Software Quality Journal manuscript No.
(will be inserted by the editor)

Comparison of a textual vs a graphical notation for the maintainability of MDE domain models: an empirical study

S. Meliá · C. Cachero · J. Hermida · E. Aparicio

Received: date / Accepted: date

Abstract *Background:* Models are a useful tool to increase the developer's productivity and satisfaction when performing maintenance tasks. However, in order to maximize these advantages, the right selection of notations must be made. Unfortunately, the software engineering field lacks a body of empirical evidence that supports such selection. A suboptimal decision in this regard may have negative consequences over the maintenance process.
Goal: To compare a textual and a graphical notation with respect to the efficiency, effectiveness and satisfaction of software developers while performing analysability and modifiability tasks on two different applications.
Method: We have carried out a quasi-experiment with 86 3rd-year students of the Computer Engineering degree at the University of Alicante. Subjects were randomly classified in two groups, and each group performed 20 maintenance tasks with a textual and a graphical notation. We measured and compared the efficiency, effectiveness and satisfaction of subjects assigned to each treatment.
Conclusions: The analysed data show that the coverage of analysability and the efficiency of modifiability are affected by the type of notation used, regardless of the application. In both cases, subjects using the textual notation

S. Meliá
DLSI. Universidad de Alicante, Spain
E-mail: santi@dlsi.ua.es

C. Cachero
DLSI. Universidad de Alicante, Spain
E-mail: ccachero@dlsi.ua.es

J. Hermida
European Commission Joint Research Centre, Italy
E-mail: jesus.hermida@jrc.ec.europa.eu

E. Aparicio
Universidad de Alicante, Spain
E-mail: eac9@alu.ua.es

performed significantly better. This study needs to be replicated to extend our conclusions to other subject profiles and application types.

Keywords MDE, Maintainability, Analysability, Modifiability, Quasi-experiment, Notations

1 Introduction

In Software Engineering, models are widely used to improve the quality of software development processes. Among their benefits, models help developers (1) to create and communicate software designs in early stages; (2) to trace the design back to the requirements, thus helping to assess the degree of fulfilment; and (3) to apply iterative development processes, where models facilitate the application of quick and frequent modifications [39]. In Model Driven Engineering (MDE) approaches, such models are core development artefacts, that need to be not only created but also maintained during the whole software product lifecycle.

In previous works, it has been empirically proven how the use of models in the context of a model-driven approach has a positive impact on efficiency and effectiveness of maintenance tasks [22], and also on developer's satisfaction [23]. In these experiments, results of performing the maintainability tasks using models were compared with the results of performing the same tasks directly on code. However, the degree of achievement of these benefits can be greatly impacted by some hidden factors that were pointed at as threats to validity in those studies, among which the quality and suitability of the chosen modeling language for the tasks at hand, the tools, the modeling processes, the knowledge and experience of modelers, and the quality assurance techniques applied over the models stand out [26]. This means that, in order to obtain the expected benefits by modelling, project managers need to choose the appropriate modelling language, editor and process, according to their goals and the empirical data available. Unfortunately, such empirical data is still scarce in the discipline.

In this paper we present an empirical study that isolates one of these variables, the quality of the modelling languages, while keeping the others (editor and process) constant. Such quality has two subcomponents: (1) the quality of the underlying (conceptual) basis of the language (i.e., the abstract syntax, usually represented in a meta-model) and (2) the quality of the external (visual) representation of the language (i.e., the concrete syntax or notation) [18, 34]. In this experiment we have centred on the representation of the language, while maintaining the abstract syntax also constant.

In particular, we have measured the impact and effect size of two different notations on the results of cognitive activities involved in MDE maintainability, namely model understanding and error detection and correction. Our departing hypothesis is, based on existing literature [2, 28], that the notation used impacts such results.

Notations can be broadly divided into two categories: graphical or textual. According to [28], graphical languages are fundamentally different from textual languages in two main aspects: how they encode information and how they are processed by the human mind.

However, the empirical evidence regarding to what extent such differences affect the developer's task outcomes is still insufficient [42,13].

This paper aims at increasing the amount of such empirical data by comparing, by means of a quasi-experiment, the efficiency, effectiveness and satisfaction of a group of novice software developers while performing maintenance tasks -regarded by many as the most expensive phase of the software development lifecycle [3]- over an object-oriented domain model using two concrete syntaxes, one textual and one graphical.

