulas to calculate derived attributes (attributes that depend on other attributes).

The system to manage a company of electrical appliance (*Problem 1*) stores the customer information. Every time a customer's appliance breaks down, he asks for a repair. Once the problem has been fixed by a repairman, a repair record is stored in the system. Later on, when the customer is ready to pay, the system creates an invoice that includes all the unpaid repair records. The invoice must calculate the total amount to be paid depending on the amount of each outstanding repair. The most common actions within this system are: create customers, create repair cards, create invoices (which can include several repair cards for the same customer), and payment of all the repairs included in an invoice. This system has 40 function points considering classes, attributes and operations shown in Fig 1.a . Note that INTEGRANOVA generates automatically CRUD operations for each class, apart from operations specific of the problem[1]. If we measure the size of Problem 1 including CRUD operations, we obtain 100 function points. Since CRUD operations were not within the problem description, we consider Problem 1 in the experiment with a size of 40 function points.



(a) Class Diagram of Problem 1 (b) Class Diagram of Problem 2

Fig 1. Reference solutions to the problems solved during the experimental tasks

The photographer agency management system (*Problem 2*) stores the information on photographers that are working at an agency. New photographers fill in a job application. Later on, the agency manager decides which applicants are accepted and stores the accepted applicants as new photographers. Applicants whose application has been rejected can reapply at the earliest one month later. In this case, only the applicant's short bio and photography equipment must be updated. Photogra-

[1] CRUD operations are generated by default with INTEGRANOVA, however, the analyst can hide them.

phers may be promoted when the quality of their photographs improve. Photographers cannot be demoted. The most common actions within this system are: create application, accept or reject applicants and promote photographers. Fig 1.b shows the class diagram for Problem 2. This system has 35 function points considering classes, attributes and operations shown in Fig 1.b. If we consider CRUD operations included by INTEGRANOVA, we obtain 94 function points. Since CRUD operations were not in the problem description, we consider Problem 2 in the experiment with a size of 35 function points.

3.9 Experiment Procedure

The procedure for running the experimental investigation matches the chosen design. The investigation was carried out over four months with two-hour sessions once a week, that is, a total of 16 training sessions. The diagram in Fig 2 summarises the procedure. The numbers mean steps in the experimental procedure, not sessions.

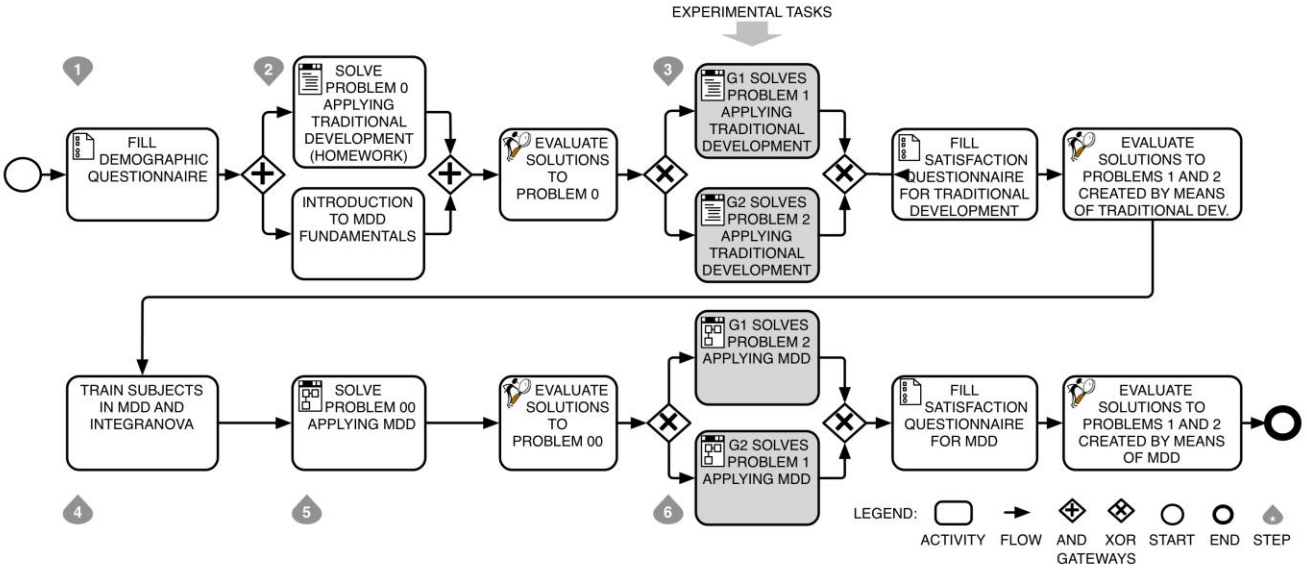


Fig 2. Summary of how the experiment was performed

The first step in the first session was to fill in the demographic questionnaire to identify the students' background. The results of this questionnaire are shown in Table 2, Table 3 and Table 4. In the second step (sessions 1 to 3), the pairs participating in the experiment develop a web application from scratch using a traditional method. However, training in a traditional development method is outside the scope of the course. In order to ensure that all the pairs were able to apply a traditional method, we defined a problem (Problem 0) to be solved as homework over a two-week period. The problem was designed as a refresher for pairs to practise a familiar development paradigm and programming language. During the first session, pairs were given the requirements specification of Problem 0 (using the IEEE standard 830-1998 notation [45]) to be developed using a traditional method. Problem 0 was a public transport bus management system. System users can perform actions such as: create buses, define routes, identify passengers getting on a bus, identify passengers getting off a bus, and provide historical information of bus journeys per passenger. The class diagram of this system can be viewed in Appendix Da.

Table 10. Programming languages used by the pairs for traditional treatment

Language	Number of pairs
PHP	2
.NET	7
JAVA	3
RUBY	1

For this refresher exercise, we advised the pairs to choose the language that they had used most in the past. Table 10 shows the distribution of the programming languages among the pairs. Pairs were allowed to use any diagrams they liked (such as UML diagrams or entity-relationship diagrams) before programming the web application. Of 13 pairs, 10 used a UML class diagram to represent the database structure. The other two pairs started to program the system directly. At the end of the third session, the trainer evaluated Problem 0 to verify that the pairs were eligible for the experimental investigation. All the systems passed most of the test cases designed for evaluation purposes; therefore, we can ensure that all the

pairs have sound knowledge of a traditional (either model-based or code-centric) software development method. While the pairs were developing Problem 0 as homework, MDD was introduced in two lessons (four hours).

In the third step (sessions 4 and 5), we divided the pairs into two groups of six pairs per group. Each group developed a system to solve a problem (Problem 1 or Problem 2) with a traditional method using the same programming language they had used in the refresher exercise. The distribution of problems by pair was random, ensuring that it was balanced. Pairs 2,4,5,7,8,11,12 solved Problem 1 and pairs 1,3,6,9,10,13 solved Problem 2. Developments with the control treatment were performed in two classroom sessions (four hours) so that we could control time. Pairs could use any diagram that they needed prior to implementation. Out of 13 pairs, seven used a UML class diagram to represent the database structure. The other six pairs did not use any diagram and went on to code directly. In order to ensure that the pairs did not develop the system at home (using up extra time), we checked that the development starting-point of fifth session matched the end-point of the fourth session. At the end of the fifth session, we evaluated the developed system with the test cases, and the pairs filled in a satisfaction questionnaire regarding their experience with the traditional method.

In the fourth step (sessions 6 to 11), the trainer spent 12 hours explaining the INTEGRANOVA tool. At the end of the learning period, in the fifth step (sessions 12 to 14), the pairs developed a web application using MDD and INTEGRANOVA on their own as hands-on training. We set a new problem (Problem 00) quite similar in complexity and functionality to Problem 0. This way we ensured that both treatments were applied under similar training conditions. Problem 00 is a video club management system. System users can perform actions such as: create movies, create members, rent a film, return a film, and provide statistical information per member. The class diagram of Problem 00 is in Appendix Db.

Problem 00 was developed in the classroom and the students were also allowed to work at home. At the end of session 14, we evaluated the developed systems through test cases. Since all the web applications passed most of the test cases, we concluded that students had enough knowledge to participate in the experimental investigation.

In the sixth step (sessions 15 and 16), the pairs swap problems (Problem 1 and Problem 2) and developed them using INTEGRANOVA. Again, we ensured that the system was developed exclusively in the classroom, and we timed the development. At the end of session 16 we evaluated the developed system with the same test cases that we had used in the manual implementation, since these are the same applications. The pairs also filled in the satisfaction questionnaire with regard to their experience with MDD.

3.10 Evaluation of Validity

Table 11 discusses (following [23]) the threats addressed (or not) by the experiment setting (as opposed to the threats arising during execution or analysis). We have classified the threats according to the classification provided by Wohlin [49] following Campbell [13]. We organised the threats of each type according to three groups: avoided, suffered and not applicable. The threats and how they were dealt with are shown in Table 11.

3.11 Data Analysis

We used the repeated measures (or within subjects) general linear model (GLM) to analyse the collected data since both levels of the Development Method factor are applied to each pair (within subjects). The blocking variable, Problem, is introduced as a covariate in the GLM test. There are two requirements for applying a GLM test: homogeneity of the covariance matrices and sphericity. Box's M test is used to check the condition of homogeneity of covariance matrices using as null hypothesis that the observed covariance matrices of the dependent variables are equal across groups [35]. For our sample, $M = 37.6$, $F = 1.298$, $df1 = 21$, $df2 = 2118.527$, sig. = 0.164, that is, the results verify the null hypothesis, meaning that the data are homogeneous. Mauchly's test is used to check the sphericity condition. In our case, however, there are only two levels of repeated measures (with MDD and with a traditional method), which precludes a sphericity violation [35] and the test is unnecessary.