The contents of the paper are organised as follows: Section 2 introduces the main concepts used in the definition of the study. Section 3 describes our quasi-experiment design, including the context of the study (Section 3.1), the hypothesis and measures (Section 3.2) and the data gathering procedure (Section 3.3). The analysis of the data is outlined in Section 4, together with the threats to the validity of this study. Then, a discussion of the results and their relationship with previous research is presented in Section 5. Finally, Section 6, draws the main conclusions of the paper and outlines the next steps of our research.

2 Background and Definitions

To the best of our knowledge, there are no studies that, so far, have measured the maintainability of software models in MDE environments from a notational point of view, which is the aim of our study. Examples of related but different research lines include model measures validation (see *e.g.* [21]) or influence of the use of aspects such as patterns [38] or documentation [19] on model maintenance. Also, in the last years we have witnesses an increasing number of papers that tackle the impact of using models *vs.* not using them on aspects such as productivity and maintainaibility of the applications (see *e.g.* [37,6, 22]).

However, several authors have discussed the theoretical advantages/ disadvantages of using textual and graphical notations. Such research papers constitute the basis for the definition of our experimental hypotheses, and they are presented next.

2.1 Impact of notation on the maintainability of models

Given the variety of proposals supporting both textual and graphical notations, several authors have discussed the manner in which the choice of a graphical or a textual modelling language affects different quality features of the software development process. Jouault et al. [15] state that the implementation of the textual notation keeps the information of line and column in

Table 1 Main benefits of each notation type.

Textual Notation	**Graphical Notation**
(+) Analizability	(+) Representation of thoughts
(+) Consistency checking	(+) Accessible
(+) Platform and tool independent	(+) Spatial reasoning
(+) Shorter learning curve	(+) Ease programming task

the model, which may improve the analysability of the textual model. In the same line, Petre [36] points at the fact that, since using textual modelling languages is similar to programming, the learning curve is lower than for graphical representations, which may increase the efficiency of these languages with respect to their graphical counterparts. Jackson *et al.* [14] consider that the use of a textual notation leads to a more efficient process of consistency checking. Grönniger *et al.* [12] also indicate that the use of this notation type contributes to improving the analysability (due to spatial efficiency) and the modifiability (due to their platform- and tool-independence, which makes possible the use of version control systems) of the model.

Regarding the advantages of a graphical notation over textual ones, Petre [36] argues that graphical notations may ease the comprehensibility of models, since they provide a more direct mapping between internal and external representations of the domain. In the same line of thought, Kosslyn [17] also suggests that if relations among objects are visually or spatially grasped, it may be easier to derive a mental model of a system structure, and in this sense graphical representations would outperform textual ones. More focused on software development, Spohrer and Soloway [40] report that the use of graphical notations can help to avoid language constructs misconceptions, since they give more scope for visual or spatial reasoning than the text-based languages. Last but not least, Myers [30] comments on the superior attractiveness of graphical representations over textual ones, and offers a reason for it: 'graphics tend to be a higher level description of the desired actions (often deemphasizing issues of syntax and providing a higher level of abstraction) and may therefore make the programming task easier even for professional programmers'.

In a position in-between we can cite Agnyal [1], who claims that using two synchronised notations (graphical-textual) for the domain model improves its maintainability.

Table 1 summarises the main arguments for and against each notation type.

Next, we introduce the definitions of the main concepts that contextualize our experiment.

2.2 Software Maintainability

The ISO/IEC 25010:2011 standard [9] defines maintainability as *the degree of efficiency and effectiveness to which a product or system can be modified by the developers.*

According to this ISO, the maintainability of the application can be decomposed into five sub-characteristics [9]: (a) *modularity*, *i.e.*, the degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components; (b) *reusability*, *i.e.*, the degree to which an asset can be used in more than one software system, or in building other assets; (c) *analysability*, *i.e.*, the degree to which the software product can be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified; (d) *modifiability*, *i.e.*, the degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading its quality; and (e) *testability*, *i.e.*, the degree of efficiency and effectiveness with which test criteria can be established for a system.

2.3 The Domain Model

In object-oriented design (OOD) the domain model is a central element to represent the business logic and the persistence layers of software applications [7]. Among the benefits of domain models, some stand out, namely: (a) they define and provide a common vocabulary for the developers; (a) they facilitate the detection of inconsistencies; and (c) they facilitate the definition of constraints in early stages of design. It is generally agreed in the MDE community that moving the development and maintenance effort up to the early phases of development, where the domain model is a core artefact, results in more cost-effective maintenance processes and applications of better quality [29].

As mentioned in the introduction, as far as notations are concerned, we can distinguish between two main trends when representing domain models in software development proposals. The first one consists in the use of a graphical notation, which, usually influenced by the main *de facto* graphical modelling standards (*e.g.*, UML or ER), defines the model as a set of nodes (representing entities) and links (representing relationships among entities). The other trend consists in representing the domain model using a textual notation. Althought there are also textual modelling standards such as HUTN (Human-Usable Textual Notation) [33]), most textual notations ignore them, and are instead based on well-known object-oriented programming languages (Java, Python, *etc.*). Table 2 presents a subset of the most representative modelling approaches together with the notation they use to represent domain models. They are categorized by (a) the notation provided for the domain model (graphical and/or textual models), (b) whether such notation is based on a standard and (c) the underlying paradigm (object-oriented or data-oriented).

Table 2 Main modelling approaches for domain models.

Approach	Textual	Graphical	Standard	Paradigm
UML		✓	✓	OO
ER		✓	✓	DO
HUTN	✓		✓	OO
USE	✓	✓(UML)		OO
RadarC		✓(UML)		DO
OOH4RIA	✓	✓(UML)		OO
WebML		✓(UML,ER)		DO
OO-Method		✓(UML)		OO
UMPLE	✓	✓(UML)		OO

OO: Object-oriented DO: Data-oriented

It is important to note that, while some approaches only allow developers to use either a textual notation (e.g., HUTN) or a graphical one (e.g., OO-Method, RadarC), others provide both types of notations for the definition of domain models (*e.g.,* OOH4RIA [25],USE [11] and UMPLE [10]).

The underlying paradigm of each proposal is of particular interest, since it determines the set of primitives made available by the modelling language. Some domain models, such as the ones provided by ER and WebML, are data-oriented and, therefore, they just contain primitives for representing the data persistence layer of the application. In contrast, other models are object-oriented (usually based on the UML class diagram), and therefore they also contain primitives for representing business operations. More focused approaches go one step further and facilitate the definition of object-relational mapping rules based on the domain model (e.g., OOH4RIA or RadarC).

3 Description of the Experiment

In May 2013, we performed a quasi-experiment at the University of Alicante. A quasi-experiment is a type of controlled experiment in which individuals or teams of individuals (i.e., the unit of study) carry out one or more tasks in order to compare different processes, methods, techniques, languages or tools (i.e., the treatments) [16]. In quasi-experiments, subjects are not selected randomly, but according to specific criteria. Quasi-experiments have been widely used in this field even though, comparing with experiments, their degree of internal validity is inferior. The reason for its popularity is that they facilitate the analysis of cause-effect relationships in scenarios like ours, in which the cost of random decisions is very high [16].

3.1 Goals and Context Definition

Following the GQM template [35], the purpose of this study was *to evaluate* the effect of the selection of *a textual (OO language syntax) vs a graphical (UML) notation for the representation of a domain model* on the *efficiency, effectiveness and satisfaction while performing maintenance tasks* from the point of view of *junior software developers*. The context of the study is *3rd-year undergraduate Computer Science students at the University of Alicante using the OOH4RIA OIDE tool*.

For the analysis of maintainability, we have focused our study on two subcharacteristics: analysability and modifiability. The reason is that these are the two maintainability subcharacteristics that most heavily rely on model comprehension, and therefore are most directly influenced by modelling notations [27]. In contrast, other subcharacteristics such as testability, modularity and reusability are more related to the abstract syntax of the language, which, in our study, remains constant.

The design of the experiment is based on Wohlin's experimentation framework for Software Engineering [41].

The research questions addressed in this study, designed to be answered with quantitative data, are the following:

RQ1: Is the objective performance of maintainability (precision, coverage and efficiency of analysability, and effectiveness and efficiency of the modifiability) affected by the use of a domain model with a textual or graphical notation?

RQ2: Is the subjective performance of maintainability (efficiency and effectiveness perceived by the developers) affected by the use of a domain model with a textual or graphical notation?

RQ3: Is the satisfaction of the developers when performing maintenance tasks affected by the use of a domain model with a textual or graphical notation?

3.1.1 Subjects

The participants of this study were the students enrolled in the course Design of Software Systems (in Spanish, "Diseño de Sistemas Software"). The initial number of subjects was 86. Due to ethical issues, all the subjects were asked to sign a form in which they accepted to participate in the study and let their data to be used in anonymized and aggregated form for research purposes. Two subjects declined to participate, so we finally gathered the information of 84 students. From them, 14% were women and 86% were men. 56.6% of the sample had at least one year of working experience as a software developer. Regarding the background of the subjects on the technologies and methods used in the experiment, the questionnaire showed that (a) 82.5% of the subjects reported an intermediate-advanced level of knowledge in object-oriented programming; (b) 75.4% claimed to have an intermediate-advanced level of knowledge in relational databases; (c) 45.6% reported that they had

an advanced level of knowledge in modelling UML class diagrams; (d) 68.4% reported basic knowledge of OOH4RIA; and (e) 64.9% claimed intermediate-advanced knowledge in software applications development.