The power of any statistical test is defined as the probability of rejecting a false null hypothesis. Statistical power is inversely related to beta or the probability of making a Type II error. In short, power = $1 - \beta$. Power in software engineering experiments tends to be low, e.g. Dyba et al. [14] reports values of 0.39 for medium effect sizes and 0.63 for large effect sizes. Low values of power mean that non-significant results may involve accepting null hypotheses when they are false.

The p-value of GLM test shows whether or not there is a significant difference between treatments of each factor. Since the test does not indicate the magnitude of that difference, we use the non-parametric statistic called Cliff's delta as the effect size measure [18]. We use this technique since accuracy, productivity and effort do not follow a normal distribution. Cliff's delta ranges in the interval [−1, 1] and is considered small for $0.148 \leq d < 0.33$, medium for $0.33 \leq d < 0.474$, and large for $d \geq 0.474$. A positive sign means that values of the first treatment are greater than the values of the second one (inversely for a negative sign). The results, grouped by accuracy, productivity, effort and satisfaction, of applying GLM, statistical power and Cliff's delta follow.

Table 11. Evaluation of Validity

Type of threat	Status	Threat	Due to	How we have dealt with it
Conclusion validity	Avoided	Low statistical power	Sample size is not big enough.	We avoided this threat using the repeated measures GLM test and calculating the statistical power for each null hypothesis that we were unable to reject.
		Subjects of random heterogeneity	Subjects are randomly selected and their background is too heterogeneous.	We avoided this threat by training the pairs with Problem 0 and Problem 00.
		Fishing	Experimenters search for a specific result.	We avoided this threat by using all the collected data, none of which was removed for any reason whatsoever.
		Reliability of measures	There is no guarantee that the outcomes will be the same if a phenomenon is measured twice.	We avoided this threat by measuring accuracy, effort and productivity with objective metrics.
	Suffered			Satisfaction suffers from this threat since it is subjective.
		Random irrelevancies in experimental settings	There are elements outside the experimental setting that may interfere with the results.	The experiment suffers from this threat since, we cannot guarantee that none of the subjects spent a little time on activities different from the experiment, such as chatting with their partner or reading e-mail.
Construct validity	Avoided	Interaction of testing and treatment	Subjects apply the metrics to the treatments.	We avoided this threat since experimenters (the researchers) were responsible for measuring the treatments.
		Mono-method bias	Experiments with a single type of measure can result in measurement bias.	We avoided this threat to satisfaction, since the satisfaction questionnaire includes redundant questions expressed in different ways. We have avoided this threat to accuracy by using test cases aggregated through three aggregation metrics.
	Suffered			Effort and productivity suffer from this threat since they cannot be cross-checked against each other. To minimise its effect, we mechanised the measurement as much as possible by means of automatic timing.
		Hypothesis guessing	Subjects guess the aim of the experiment and act conditionally upon it.	The experiment suffers from this threat, which we minimised by not talking about the research questions.
		Evaluation apprehension	Subjects are afraid of being evaluated.	The experiment suffers from this threat, which we minimised by including the experimental tasks as exercises that students had to complete to pass the course without mentioning the term “experiment” or “test”.
		Interaction of different treatments	There is no way of concluding whether the effect is due to either of the treatments or to a combination of several treatments.	The experiment suffers from this threat since the control treatment is always applied first. We cannot assure that the order of applying treatments does not affect the results.
		Mono-operation bias	A single operationalisation of treatments can lead to bias.	The experiment suffers from this threat since the development with the MDD paradigm is based on the use of INTEGRANOVA only. It would be risky to generalise the results of the experiment to any other MDD tool than INTEGRANOVA.
Internal validity	Avoided	History	Different treatments are applied to the same object at different times.	We avoided this threat by doing the second session as soon as possible (seven days later). We avoided copies between students by forcing the pairs to start the second session with the last version of the assignment

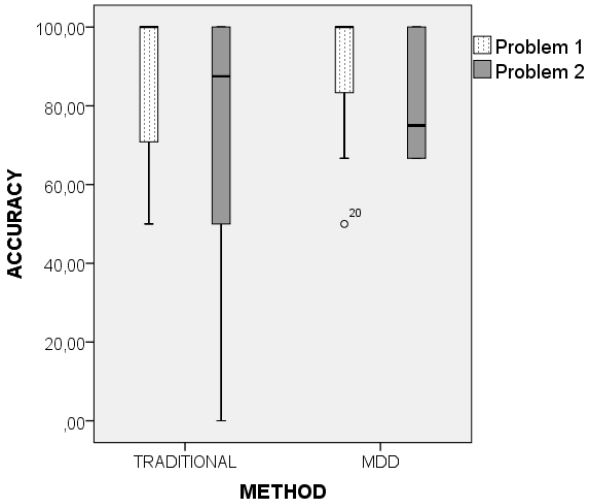
				that they uploaded at the end of the first session.
		Learning of objects	Pairs can acquire knowledge with the first treatment and apply it to the second one.	We avoided this threat by using two different problems in our design, so subjects do not get the chance to learn objects.
		Subject motivation	Less motivated pairs may achieve worse results than highly motivated pairs.	We avoided this threat by choosing a design that fits the goal of the course to avoid demotivation. Since the topic of the subject was MDD, pairs might find it a little frustrating if they had to develop the case study using a traditional method after several lessons on MDD.
		Maturation	The pairs react differently as time passes. This might happen in the second treatment, when pairs could have learned from the first one.	We avoided this threat in our design by applying a traditional method first and MDD second, since this is the regular order in reality for the learning process.
	Suffered	Selection	The outcomes can be affected by how the subjects are selected.	The experiment suffers from this threat since we recruited students from a master's course, and they had to participate in the experiment to pass it.
	Not applicable	Resentful demoralisation	Subjects are divided into groups and only one treatment is applied to each group.	This threat does not apply to the experiment according to our design where all subjects work with both treatments.
		Mortality	Some subjects leave the experiment before completion.	This threat does not apply to the experiment since students had to participate in order to pass the course. There were no drop-outs.
		Compensatory rivalry	Pairs receiving less desirable treatments may be motivated to reduce the expected outcomes.	This threat does not apply to the experiment since we allocated both treatments to all pairs.
External validity	Suffered	Interaction of selection and treatment	The subject population is not representative of the population that we want to generalise.	The experiment suffers from this threat since, according to demographic questionnaire, our subjects have a similar background. Results are valid for people with profiles similar to our subjects.
		Object dependency	The results may depend on the objects used in the experiment and they cannot be generalised.	The experiment suffers from this threat, which we mitigated somewhat by using two objects for each treatment.
		Interaction of history and treatment	Treatments are applied on different days and the circumstances on that day affect the results.	The experiment suffers from this threat since the two treatments are separated by several sessions in our design because pairs needed MDD lessons and training before solving the problem using MDD. We minimised this threat applying both treatments in the same room and the same schedule.
	Not applicable	Interaction of setting and treatment	The elements used in the experiment are obsolete.	This threat does not apply to the experiment since we used recently published and validated questionnaires, and INTERANOVA is now in use.

4 Results

In this section, we report the quantitative results of our experiment in order to address the research questions. All analyses have been performed using SPSS V20. Row data is included in Appendix E for disclosure and re-analysis purposes.

4.1 Accuracy

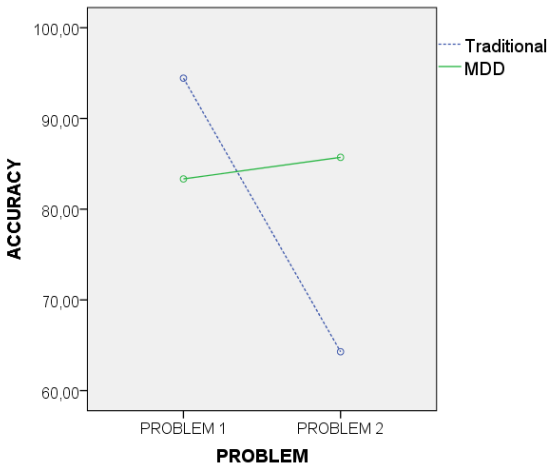
Accuracy was measured as the percentage of test cases passed (the higher percentage, the best accuracy). As discussed in Section 3.8, we aggregated the items within a test case using three aggregation metrics: *all or nothing, relaxed all or nothing, weighted items* and *same weight for all items*. First, we focus our analysis on the most conservative metric (**all or nothing**). Fig 3.a shows the box-and-whisker plot with the percentage of test cased passed using the all or nothing aggregation metric. This plot compares accuracy between the traditional method and MDD, differentiating between the two problems. We find that the difference between the medians of Problem 1 and Problem 2 is more obvious for the traditional development method. All the subjects that used a traditional method to solve Problem 1 achieved a 100% success rate, except for one subject with 66%. The variability of Problem 2 is higher, to the point one pair did not pass any test case. This does not apply when pairs use MDD (even though they have less experience with MDD). Problem 1 and Problem 2 overlap almost completely; all subjects show accuracies roughly between 60% and 100%. MDD appears to be more robust than the traditional method being less affected by developers' abilities. The box-and-whisker plots for the other aggregation metrics are similar to Fig 3.a and they are not reproduced for reasons of space, but the data analysis results for such metrics are discussed later in this section.