78 out of the 84 subjects (92.9% of the sample) correctly filled in the data gathering instrument corresponding to the analysability tasks (RQ1), while 67 subjects (79.8%) correctly filled in the data gathering instrument corresponding to the modifiability tasks. After a careful study, we can conclude that the errors made by subjects while completing the questionnaires were computer and network related, and therefore independent from the particular treatments applied in the experiment, so we can assume that the obtained results were not compromised.

After finishing the tasks, subjects were asked to fill in a questionnaire where they gave their perceptions regarding their subjective performance and satisfaction (RQ2 and RQ3). Again, participating in this part of the study was voluntary. 72 (85.7%) attempted to fill in in the questionnaire, 15 of whom did not finish. Therefore, the final sample over which RQ3 has been answered comprises 57 subjects (67.8%).

3.1.2 Treatments: Applications and Notations

We randomly assigned all the subjects to the following two domain models:

- Massive On-line Open Course manager (MOOC). This application manages a collection of MOOCs, facilitating the classification and sorting of the units and the process of enrolment of students and lecturers.
- TicketSeller application (Web platform for selling tickets). This application sells tickets for several events. The system manages the process of invoicing for each order and the process of communication and marketing with the customers by using mailing lists.

The main features of each model are the following:

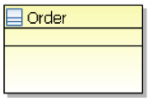
- MOOC: 9 classes, 45 attributes, 66 operations, 9 associations, 3 enumerations.
- TickerSeller: 12 classes, 68 attributes, 70 operations, 15 associations, 2 enumerations.

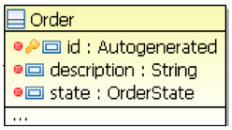
For each domain model, we created two representations, one textual and one graphical, using the OOH4RIA textual and graphical notation. The use of OOH4RIA allowed us to control all the variables related to the tool support, since both notations were inserted in the same tool and had the same options available, the only difference being the concrete syntax used. The table 3 depicts the notation of different elements of the OOH4RIA domain models. The table has three columns: the domain element, the textual representation and the graphical representation. First we start with the central element of the Domain Model, the Class element. The textual notation represents a class that contains the name, an alias database and the attributes and operations lists. On the other hand, the graphical notation presents a concrete syntax based

on UML class diagrams where the class is a rectangle that has three compartments: the upper one contains the name and other two compartments hold the attributes and operations lists. Since the graphical notation cannot visually depict all the properties of the elements, it must be complemented with a property view that follows a form-based representation. In order to depict the attributes and operations elements, we have followed the syntax of the UML. Thus, we represent some aspects as their names, datatype and visibility (private, public and protected) with the same notation. However, OOH4RIA has introduced some extensions in both elements that affect in their concrete syntax. In the attribute case, we have signalized the object identifier (OID) with a key in the graphical notation that permits to establish an adequate mapping between the class ID and the primary key of the mapped table. Moreover, the sqlType is used in cases where a datatype is specific of a database manager (e.g. Oracle, SQL Server, etc.). As regards the operation element, its concrete syntax is also affected by the abstract syntax extensions of OOH4RIA. We have introduced the operationType property, which permits to specify the functionality of an operation in a way that is used later to generate the final implementation. An operation can be classified as a CRUD operation (Create, Read, Update and Delete) or as a custom operation that represents any type of non CRUD operation that performs an arithmetical or statistical calculation or an invocation to an external library. In the operation examples, the figure shows different operation types. We would like to draw attention to the ReadFilter operation called getCart that contains a Hibernate Query language (HQL) expression to gather the Orders that are considered as a Cart (with attribute state equals 1) from the database. In the textual notation we can, by means of the property Filter, specify the HQL expression. The graphical notation, however, is limited by the UML standard and it needs a property view to specify the filter. Also, we have defined two relationships that could be defined in an object oriented domain model: the association and the inheritance. On the one hand, the association is a relationship between two domain classes that describes links between their object instances. Each association has two roles, and each role is an attribute with the type of the class related. It also has associated an upper and lower cardinality, a navigable property, and an aggregationKind property that establishes a whole/part relationship (none, shared or composite). All the association properties are expressed using the graphical notation, e.g. the color or the presence or absence of a diamond represents the aggregationKind, and the name and cardinalities of roles are represented together with the link. For its part, the textual notation presents a separated Association element that indicates the related classes and the two different roles with their properties. The other type of relationship is the Inheritance, that defines a taxonomic relationship between a more general class with a more specific class. Thus, a specific class inherits the features of a more generic one. Graphically, the inheritance is depicted with a white-head arrow that connects the specific class with the generic class. Textually, the specific class must include the word 'extends' with the name of the generic class. Last but not least, the enumeration element defines a new data type whose values