ACCURACY

Source	Method	Type III Sum of Squares	df	Mean Square	F	Sig.
Method	Linear	171,987	1	171,987	,689	,424
Method * Problem	Linear	1710,628	1	1710,628	6,856	,024
Error(Method)	Linear	2744,763	11	249,524		

Fig 3.(a) Box-and-whisker plot for Accuracy with the all or nothing aggregation metric. (b) Output of the GLM test for accuracy with all or nothing aggregation metric



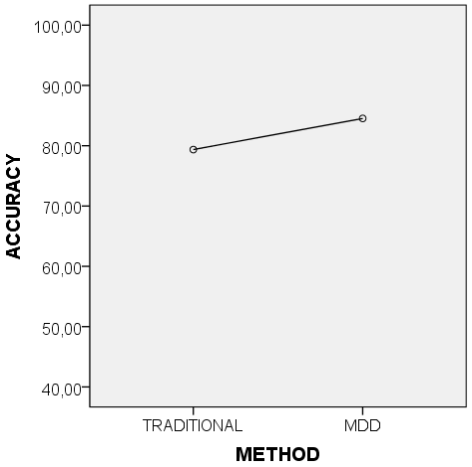


Fig 4.(a) Profile plot of the Development Method*Problem interaction. (b) Profile Plot of both methods

In order to analyse in more detail the robustness of MDD regarding problem complexity observed with the box-and-whiskers plot (Fig 3.a), we have drawn the profile plot in Fig 4.a. This shows an apparent Development Method*Problem interaction, more acute in the case of the traditional method, although MDD is also affected. Differences between Development Methods are not observed, as shown in Fig 4.b.

We applied the GLM test to analyse the percentage of satisfied test cases for each aggregation metric to learn whether or not accuracy is independent of using MDD or traditional methods (H_{01}), shown in Fig 3.b. The p-value of the Development Method factor using the all or nothing metric is 0.42. Therefore, we cannot reject the null hypothesis, that is there is a not a significant difference regarding accuracy between developers using a traditional method and developers using MDD. Statistical power scores 0.12 (a low power), meaning a larger sample size is needed in order to increase experiment power and be able to identify differences (if they exist). Larger sample sizes could reveal if our results is a false negative case. Cliff's delta has the value -0.05, which means that the effect of Development Method is tiny.

As indicated above, the GLM confirms there is a significant Development Method*Problem interaction with a p-value of 0.02. Comparing the means of the possible combination between Development Method and Problem (Fig 4.a), we conclude that when pairs use a traditional method, the accuracy is worse for Problem 2 than for Problem 1. However, this difference between problems does not appear when subjects use the MDD method. The statistical analysis confirms the results of the visual inspection, which suggest that MDD seems to be a less sensitive development method to problem characteristics. The value of Cliff's delta is 0.57 when we compare Problem 1 with Problem 2 for pairs that used a traditional method. This is a large effect and reflects huge differences between both problems[2] that are affecting developers. Cliff's delta is 0 when we compare Problem 1 with Problem 2 for pairs working with MDD. This means that accuracy for Problem 1 is better than for Problem 2 for pairs using a traditional method only. Even though both problems had similar number of function points, we believe that an issue of inheritance present in Problem 2 but not in Problem 1 (see Fig 1.a and Fig 1.b) might be a reason for the different performance observed between the two problems.

From all these results we deduce that H_{01} can be accepted. However, accuracy in MDD is significantly more robust to small variations in problem complexity.

When we use **weighted items** and **same weight for all items** as aggregation metrics, the results for the GLM test, Cliff's delta and statistical power are similar to all or nothing aggregation metric. A summary of the key statistical parameters is shown in Table 12. There are no significant differences for the Development Method factor but there are significant differences between the Development Method*Problem interaction. Note that statistical power is not a problem when the GLM test rejects the null hypotheses because a Type-II error cannot exist in this case (only a Type-I error). In such cases, power has been specified as "N/A" in Table 12.

When we use the **relaxed all or nothing** aggregation metric, there are no significant differences between a traditional method and MDD either, as can be seen in the summary of Table 12. The difference regarding the above aggregation metrics is that relaxed all or nothing does not make a significant difference to the Development Method*Problem interaction. This means that problem complexity does not affect the results if the concept of quality is relaxed. This result makes sense since the differences in quality between Problem 1 and Problem 2 are due to a small set of items in the test case (around 11% of items) that exercise the inheritance. If we consider that a system has good quality when it passes 75% of the items in each test case, we overlook the small difference between the two problems in the omitted 25% of items.

Table 12. Analysis of other aggregation metrics for Accuracy

Aggregation Metric	P-value		Power		Cliff's Delta		
	DM	DM*P	DM	DM*P	Trad. vs MDD	P1 vs P2 with Trad.	P1 vs P2 with MDD
Weighted items	0.22	0.05	0.21	N/A	0.00	0.4	0
Same weight for all items	0.24	0.05	0.20	N/A	0.00	0.4	0
Relaxed all or nothing	0.16	0.18	0.25	0.25	-0.15	0.28	0

DM means Development Method, P means Problem and "*" means interaction

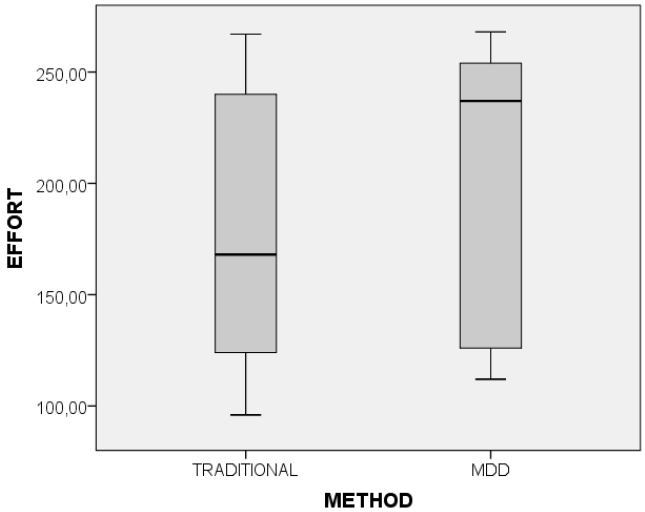
Then results for accuracy do not change using different metrics and we can conclude, as the analysis of problem complexity shows, that MDD seems to be less sensitive to problem complexity than traditional methods.

4.2 Effort

Effort is measured as the time that pairs take to solve a problem (the less time spent in the development, the best effort). Fig 5.a shows the box-and-whisker plot comparing effort (in seconds) expended using a traditional method versus MDD. The first and third quartiles are very similar, but there is a marked difference between medians. The median for effort using a traditional software development method is lower than for MDD since best performing subjects (i.e. those in the first and

[2] Please, remember both problems have a similar complexity measured in function points as well as be similar in the type of functionality.

second quartile) using the traditional method tend to be faster than those using MDD. Although the worst performing ones (i.e. those in the third and fourth quartile) perform rather similar in the traditional development and MDD.



EFFORT

Source	Method	Type III Sum of Squares	df	Mean Square	F	Sig.
Method	Linear	3521,539	1	3521,539	1,560	,238
Method * Problem	Linear	272,001	1	272,001	,121	,735
Error(Method)	Linear	24829,845	11	2257,259		

Fig 5. Box-and-whisker plot for Effort. (b) Output of the GLM test for effort

The aim of our experimental investigation is to identify whether or not effort is independent of using a traditional method or MDD (H_{02}). The p-value for Development Method is 0.24, indicating no significant differences between MDD and traditional methods. Statistical power scores 0.21, which implies that a Type-II error might be happening and therefore results could reverse with bigger sample size. Cliff's delta is -0.21, denoting small effect in the direction that MDD takes a little more effort than traditional development.

The Development Method*Problem interaction is not significant with a p-value of 0.73, which means that problem type does not influence effort. The statistical power of the interaction between Development Method and Problem is 0.06, which is also remarkably low. Cliff's delta comparing Problem 1 with Problem 2 for pairs using a traditional method is -0.26, which is a small effect size. This result shows that subjects expended less effort on Problem 1. Cliff's delta comparing Problem 1 with Problem 2 for pairs using MDD is 0.16, which is also small. This result shows that subjects using MDD solve Problem 2 more effortlessly than Problem 1. When the problem complexity increases (Problem 2 is more complex than Problem 1), MDD gets slightly better results for effort than a traditional method, even though this difference is not significant.

We conclude that H_{02} is accepted: there is no significant difference between effort using a traditional method and effort using MDD, although MDD tends to require slightly more effort than a traditional method.

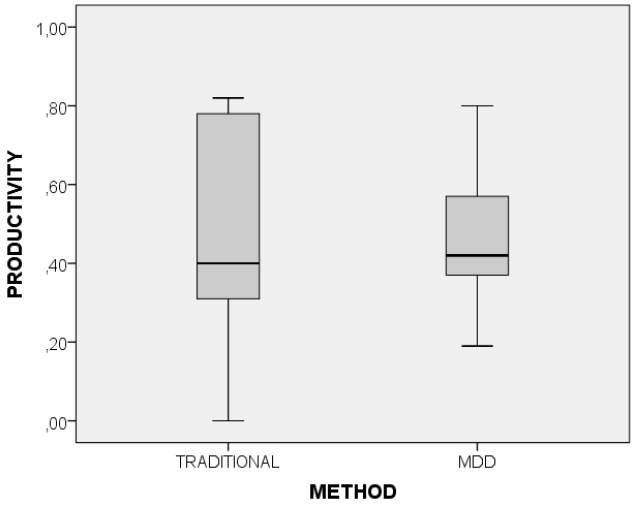
4.3 Productivity

Productivity is measured as the accuracy to effort ratio (the higher ration the best productivity). Fig 6.a shows the box-and-whisker plot comparing productivity using a traditional method and MDD. The median matches for a traditional method and for MDD, but the third quartile has a higher value for a traditional method. This means that high productive pairs (the top 50% above the median) achieve better results with a traditional method than with MDD.