are defined as set of literals. On the one hand, the graphical notation represents an enumeration with a green rectangle with the type name and a container with the set of literals with the corresponding value. On the other hand, the textual notation also defines an individual element called Enumeration that represents the type name and different literals.

Element	Textual	Graphical

Class	'Class' name=EString ('alias' alias=EString)? '{' description=EString '}')? ('attributes' '{'(attributes+=Attribute ';')* '}')? ('operations' '{' (operations+=Operation ';')* '}')? '}')?;
	
	EXAMPLE:
	Class Order **alias** Order01{
	Attributes {}
	Operations{}
	};
		

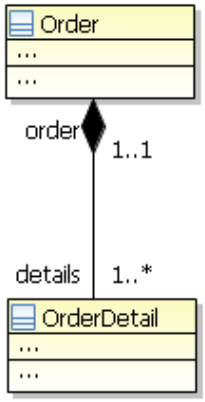
Attribute	Attribute_Impl: visibility=Visibility (isOID?='oid' | unique?='unique')? ('alias' alias=EString)? ':' (type=PrimitiveType | typeEnum=[Enumeration|Fqn]) ('(' lower=EString ',' upper=EString ')' ('{' ('description' '{' ' description=EString '}')? ('sqlType' '=' sqlType=EString)? '}')?; EXAMPLES: **private oid** id **alias** idOrder : Integer; **private** description: Text ("0","1") {sqlType="VARCHAR(256)"}; **private** state : OrderState;	

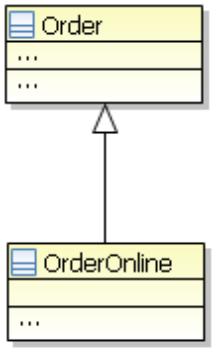
<table>
<tr>
<td>Operation</td>
<td>

```
Operation:
        visibility=Visibility
        name=EString
        ('::'
(operationType=OperationType
    ->' relatedRole=[AssociationRole |
Fqn])
        (customized?='customized')?
        (paging?='paging')? )?
        ('(' (argList=ArgumentList)? ')')?
         ':' ((type=PrimitiveType |
         typeEnum=[Enumeration|Fqn] |
         type=ReferenceType '->')
         ('{{'description' '{'
          description=EString '}')?    ('filter
         '=' filter=EString)?    '}')?;

EXAMPLES:
public new::New (p_customer : OID-
>Customer, p_description : String, p_stat
: OrderState) : Object->Order;

public modify::Modifier (p_Order_OID :
OID->Order, p_description : String,
p_state : OrderState) : Void;

public destroy::Destroy (p_Order_OID :
OID->Order) : Void;

public getCart::Custom () : Object-
>Order { filter = "FROM OrderEN as self
where self.State = 1"};
```

</td>
<td> 
</td>
</tr>
<tr>
<td>Association</td>
<td>

```
Association:
        'Association'  name=EString
        '(' classOrigin=[Class|Fqn] '-'
         classTarget=[Class|Fqn] ')'
        ('alias' alias=EString)?
        ('description' '{'
description=EString '}')?          '=>'
rolOrigin=AssociationRole ';'
        ('<=' rolTarget=AssociationRole
';')?
'}';

EXAMPLE:
Association orderDetail_order ( Order -
OrderDetail )
{
=> navigable orderDetail :
Composite("0", "*");
<= navigable order ("1", "1");
};
```

</td>
<td> 
</td>
</tr>
</table>

Inheritance	'Class' name=EString
	('extends' ascendant=[Class])? ...
	