The aim of the experiment is to identify differences between MDD and a traditional method with regard to productivity (H_{03}). Fig 6.b shows the results of the GLM analysis. The p-value for Development Method is 0.42, indicating that there are no significant differences between MDD and a traditional method for productivity. Statistical power is 0.09, so results may be different with a bigger sample size. The value of Cliff's delta is -0.12, which means that the effect size is small, and the MDD group results for productivity are poorer than for the control group.

The Development Method*Problem interaction is not significant, with a p-value of 0.14; hence problem type does not affect productivity. The statistical power of the interaction is 0.12, which is also very low. Cliff's delta comparing Problem 1 with Problem 2 for pairs using a traditional method is 0.5, which is a large effect size. This result shows that productivity is better for Problem 1 using a traditional method. Cliff's delta comparing Problem 1 with Problem 2 for pairs using MDD is -0.14, which is a small effect size. Results show that differences of productivity between Problem 1 and Problem 2 working with a traditional method are greater than when working with MDD, even though these differences are not significant.

We conclude that H_{03} is accepted. There is not a significant difference between the productivity of MDD and the productivity of a traditional method; even though productivity with a traditional method is slightly higher.



PRODUCTIVITY

Source	Method	Type III Sum of Squares	df	Mean Square	F	Sig.
Method	Linear	,020	1	,020	,688	,424
Method * Problem	Linear	,074	1	,074	2,595	,136
Error(Method)	Linear	,314	11	,029		

Fig 6. Box-and-whisker plot for productivity. (b) Output of the GLM test for productivity

4.3 Satisfaction

Satisfaction is measured in terms of PU, PEOU and ITU on a 5-point Likert scale (the higher mark in the scale the better satisfaction). Since each metric is measured through several questions of the Likert questionnaire (8 for PU, 6 for PEOU and 2 for ITU), we have aggregated questions adding the responses per metric. Fig 7.a, Fig 8.a and Fig 8.b shows the box-and-whisker plot comparing PU, PEOU and ITU respectively. In all three plots, medians using a traditional method and using MDD match together almost exactly. If we focus on PU (Fig 7.b) and PEOU, MDD achieves slightly better results since the first and third quartiles are higher than the corresponding ones for the traditional method. For ITU, the traditional method achieves slightly better results since the first and third quartiles are higher than the ones in MDD.

We aim to verify whether or not satisfaction is independent of using MDD or traditional methods (H_{04}). Table 13 summarises the main statistical parameters (p-value, the statistical power and the effect size).

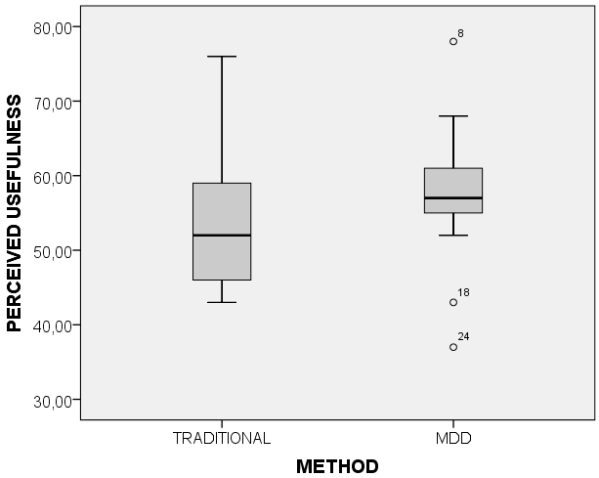
Table 13. P-values for satisfaction using GLM test

Satisfaction metric	P-value		Power		Cliff's delta		
	DM	DM*P	DM	DM*P	Trad. vs MDD	P1 vs P2 with Trad.	P1 vs P2 with MDD
PU	0.42	0.92	0.12	0.5	-0.29	-0.3	0.42
PEOU	0.63	0.41	0.07	0.11	-0.16	-0.4	-0.04
ITU	0.72	0.45	0.06	0.11	0.08	-0.52	0.14

DM means Development Method, P means Problem and "*" means interaction

Since the p-values for all three satisfaction metrics after applying the GLM test to Development Method are greater than 0.05 (Fig 7.b shows the results of the GLM analysis), we can conclude that there are no significant differences for any satisfaction metric. So there are no significant differences between MDD and traditional methods regarding satisfaction. The statistical power is low in all cases making false negatives possible. The effect size is small in all cases too. The MDD group scored better on PU and PEOU. For ITU, the traditional method performed somewhat better.

Neither are there any significant differences in the Development Method*Problem interaction; it implies that problem type does not affect satisfaction. Statistical power is large for PU and small for PEOU and ITU. In the former case, the high power denotes that the non-significant result is rather trustable as false negatives are unlikely ($\beta<0.08$). In the later cases, PEOU and ITU are likely to change if larger sample sizes are used. Effect sizes comparing Problem 1 with Problem 2 for pairs that use a traditional method is large for all three satisfaction metrics, resulting in a higher satisfaction for Problem 2. When we compare Problem 1 with Problem 2 for pairs that use MDD, we find that only PU yields a large effect size. PEOU fares better for Problem 2, whereas PU and ITU achieve better results for Problem 1. It is remarkable that subject perception does not match productivity. Even though the productivity of subjects that solve Problem 2 is higher with MDD than with a traditional method, subjects that solved Problem 2 with MDD are less satisfied than subjects that solved Problem 1.



PERCEIVED USEFULNESS

Source	Method	Type III Sum of Squares	df	Mean Square	F	Sig.
Method	Linear	89,715	1	89,715	,696	,422
Method * Problem	Linear	1,254	1	1,254	,010	,923
Error(Method)	Linear	1418,131	11	128,921		

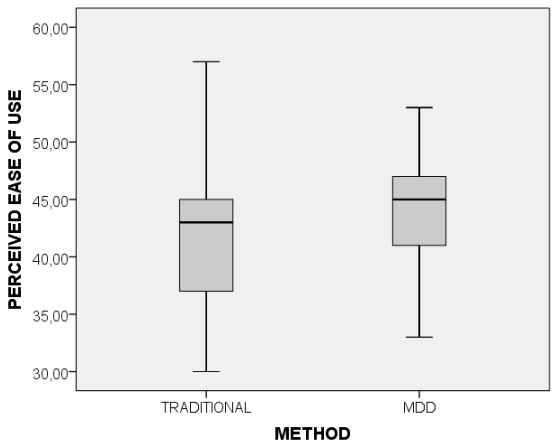
PERCEIVED EASE OF USE

Source	Method	Type III Sum of Squares	df	Mean Square	F	Sig.
Method	Linear	16,443	1	16,443	,245	,630
Method * Problem	Linear	48,443	1	48,443	,723	,413
Error(Method)	Linear	737,095	11	67,009		

INTENTION TO USE

Source	Method	Type III Sum of Squares	df	Mean Square	F	Sig.
Method	Linear	1,393	1	1,393	,140	,716
Method * Problem	Linear	6,008	1	6,008	,603	,454
Error(Method)	Linear	109,607	11	9,964		

Fig 7.(a) Box-and-whisker plot for PU. (b) Output of the GLM test for PU, PEOU and ITU



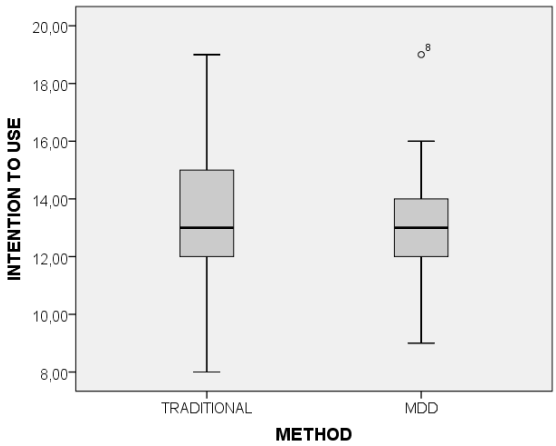


Fig 8. (a) Box-and-whisker plot for PEOU. (b) Box-and-whiskers plot for ITU

We conclude that H_{04} is accepted and there is no significant difference between developer satisfaction using MDD or a traditional method, even though PU and PEOU are moderately better for MDD.

4.4 Differences between Code-Centric and Model-Based Traditional Development Approaches

Since the control treatment is the traditional development paradigm preferred by subjects, we wondered whether **there are differences between the pairs using a code-centric paradigm and pairs using a model-based paradigm as a traditional development method**. As already mentioned, a code-centric paradigm is when developers focus only on the code, whereas, in a model-based paradigm, models are built before implementing the code.

We allowed pairs to choose their preferred paradigm since we wanted subjects to feel comfortable when carrying out the control task. Six pairs used code-centric and seven pairs used model-based approaches. Although a priori MDD is expected to be independent of both approaches, we cannot rule out that some relationships exist between the paradigm used as control by the subjects and MDD. For instance, it is conceivable that pairs applying a model-based approach might be more acquainted with modelling solutions, so they may be more effective when using MDD. Table 14 shows the accuracy of the pairs depending on the approach used as a control group and the accuracy of the same pairs using MDD.

Table 14. Accuracy by paradigm used as traditional development

		Pairs												
		P1	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**P9**	**P10**	**P11**	**P12**	**P13**
Code-Centric	Control Task	-	100	-	-	50	100	-	0	-	-	75	-	100
	Treatment Task	-	100	-	-	66.67	75	-	66.67	-	-	100	-	75
Model-Based	Control Task	100	-	100	100	-	-	50	-	100	66.67	-	75	-
	Treatment Task	100	-	100	100	-	-	100	-	100	50	-	66.67	-

For the analysis, we divided pairs into two groups depending on the approach they used for the control group: model-based and code-centric. This way we analysed using the same procedure used before (a repeated measures GLM), model-based versus MDD and code-centric versus MDD across all the response variables.