	EXAMPLE:
	Class OrderOnline extends Order {
	...
	};
		

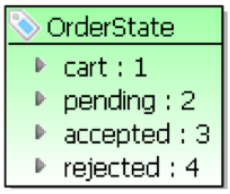
Enumeration	Enumeration: {Enumeration} 'Enumeration' name=EString '{' ('description' '{' description=EString '}')? (enumerationLiterals+= EnumerationLiteral (',' enumerationLiterals+=EnumerationLiteral)*)? '}' ; EXAMPLE: Enumeration OrderState { ("cart", 1), ("pending", 2), ("accepted", 3), ("rejected", 4) };	

Table 3: Notation of the OOH4RIA Domain Model Elements

The complete versions of the models (textual and graphical) used in this experiment are available as part of the experimental package at the following URL: `http://artemisa.dlsi.ua.es/ooh4ria/GraphicTextualExp/ExperimentMaterial.zip`

3.1.3 Implementation Environment

As we have already mentioned, the tool used in this study is the OOH4RIA Integrated Development Environment (OIDE) [24][1], developed as a collection of plug-in components for the Eclipse framework. It supports both a graphical concrete syntax of object-oriented domain models (based on the notation of the UML class diagrams) and an equivalent textual concrete syntax, similar to a traditional object-oriented language. The tool has already been used for three years at the University of Alicante in several Software Engineering courses.

[1] `http://suma2.dlsi.ua.es/ooh4ria/`

Table 4 Design of the experiment.

	Textual	**Graphical**
TicketSeller	S1-S43 (5 A. + 5 M)	S1-S43 (5 A. + 5 M.)
MOOC	S44-S86 (5 A. + 5 M.)	S44-S86 (5 A. + 5 M.)

A: Analysability M: Modifiability S: Subject

3.2 Experimental Setup

In order to answer the research questions of Section 3.1, we applied a two way, mixed model ANOVA design.

First, we defined two domains, and defined 20 tasks (10 analysability tasks and 10 modifiability tasks) for each domain. We then randomly assigned the subjects to one of the two possible domains. Subsequently, we asked them to detect five errors (analysability tasks) and to modify five aspects of the model (modifiability tasks) with each notation. In order to alleviate the learning bias, inside each domain half of the users started with the textual notation, and half of the users started with the graphical notation.

As discussed in Sect.3.1.2, both domains were similar in complexity, and the tasks on the two domains were defined so that they were also equal in complexity (e.g. if one task involved checking the cardinality of a given relationship in the MOOC application, the corresponding task in the TicketSeller application equally involved checking the cardinality of a similar relationship). Furthermore, the tasks were designed based on a taxonomy of frequently committed errors during the development of domain models by novice system analysts [20].

The design of the experiment is illustrated in Table 4.

3.2.1 Variables

We defined the following independent variables (IV, also called factors or treatments):

- *Not*: Notation, a categorical, intra-subject variable with two possible values: Textual and Graphical. *Not* is a fixed factor, since it includes the two types of notation in which we are interested in this experiment.
- *App*: Application, a categorical, inter-subject variable with two levels: TicketSeller and MOOC. *App* is a random factor, since the proposed applications just provide two examples of domain models.

The dependent (or measurable) variables (DV) were defined as follows:
Analysability measures:

- *AAPrec*: Actual Analysability Precision. This measure of effectiveness (type ratio) represents the percentage of errors correctly detected with respect to the total number of detected errors. Range: from 0% (all the errors reported were false errors) to 100% (all the errors reported were true errors).

- *AACov*: Actual Analysability Coverage. This measure of effectiveness (type ratio) represents the percentage of errors correctly detected with respect to the total number of errors in the domain model. Range: from 0% (no error detected) to 100% (all the 5 errors were detected).
- *AAEffc*: Actual Analysability Efficiency. This measure (type ratio) represents the number of true errors that a subject can detect in an hour (AACov/Time).
- *PAEffv*: Perceived Analysability Effectiveness. This measure (type ratio) represents the percentage of errors that the subject believes to have detected correctly. Range: from 0% to 100%.
- *PAEffc*: Perceived Analysability Efficiency. This measure (type interval) represents the speed of error detection perceived by the subject. Range: from 1 (very slowly) to 7 (very fast).

Modifiability measures:

- *AMEffv*: Actual Modifiability Effectiveness. This measure (type ratio) represents the percentage of modification tasks correctly performed by the subjects. Range: from 0% to 100%.
- *AMEffc*: Actual Modifiability Efficiency. This derived measure (type ratio, AMEffv/Time) represents the number of modification tasks correctly performed by a subject in an hour.
- *PMEffv*: Perceived Modifiability Effectiveness. This measure (type ratio) represents the percentage of modification tasks the subject believes to have done correctly. Range: from 0% to 100%.
- *PMEffc*: Perceived Modifiability Efficiency. This measure (type interval) represents the speed at which the subject believes to have completed the tasks. Range: 1 (very slowly) to 7 (very fast).

Global measures of maintainability:

- *Satisf*: Satisfaction of the developers after completing the tasks of analysis and modification. Type interval, based on a semantic-differential Likert scale with 11 seven-point items.

3.2.2 Hypotheses

Based on the research questions of Section 3.1 and the defined measures, this quasi-experiment includes the following hypotheses:

- Actual Analysability Precision (HAAPrec, RQ1)
  - HAAPrec_0: $\text{AAPrec}_{\text{Textual}} = \text{AAPrec}_{\text{Graphical}}$. Using the OOH4RIA domain model, subjects are equally accurate in detecting errors with the textual or the graphical notations, independently from the software application developed.
  - HAAPrec_A: $\text{AAPrec}_{\text{Textual}} \neq \text{AAPrec}_{\text{Graphical}}$
- Actual Analysability Coverage (HAACov, RQ1)

  - $HAACov_0$: $AACov_{Textual} = AACov_{Graphical}$. Using the OOH4RIA domain model, subjects detect the same number of errors with both notations, independently from the software application developed.
  - $HAACov_A$: $AACov_{Textual} \neq AACov_{Graphical}$
- Actual Analysability Efficiency (HAAEffc, RQ1)
  - $HAAEffc_0$: $AAEffc_{Textual} = AAEffc_{Graphical}$. Using the OOH4RIA domain model, subjects are equally efficient (AACov/Time) in detecting errors with both notations, independently from the software application developed.
  - $HAAEffc_A$: $AAEffc_{Textual} \neq AAEffc_{Graphical}$
- Actual Modifiability Effectiveness (HAMEffv, RQ1)
  - $HAMEffv_0$: $AMEffv_{Textual} = AMEffv_{Graphical}$. Using the OOH4RIA domain model, subjects are equally effective in performing modifications in the applications using both notations, independently from the software application developed.
  - $HAMEffv_A$: $AMEffv_{Textual} \neq AMEffv_{Graphical}$
- Actual Modifiability Efficiency (HAMEffv, RQ1)
  - $HAMEffc_0$: $AMEffc_{Textual} = AMEffc_{Graphical}$. Using the OOH4RIA domain model, subjects are equally efficient (AMEffv/Time) in performing modifications in the applications with both notations, independently from the software application developed.
  - $HAMEffc_A$: $AMEffc_{Textual} \neq AMEffc_{Graphical}$
- Perceived Analysability Effectiveness (HPAEffv, RQ2)
  - $HPAEffv_0$: $PAEffv_{Textual} = PAEffv_{Graphical}$. Using the OOH4RIA domain model, subjects believe to be equally effective in detecting errors with both notations, independently from the the software application developed.
  - $HPAEffv_A$: $PAEffv_{Textual} \neq PAEffv_{Graphical}$
- Perceived Analysability Efficiency (HPAEffc, RQ2)
  - $HPAEffc_0$: $PAEffc_{Textual} = PAEffc_{Graphical}$. Using the OOH4RIA domain model, subjects believe to be equally efficient in detecting errors with both notations, independently from the the software application developed.
  - $HPAEffc_A$: $PAEffc_{Textual} \neq PAEffc_{Graphical}$
- Perceived Modifiability Effectiveness (HPMEffv, RQ2)
  - $HPMEffv_0$: $PMEffv_{Textual} = PMEffv_{Graphical}$. Using the OOH4RIA domain model, subjects believe to be equally effective in performing modifications in the applications with both notations, independently from the software application developed.
  - $HPMEffv_A$: $PMEffv_{Textual} \neq PMEffv_{Graphical}$
- Perceived Modifiability Efficiency (HPMEffc, RQ2)
  - $HPMEffc_0$: $PMEffc_{Textual} = PMEffc_{Graphical}$. Using the OOH4RIA domain model, subjects believe to be equally efficient in performing modifications in the applications with both notations, independently from the software application developed.
  - $HPMEffc_A$: $PMEffc_{Textual} \neq PMEffc_{Graphical}$

Table 5 Sensitivity Analysis of the design (desired power=0.8).