Fig 9.a shows the box-and-whisker plot that represents the accuracy of both model-based and code-centric approaches. Looking at the plot we find that the pairs that used a model-based approach in the control group achieve better accuracies in both traditional method and MDD. Pairs that use code-centric approach in the control group achieve better results in a traditional method than in MDD. Accordingly, if we compare the accuracy for the control groups using the model-based and code-centric approaches, accuracy is better for users of the model-based approach. This result ties in with the importance of models in software development.

Fig 9.b shows the box-and-whisker plot that represents the effort expended by both model-based and code-centric groups. It shows that the effort between the control group and the treatment group is almost identical for pairs working with code-centric methods in the control group. However, the effort of pairs working with the model-based approach in the control group is better (less effort is better) when they were working on the control task than on the treatment task. This indicates that subjects know how to use models in a traditional development, but this expertise could not be effectively transferred to MDD as to reach the same (low) effort level.

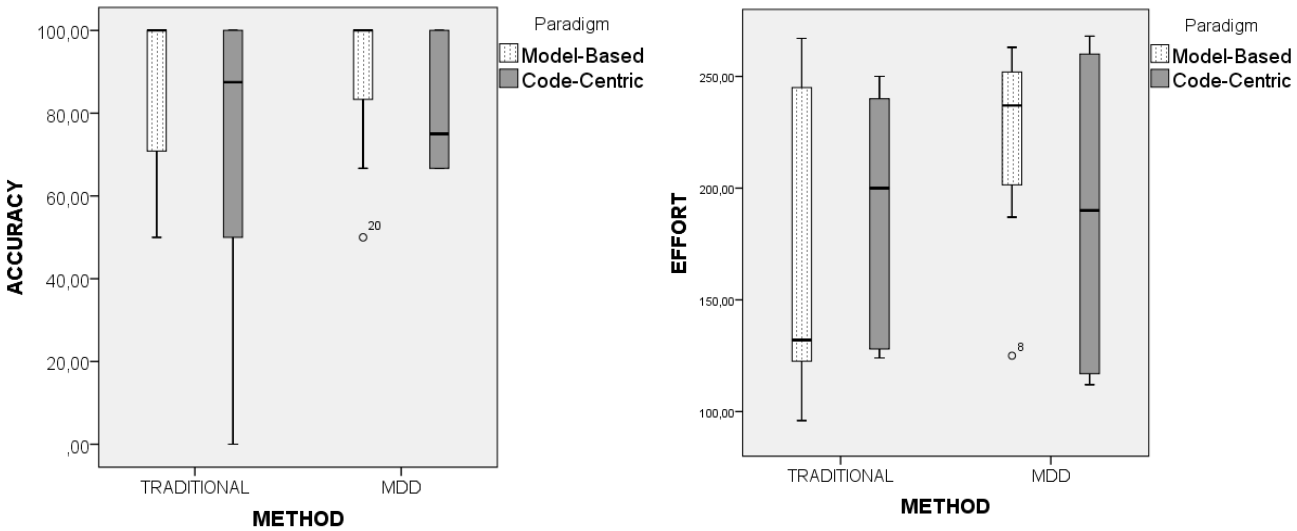


Fig 9.(a) Box-and-whisker plot for Accuracy in code-centric and model-based methods. (b) Box-and-whisker plot for Effort in code-centric and model-based methods

The box-and-whisker plot for productivity reveals that, as for effort, results are better for model-based traditional development. Plots for satisfaction do not show any difference between model-based versus MDD and code-centric versus MDD, so they are not reproduced for reasons of space.

We complemented the visual inspection with a GLM test to study whether there are significant differences between both paradigms. The results (Table 15) show that there are no differences between model-based versus MDD and code-centric versus MDD with regard to accuracy, effort, productivity, PEOU and ITU.

Table 15. P-values for the GLM test differentiating model-based and code-centric paradigms

Variable	Model-based Methods vs MDD			Code-centric Methods vs MDD		
	p-values	Power	Cliff's delta	P-values	Power	Cliff's delta
Accuracy	0.57	0.08	-0.1	0.93	0.05	0
Effort	0.28	0.16	-0.3	0.39	0.12	-0.05
Productivity	0.35	0.14	0.24	0.29	0.07	-0.05
PU	0.5	0.09	0.24	0.03*	-	-0.97
PEOU	0.52	0.09	0.16	0.35	0.13	-0.58
ITU	0.5	0.09	0.22	0.88	0.05	-0.3

* significance level = 0.05

There is only one significant result for PU for the pairs that use the code-centric approach. In this case, PU is greater for MDD than for a traditional method. Cliff's delta is close to -1, which means that the pairs that chose to directly code the

control task found MDD very useful. This perception of usefulness is not shared by the pairs that applied a model-based approach for the control task.

In brief, values for PU of pairs using code-centric in the control group are significantly better in MDD. This difference does not hold when the control group uses the model-based approach. For accuracy, effort, productivity, PEOU and ITU, there are no significant differences between model-based and MDD, or between code-centric and MDD. We have also identified differences between model-based and code-centric approaches, even though these differences are not significant. Accuracy is better in both control and treatment groups for pairs using model-based in the control group. Effort and productivity are better in the control group using the model-based approach, which again highlights the importance of models during the software development process.

Finally, we have to remark that the sample size is very small for the composition of code-centric and model-based approaches (six pairs for code-centric and seven pairs for model-based) and thus the tests have virtually no power. Statistical power for each variable is lower than 0.2, which means that results might change with larger sample sizes. More research is needed to clarify whether these results hold. Note that our experiment did not set out to study the differences between code-centric and model-based approaches, but we perform this post-hoc analysis expecting that these results are instructive.

5 Threats to Validity After Running the Experiment

This section discusses new threats that we identified after we had run the experiment and analysed the data. These threats might help to explain why no significant differences were found between traditional methods and MDD regarding effort, productivity and satisfaction. One threat is the **time taken to solve Problem 1 and Problem 2**. We limited the time per problem to a maximum of four hours (two sessions) in order to control all the experimental variables inside the classroom. Accuracy might have been better for some pairs if they had been given more time. This time limitation may conceal differences between the effort expended on MDD and traditional methods. Before the experiment, we checked that four hours was long enough to solve the small problems used. We expected participants to take different times (as long as they needed) to complete the experimental tasks. It turned out that subjects used up all the allocated time. We hypothesize that rather than participants requiring all the allotted time, students tend to use up all the time they are given in a never-ending improvement iteration [39]. This transforms the effort metric into a constant for all subjects, since most subjects have the same value for effort (four hours).

Undermotivation is another threat in some pairs. Subjects do not participate on a voluntary basis; they are students of the course of which the experiment is part. Due to undermotivation, we found that a few subjects arrived a few minutes late to the experiment sessions and had less than four whole hours to solve each problem. Undermotivation could also affect satisfaction results in such a way that the least motivated subjects would have scored satisfaction for both treatments low. In order to study this possibility, we have applied Pearson's correlation test to look for significant correlations of satisfaction between the control group and the treatment group. We got the following p-values: PU=0.415; PEOU=0.173; ITU=0.454. Since p-values are greater than 0.05, we must accept the null hypothesis, which means that there is no significant correlation between the satisfaction with a traditional development method and MDD. Even though there are no significant correlations, undermotivation in a few pairs could in any case affect their responses to satisfaction questionnaires.

Time taken to run generated code is another threat. When using INTEGRANOVA, the model must be sent over the Internet, and the generated code is received in a computer folder about five minutes later. The received code includes two C# projects, one for the client side and the other for the server side. It also includes four scripts for building the database (creating tables, primary keys, foreign keys and indexes). Once the code has been received, the pairs must open and compile both C# projects and run the scripts to create the database. The entire process (from sending the model until the application is running) takes about 15 minutes. All these tasks must be repeated every time the pairs generate a new version of the code. Note that none of these tasks are related to MDD and they all depend exclusively on the MDD tool used in the experiment. Other tools might use a different system to generate and setup the system. In our experimental investigation we have timed the setup process and the modelling activities. This means that MDD includes an extra effort that is not required in a traditional method, where applications are deployed and run in a few seconds. Note that even though the subjects using MDD had to expend extra effort to generate the code and setup the system, accuracy, effort and productivity are similar for both MDD and a traditional development method.

All MDD tools require a number of tasks to generate the code and setup the system. This experiment has allowed us to pinpoint these additional tasks that are often overlooked when MDD is theoretically compared with a traditional development method. However, since these tasks are essential for producing fully functional software, they must be taken into account in any discussion on the improvements in productivity supposedly offered by MDD.

Since there are subjects with more experience in MDD than others, we analysed **subject background**. According to the demographic questionnaire, three subjects had a sound knowledge of MDD before attending the course (Fig 10.a). If we compare these data with previous knowledge of any programming language (Fig 10.b), MDD was at a disadvantage before starting the course. This implies that most subjects have at most 30 hours' experience of MDD (number of hours spent in the classroom as part of the course), whereas they all had previous knowledge of several programming languages. Note that the programming language used by each pair (Table 10) was one of the languages they have good knowledge of, but there

are pairs with knowledge in more than one programming language. Fig 11 shows the number of subjects that have worked at real companies and their role. Again, these data show that subjects have a sound background of programming languages, since 15 subjects have been working as junior programmers for some time. Therefore, subject background benefits the control treatment over MDD.

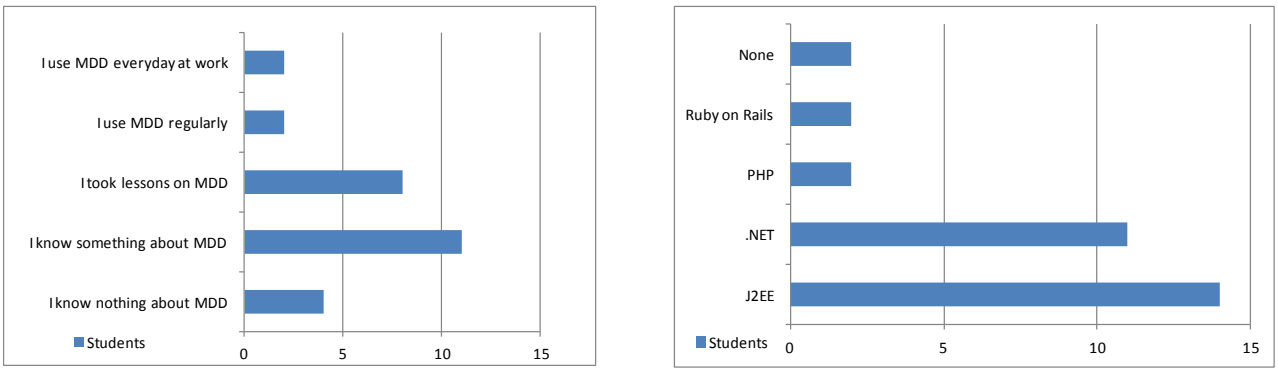


Fig 10. (a) Subjects' previous experience with the MDD paradigm. (b) Students' previous experience with programming languages

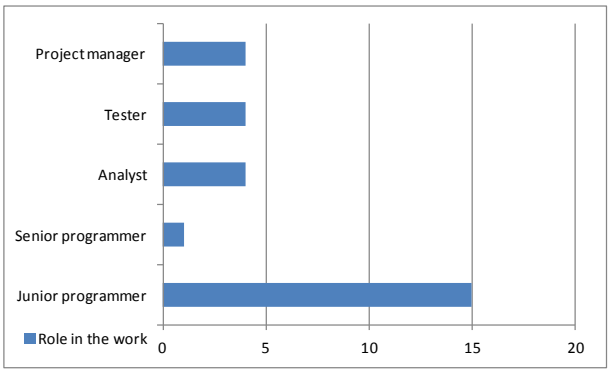


Fig 11. Students' previous roles in companies

6 Discussion

This section reviews all the results and threats of the experiment in order to draw some conclusions regarding the comparison of the MDD paradigm and traditional development methods. For each response variable, we discuss the following information: interpretation of results, some recommendations regarding when to apply MDD or a traditional method, and whether results agree with previous claims on MDD.

The results related to quality (operationalised as ***accuracy****) show that accuracy* using MDD is less sensitive to problem complexity than a traditional method. Problem 1 has 40 function points and Problem 2 has 35 function points. Even though the size of problems was similar, Problem 2 was slightly more complex than Problem 1 because of an inheritance between two classes. For small variations in the complexity of problems, mean accuracy remains stable at around 80%. However, when we increase problem complexity to a small degree, the accuracy of a traditional method decreases from 90% to 60%. Therefore, differences between MDD and a traditional development method might be considerable for larger and more complex problems. Note that as outlined in Section 5, most subjects had quite a lot of experience with traditional development and hardly any experience with MDD. Even though most subjects' experience with MDD is limited to a training of 18 hours, MDD achieves higher quality when problem complexity increases, which is a promising result for MDD.

The recommendation derived from the results of accuracy is that MDD could be better suited than a traditional method for solving highly complex problems.

Findings of the experiment regarding accuracy tie in with previous research claiming that MDD helps to improve code quality [44] [10] [34]. This experiment contributes by better specifying this claim since when problem complexity increases slightly, MDD is significantly more stable and obtains better results than for a traditional method. As results show, MDD and traditional methods do not differ as regards to accuracy when problem complexity is low.

The results related to ***effort*** *show that* the effort to develop a system with MDD is not significantly different from the effort with a traditional method. If we consider that most subjects did not have any other experience in MDD than the training received during the experiment, whereas they have a wide experience in a traditional method, the absence of significant differences between both treatments is a positive result for MDD. This leads to think that when the background experience of both treatments is the same, significant differences might appear.

A recommendation derived from the results of effort is that a traditional method is more suitable for developments where the team is not very experienced in applying MDD. Even though there are no significant differences in the results of the experiment, MDD takes slightly longer to apply and MDD requires a longer training time (in our case, 20 hours of lessons).

Another recommendation is that a traditional development is more suitable whenever analysts need to run the system frequently during the development process, for instance, to carry out testing activities. Note that code generated with the MDD tool needs to be deployed in the execution environment. This action takes several minutes (around 15 minutes for INTEGRANOVA), because several set-up tasks need to be performed, i.e., compile the models, download the source code, compile the source code, create the database, create an ODBC connection, create a COM+ component, and register the web in the web application server. The installation process needs to be repeated every time subjects run the code. However, a traditional development method does not require to repeat these set-up actions every time and subjects can run the code immediately after making changes on it. In any case, this situation may change when MDD technologies evolve.

Findings of the experiment regarding effort contradict previous research claims stating that MDD reduces developer effort [44] [4] [10] [19] [43] [42]. As commented in the threats to validity after running the experiment, we noticed that subjects working with both paradigms used up all the available time (four hours). This renders the effort study inconclusive, since time taken with both paradigms is the same. This is consistent with previous studies [39] showing that there is a tendency for work to expand to fill the available time. Therefore, we do not have enough evidence to confirm or refute the results of previous works that claim that MDD reduces developer effort.

*Results of **productivity** show that* using MDD is the same as productivity for a traditional method, independently of problem complexity. Since productivity depends on time, this variable is affected by the same threats as effort. Subjects have more experience with a traditional method than with MDD.

Recommendations derived from the results of productivity are the same as for effort. MDD is not recommended when there is not much time to train developers for MDD or when the system to develop needs to be tested frequently on a short time frame.

Findings of the experiment regarding productivity are not useful to confirm or refute previous claims ensuring that MDD improves productivity [42] [11] [34] since subjects filled all the available time for both paradigms.

*Results of developer's **satisfaction** show that* satisfaction when applying MDD is not significantly different from satisfaction when applying a traditional method. If we divide satisfaction into PU, PEOU and ITU, we find some differences between MDD and traditional methods, even though these differences are not significant. PU and PEOU are somewhat better for MDD, whereas ITU is slightly better for a traditional method. This means that subjects identify the MDD paradigm as being more useful and easier to use but they prefer a traditional development for future projects. The benefits of MDD regarding usefulness and ease of use make sense since the experiment was run in the context of a course. Subjects perceive the usefulness of MDD when they automatically transform any model into code. Moreover, subjects had access to manuals and teachers during the course, which makes MDD easier to use. Low intention to use MDD in future developments might be the result of inertia. Most subjects have developed several software systems with a traditional method but only two systems with MDD (during the training and the experiment).

The recommendation derived from the results of satisfaction is offering some reward to foster the adoption of MDD. Developers have the certain resistance to changes and feel more comfortable with the paradigms they master. The subjects in our experiment considered the MDD method as useful and easy to use but they still do not have much intention to use it. Therefore, if developers are already experienced with traditional development, then the adoption of MDD needs to be rewarded. The type of reward could be the topic of future empirical studies. In any case, if computer science curricula evolve and MDD takes higher relevance, the perception of forthcoming software engineers may be different.

Findings of the experiment regarding satisfaction match previous research claims that MDD enhances developer satisfaction. Note that subjects were recruited among students of a course (i.e they were not volunteers) and might not be very motivated, as discussed under the threats. This undermotivation might be what stops there being significant differences between satisfaction in MDD and traditional methods.

Comparing the results of **pairs that worked with model-based and code-centric methods**, we get interesting findings. First, PU of pairs that worked with code-centric methods is significantly higher for MDD than for pairs that worked with a traditional development method. This means that subjects who are not used to working with models regard the MDD paradigm, whose key artefact are models, as being useful. This ties in with previous research claiming that MDD enhances developer satisfaction [30]. Second, we have identified several non-significant differences between model-based and code-centric methods. Accuracy is better for both a traditional development method and MDD used by pairs that worked with model-based methods. Effort and productivity are better for pairs that worked with model-based traditional development methods, but they remain constant for pairs that worked with code-centric methods. Regarding satisfaction, there are no observable differences between model-based and code-centric methods.

We can conclude that some of the benefits of MDD claimed in the literature have not been observed in the context of our empirical evaluation. According to our results, differences between paradigms might be more evident when problem complexity increases. The limitation of the experiment to four hours and the use of two problems with low functional sizes might reduce the differences between paradigms. Our results are based on the use of one MDD tool, INTEGRANOVA. This tool is fully compliant with the MDD paradigm, where analysts focus on building conceptual models from which code is automatically generated. However, we cannot generalise these results to any other MDD tool, since each tool has specific

particularities. For example, INTEGRANOVA takes around 15 minutes to generate the code and run the system. This time may differ for other tools. Moreover, there are MDD tools where transformation rules do not generate fully functional systems and some chunks of code must be manually implemented.

We also have to consider the context where the experiment was run, since these results cannot be generalised for any developer and any type of system. Subjects are students whose main goal is to pass the course. Other subjects in an industrial setting with other goals might get different results. Also, our subjects were already acquainted with traditional development, whereas most of them had to learn MDD. If subjects were experienced MDD practitioners the results might be different. Therefore, after running the experiment, we can state that our results are only valid for software built by novice developers with a good grounding in traditional developments but newcomers to MDD.

Calculations using G*Power [16] show that the sample size used in the experiment can only identify effect sizes of 0.4 with an actual power of 60%. *The non-significant effects might be due to the effect size between MDD and traditional development methods is medium at most*. In order to detect these low to medium effect sizes, we need to conduct experiments on larger sample sizes or replicate the experiment multiple times and synthesize the individual results.

7 Conclusions

This paper compares traditional software development methods with MDD paradigm in terms of quality (focused on accuracy), effort, productivity and satisfaction through an experimental investigation. Previous experiments have been conducted in the field of MDD, but only a few have developed a fully functional system from scratch comparing MDD and a traditional method. Most research deals with the implementation of small chunks of code. Moreover, existing research focuses on estimating developer effort, overlooking quality, productivity and satisfaction. Our experiment fills a gap in the empirical evidence by checking the benefits of MDD for developing a fully functional system from scratch.

The results of our study conclude that when problem complexity increases slightly, the accuracy remains stable only with MDD, obtaining better results than with a traditional method. This result means that MDD accuracy is more robust to variations in the complexity of the problems to be solved. There are no significant differences for effort, productivity and satisfaction. This statement contradicts much of the existing literature, which claims that MDD should always outperform traditional methods on these variables. Results also show that subjects who usually work with a code-centric paradigm have a high degree of perceived usefulness for MDD.

Note importantly that these results must be contextualised in the type of experiment that we have conducted. First, sample size is rather small. We have worked with 13 pairs (26 developers). Second, subjects were master's students and did not have much experience of developing real web applications in industry. Third, in order to control extraneous variables, we had to use simple problems that could be solved in the classroom within four hours. Fourth, the MDD tool that we have used in the experiment is 100% MDD compliant. This means that pairs using the MDD paradigm did not write a single line of code. We cannot guarantee that another experiment would yield the same results if any of these four characteristics were changed.

In the future we aim to replicate this experiment changing some elements of the design. First, we are planning to use more complex problems to be solved by the pairs. This will clarify whether or not the complexity of the problems is currently affecting our conclusions. Second, we will use another MDD tool different from INTEGRANOVA to verify whether or not our results are independent of the tool used.

Acknowledgments

This work was developed with the support of the Spanish Ministry of Science and Innovation project SMART ADAPT (TIN2013-42981-P), TIN2011-23216 and was co-financed by ERDF. It also has the support of Generalitat Valenciana-funded IDEO project (PROMETEOII/2014/039) and UV (UV-INV-PRECOMP13-115032).

References

[1] *Genexus: http://www.genexus.com*.

[2] "INTEGRANOVA Technologies: http://www.integranova.com," ed.

[3] "ISO/IEC 9126-1, Software engineering - Product quality - 1: Quality model," ed, 2001.

[4] "The Middleware Company. Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach. http://www.omg.org/mda/mda_files/MDA_Comparison-TMC_final.pdf," 2003.

[5] *UML: http://www.uml.org/*.

[6] R. Acerbis, A. Bongio, M. Brambilla, and S. Butti, "WebRatio 5: An Eclipse-Based CASE Tool for Engineering Web Applications," *Lecture Notes in Computer Science,* vol. 4607, pp. 501-505, 2007.

[7] B. Anda and K. Hansen, "A case study on the application of UML in legacy development," presented at the Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, Rio de Janeiro, Brazil, pp. 124-133, 2006.

[8] P. Baker, S. Loh, and F. Weil, "Model-driven engineering in a large industrial context - motorola case study," in *8th International Conference of Model Driven Engineering Languages and Systems (MoDELS)*, Montego Bay, Jamaica, pp. 476–491. 2005

[9] V. Basili and H. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering,* vol. 14, pp. 758-773, 1988.

[10] Borland, "Keeping your business relevant with model driven architecture (MODEL-DRIVEN ARCHITECTURE)," 2004.

[11] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice (Synthesis Lectures on Software Engineering)*, first ed.: Morgan & Claypool Publishers, 2012.

[12] C. Bunse, H.-G. Gross, and C. Peper, "Embedded System Construction --- Evaluation of Model-Driven and Component-Based Development Approaches," in *Models in Software Engineering*, R. C. Michel, Ed., ed: Springer-Verlag, pp. 66-77, 2009.

[13] D. T. Campbell and J. C. Stanley, *Experimental and Quasi-experimental Designs for Research*. Boston, USA: Houghton Mifflin, 1963.

[14] T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg, "A systematic review of statistical power in software engineering experiments," *Information and Software Technology,* vol. 48, pp. 745-755, 2006.

[15] D. W. Embley, S. Liddle, and Ó. Pastor, "Conceptual-Model Programming: A Manifesto," in *Handbook of Conceptual Modeling*, ed: Springer, pp. 3-16, 2011.

[16] F. Faul, E. Erdfelder, A.-G. Lang, and A. Buchner, "G*Power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences," *Behavior Research Methods,* vol. 39, pp. 175-191, 2007/05/01 2007.

[17] G. Giachetti, F. Valverde, and B. Marin, "Interoperability for model-driven development: Current state and future challenges," in *Proc of Sixth International Conference on Research Challenges in Information Science (RCIS)*, pp. 1-10. 2012

[18] R. J. Grissom and J. J. Kim, *Effect Sizes For Research: A Broad Practical Approach*: Taylor & Francis Group, 2005.

[19] B. Hailpern, Tarr, P., "Model-Driven Development: the Good, the Bad, and the Ugly," *IBM Syst. J.,* vol. 45, pp. 451-461, 2006.

[20] W. Heijstek and M. R. V. Chaudron, "Empirical Investigations of Model Size, Complexity and Effort in a Large Scale, Distributed Model Driven Development Process," in *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*, pp. 113-120. 2009

[21] J. Hutchinson, M. Rouncefield, and J. Whittle, "Model-driven engineering practices in industry," presented at the Proc. of the 33rd International Conference on Software Engineering (ICSE), Waikiki, Honolulu, HI, USA, pp. 633-642, 2011.

[22] IEEE, "Systems and software engineering -- Vocabulary," *ISO/IEC/IEEE 24765:2010(E),* pp. 1-418, 2010.

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

[24] M. Jose Escalona and G. Aragon, "NDT. A Model-Driven Approach for Web Requirements," *IEEE Transactions on Software Engineering,* vol. 34, pp. 377-390, 2008.

[25] N. Juristo and A. Moreno, *Basics of Software Engineering Experimentation*: Springer, 2001.

[26] T. Kapteijns, S. Jansen, S. Brinkkemper, H. Houët, and R. Barendse, "A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare," presented at the Proc. of 4th European Workshop on From code centric to model centric software engineering: Practices, Implications and ROI (C2M), Enschede, The Netherlands, pp. 22-33, 2009.

[27] G. Karsai and J. Sztipanovits, "Model-Integrated Development of Cyber-Physical Systems," in *Software Technologies for Embedded and Ubiquitous Systems*. vol. 5287, U. Brinkschulte*, et al.*, Eds., ed: Springer Berlin Heidelberg, pp. 46-54, 2008.

[28] K. Krogmann and S. Becker, "A case study on model-driven and conventional software development: The palladio editor," in *Software Engineering 2007 - Beiträge zu den workshops*, pp. 169–176. 2007

[29] O. I. Lindland, G. Sindre, and A. Sølvberg, "Understanding Quality in Conceptual Modeling," *IEEE Software,* 1994.

[30] Y. Martínez, C. Cachero, and S. Meliá, "MDD vs. traditional software development: A practitioner's subjective perspective," *Information and Software Technology,* vol. 55, pp. 189-200, 2013.

[31] A. McNeile. (2003, *MDA: The Vision with the Hole?* http://www.metamaxim.com/download/documents/MDAv1.pdf.

[32] N. Mellegård and M. Staron, "Distribution of Effort among Software Development Artefacts: An Initial Case Study.," in *Proc. of 11th International Workshop BPMDS 2010, held at CAiSE 2010*, Hammamet, Tunisia, pp. 234-246. 2010

[33] S. J. Mellor, A. N. Clark, and T. Futagami. Guest Editors' Introduction: Model-Driven Development. *IEEE Software*. 14-18, 2003.

[34] T. O. Meservy, "Transforming software development : an MDA road map " *IEEE Computer,* vol. 38, 2005.

[35] L. S. Meyers, *Applied multivariate research : design and interpretation / Lawrence S. Meyers, Glenn Gamst, A.J. Guarino*. Thousand Oaks: SAGE Publications, 2006.

[36] D. L. Moody, "The method evaluation model: a theoretical model for validating information systems design methods," presented at the European Conference on Information Systems (ECIS 03), Naples, Italy pp. 1327-1336, 2003.

[37] A. Olive, "Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research," in *Proceedings of the 16th Conference on Advanced Information Systems Engineering, Oscar Pastor, João Falcão e Cunha (Ed.), Lecture Notes in Computer Science, Springer-Verlag, Porto, Portugal, Lecture Notes in Computer Science, Vol. 3520, ISBN 3-540-26095-1.pp. 1-15.*, pp. 1-15. 2005

[38] P. Papotti, A. Prado, W. Souza, C. Cirilo, and L. Pires, "A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation," in *Advanced Information Systems Engineering*. vol. 7908, C. Salinesi*, et al.*, Eds., ed: Springer Berlin Heidelberg, pp. 321-337, 2013.

[39] N. Parkinson, *Parkinson's Law and Other Studies in Administration*. Boston: Houghton Mifflin Cornpony, 1957.

[40] O. Pastor, Molina, J., *Model-Driven Architecture in Practice*. Valencia: Springer, 2007.

[41] D. Schwabe, R. d. A. Pontes, and I. Moura, "OOHDM-Web: an environment for implementation of hypermedia applications in the WWW," *SIGWEB Newsl.,* vol. 8, pp. 18-34, 1999.

[42] B. Selic, "The Pragmatics of Model-Driven Development," *IEEE software,* vol. 20, pp. 19-25, 2003.

[43] S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," *IEEE Software,* vol. 20, pp. 42-45, 2003.

[44] Y. Singh and M. Sood, "Model Driven Architecture: A Perspective," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pp. 1644-1652. 2009

[45] I. C. Society, "830-1998 - IEEE Recommended Practice for Software Requirements Specifications ", ed, 1998.

[46] M. Staron, "Transitioning from Code-Centric to Model-Driven Industrial Projects: Empirical Studies in Industry and Academia," in *Model-Driven Software Development: Integrating Quality Assurance*, ed: IGI Global, pp. 236-262, 2009.

[47] T. Strasser, C. Sunder, and A. Valentini, "Model-driven embedded systems design environment for the industrial automation sector," in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pp. 1120-1125. 2008

[48] B. Vanderose and N. Habra, "Towards a Generic Framework for Empirical Studies of Model-Driven Engineering," in *Proc. of workshop of Empirical Studies of Model-Driven Engineering (ESMDE) at International Conference on Model Driven Engineering Languages and Systems (MoDELS'08)* Toulouse (France), pp. 71-80. 2008

[49] C. Wohlin, P. Runeson, and M. Höst, *Experimentation in Software Engineering: An Introduction*. Boston: Kluwer Academic Publishers, 2000.

[50] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*: Springer, 2012.

APPENDIX A

We describe the conceptual models needed to work with INTEGRANOVA (the tool used in the experiment):

- An Object Model that specifies the system structure in terms of classes of objects and their relationships. It is modelled as an extended UML [5] class diagram.
- A Dynamic Model that represents the valid life-cycle of events for an object, as a set of state-transition diagrams.
- A Functional Model that specifies how events change object states, by means of rules associated to class methods.
- A Presentation Model that represents the interaction between the system and the user, by means of an abstract interface model.

INTEGRANOVA is capable of automatically creating a Presentation Model, so this model does not need to be created in order to develop a fully-functional software. Although interfaces generated automatically have lower quality than interfaces modelled with the Presentation Model, they allow running the software. Since interface quality is outside the scope of our experiment and the lessons where the experiment was performed did not address this model, we decided to use the automatically generated Presentation Model, in order to reduce development time. Therefore, the analysis of the MDD tool is focused on the Object, Dynamic and Functional Models.

APPENDIX B

Test Cases for Problem 1: An electrical appliance repair company management system

Test 1: Create customer

-Insert passport no.: 33472035L
-Insert name: Ignacio
-Insert surname: Panach Navarrete
-Insert address: Colón 70
-Insert city: Alboraya
-Insert postal code: 46120
-Insert telephone no.: 96 123 12 12
-Check that all above data have been saved in the system

Test 2: Create repair card

-Insert identifier: 1
-Insert customer passport no.: 33472035L
-Insert repair date: 21/02/2013
-Insert description: TV repair
-Insert amount due: 100€
-Insert technician name: Paco Valverde

- Check that all above data have been saved in the system

Test 3: Create invoice

(create another customer)

-Insert passport no.: 44444488L
-Insert name: Paco
-Insert surname: García López
-Insert address: Ausias March, 32
-Insert city: Valencia
-Insert postal code: 46520
-Insert telephone no.: 96 123 22 33
- Check that all above data have been saved in the system

(create another repair card)

-Insert identifier: 2
-Insert customer passport no.: 33472035L
-Insert repair date: 22/02/2013
-Insert description: antenna repair
-Insert amount due: 200€
-Insert technician name: Vicente Pelechano
- Check that all above data have been saved in the system

(create another repair card)

-Insert identifier: 3
-Insert customer passport no.: 44444488L
-Insert repair date: 23/02/2013
-Insert description: fridge repair
-Insert amount due: 50€
-Insert technician name: Vicente Pelechano
- Check that all above data have been saved in the system

(create the invoice)

-Insert identifier: 1
-Insert creation date: 01/03/2013
-Insert customers passport no.: 33472035L
- Check that invoice includes the repair cards with identifier 1 and 2 only
-Check that total amount invoiced is 300€

APPENDIX C

Test Cases for Problem 2: A photographer agency management system

Test 1: Create application

-Insert application identifier: 1
-Insert application date: 03/01/2013
-Insert description of the photography equipment: Canon camera with 10 Mx, tripod
-Insert brief bio: 3 years working for a newspaper and 2 years working for a tabloid
-Insert applicant passport no.: 12345678A
-Insert applicant name: Sergio
-Insert applicant surname: España Cubillo
-Insert applicant address: Pio XII, 3, 4
-Insert applicant city: Valencia
-Insert applicant postal code: 46230
-Insert applicant telephone no.: 96 123 45 67
-Check that all above data have been saved in the system

Test 2: Create new application for same photographer

-Insert application identifier: 1
-Insert application date: 31/01/2013
-Insert description of the photography equipment: Canon camera with 10 Mx, tripod
-Insert brief bio: 3 years working for a newspaper and 2 years working for a tabloid
-Insert applicant passport: 12345678A (check that the photographer's other personal details do not have to be re-entered)
-Check that the system does not allow more than one application to be created in the same month

Test 3: Approve application

-Insert applicant passport no.: 12345678A
-Insert level: 2
-Check that the applicant appears in the list of accepted photographers
Test 4: Promote photographer
-Insert photographer passport no.: 12345678A
-Insert level: 1
-Check that the system does not allow the level to be decreased
-Insert level: 3:
-Check that this last promotion has been saved in the system

APPENDIX D

This appendix shows the class diagrams of Problem 0 and Problem 00 used as training for a traditional method and MDD respectively.

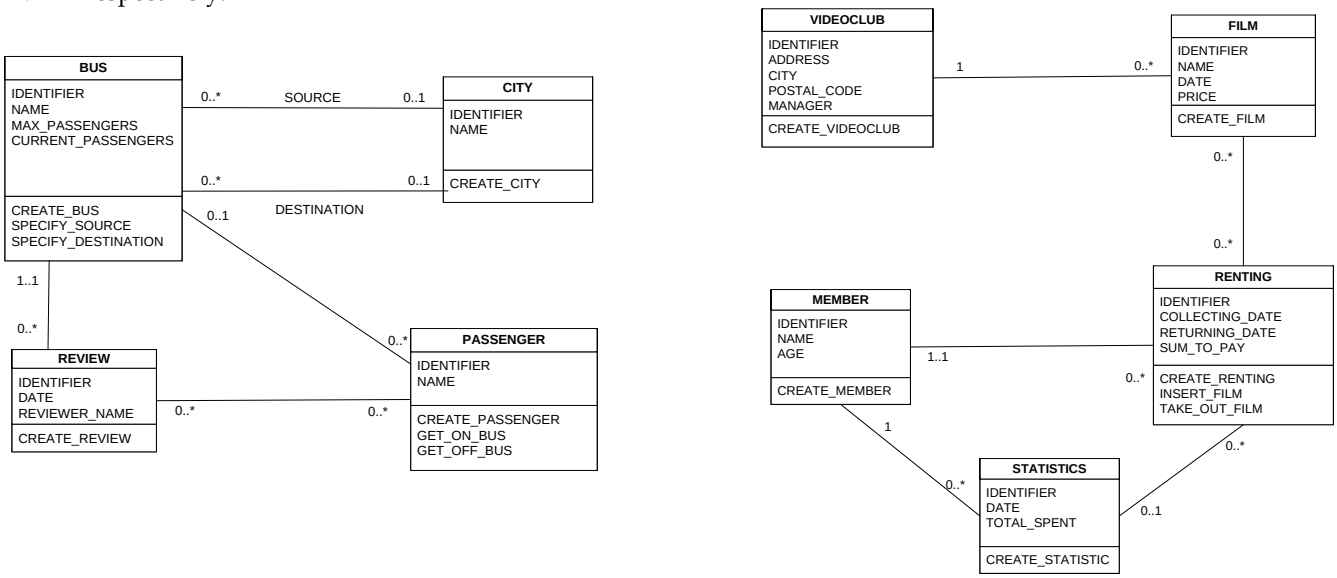


Fig 12. (a) Class Diagram of Problem 0. (b) Class Diagram of Problem 00

APPENDIX E

Pair	Type	Accuracy Trad. all nothing	Accuracy MDD all nothing	Accuracy Trad. Weighted items	Accuracy MDD Weighted items	Accuracy Trad. Same weight	Accuracy MDD Same weight	Time Trad.	Time MDD
#1	1	100	100	100	100	100	100	258	252
#2	2	100	100	100	100	100	100	250	268
#3	1	100	100	100	100	100	100	123	216
#4	2	100	100	100	100	100	100	132	125
#5	2	50	66,67	68,75	91,67	70,75	94,33	128	117
#6	1	100	75	100	96,25	100	95,75	124	112
#7	2	50	100	88,25	100	89,75	100	267	187
#8	2	0	66,67	75,75	95,33	77	95,33	232	260
#9	1	100	100	100	100	100	100	122	237
#10	1	66,67	50	83,33	77,5	89	83,5	232	263
#11	2	75	100	96,25	100	95,75	100	240	254
#12	2	75	66,67	100	95,33	100	95,33	96	252
#13	1	100	75	100	98.25	100	98	168	126

Pair	Time Trad.	Time MDD	Satisfaction Trad. PEOU	Satisfaction MDD PEOU	Satisfaction Trad. PU	Satisfaction MDD PU	Satisfaction Trad. ITU	Satisfaction MDD ITU
#1	258	252	34	46	46	57	8	15
#2	250	268	36	47	51	63	13	14
#3	123	216	37	48	52	57	12	12
#4	132	125	49	53	61	78	15	19
#5	128	117	30	50	43	68	12	14
#6	124	112	37	45	49	55	12	13
#7	267	187	57	42	76	52	19	12
#8	232	260	43	41	46	61	13	14
#9	122	237	42	33	55	43	13	11
#10	232	263	44	47	59	60	16	16
#11	240	254	45	38	55	58	14	13
#12	96	252	52	33	62	37	19	9
#13	168	126	43	44	43	57	13	10