Dependent Variables	**# observations**	**Not η^2**	**App η^2**
AAPrec, AACov, AAEffc	78	0.16	0.28
AMEffv, AMEffc	67	0.17	0.30
PAEffv, PAEffc, PMEffv, PMEffc, Satisf.	57	0.33	-

- Maintainability Satisfaction (Analysability and Modifiability) (HSatisf, RQ3)
  - HSatisf_0: $\text{Satisf}_{\text{Textual}} = \text{Satisf}_{\text{Graphical}}$. Using the OOH4RIA domain model, subjects feel equally satisfied with both notations when they perform modifications in the applications, independently from the software application developed.
  - HSatisf_A: $\text{Satisf}_{\text{Textual}} \neq \text{Satisf}_{\text{Graphical}}$

3.2.3 Sensitivity Analysis of the Design

Last but not least, in order to validate our experimental setup, we conducted a sensitivity analysis. Since our sample size is fixed (determined by the number of students enrolled in the course) and cannot be changed, this analysis has served us not to establish the adequate number of subjects but to qualify the meaning of our results. In particular, the goal of this analysis is to ensure that not only does the design of the experiment reject the null hypothesis with a degree of confidence higher than 95% ($\alpha = 0.05$), but also the design has sufficient power to limit the risk of Type II errors (i.e., not rejecting the null hypothesis when it is actually false). Cohen [4] suggests that the power of an analysis should be greater than 0.7 to be useful. Following this recommendation, we have established the power of the study to 0.8. With this value settled, it is possible to calculate the effect size that is detectable in our study, that is, how much the independent variable has to affect the dependent variable to be detectable in our experimental setting. With ANOVA, the effect size is measured through eta-squared (η^2, see Table 5). All these calculations were performed with G*Power 3.1.7 [8].

As we can observe in Table 5, the effect size detectable by our experiment due to the change of notation for the AAPrec, AACov and AAEffc dependent variables (78 valid observations) is 0.16. This value means that our experimental design actually will detect -with 95% certainty- significant differences between the notations (reject the null hypothesis) when the notation used is responsible for at least 16% of the total variability in the dependent variable (effect + error). On the contrary, we can be 80% sure that, if our analysis does not reject the null hypothesis, it is because the notation is responsible for less

than 16% of the variability of the dependent variable (and therefore, in practice, the effect of choosing one or other notation is not of great importance).

The same interpretation holds for the remaining values in Table 5.

As we can observe, the detectable notation effect sizes (which is our main concern in this paper) are small enough to go on with the study, since this means that, if we do not find statistical significant differences, we can be reasonably sure that the real effect size of notation on the dependent variable is of little concern, at least in the context tested in our study.

3.3 Operation and Data Gathering

We conducted this experiment during a practical session of the Design of Software Systems course (2 hours). Prior to the experiment, the subjects had provided data regarding their personal background and professional career, including their previous experience with similar notations. Also, we made sure that subjects received the appropriate training on both the notations and the OIDE model editors before the experiment. The time devoted for training was 2 hours for each notation.

First, the subjects were communicated the application that they had to download, and they performed the 20 tasks (10 using the graphical notation and 10 using the textual notation). As mentioned before, in order to alleviate the learning bias, the order in which they applied the notations was randomized inside each group. After completing the tasks, the subjects filled in a post-experiment questionnaire that measured their subjective degree of efficiency, effectiveness and satisfaction regarding the use of both notations in the experiment.

In order to maintain the comparability of the results, during the experiment we did not provide feedback to the participants about their performance in the tasks. Moreover, during the experiment, the subjects were supervised by two lecturers in order to control the interaction bias.

The objective measures (AEffc and AEffv) were calculated based on the manual correction of the tasks. Such correction was performed by two lecturers. For the students, this experiment was an assignment of the course. Some subjects were not able to properly submit the results for all the tasks. For each analysis, we only considered those subjects who submitted the results for all the tasks associated to the analysis. For this reason, the degrees of freedom vary between analysability and modifiability statistical analyses.

The data for the subjective measures was gathered by means of an online questionnaire, created with Qualtrics (`http://www.qualtrics.com/`). Again, some of the subjects did not complete the questionnaire. For this reason, the degrees of freedom of the analyses of the subjective hypotheses also vary.

Table 6 Mean and standard deviation for each variable.

Var	Textual		Graphical		TicketSeller		MOOC	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
AAPrec	79.04	20.43	75.70	19.43	78.31	16.93	75.53	23.09
AACov	73.08	22.98	65.90	17.69	72.62	18.04	65.48	23.16
AAEffc	9.79	5.94	9.38	5.86	10.25	6.32	8.81	5.24
AMEffvR	84.04	19.72	83.16	20.19	76.22	15.76	78.24	14.55
AMEffc	20.39	8.66	14.59	8.17	17.70	9.38	17.45	8.54
PAEffv	79.65	17.21	76.84	18.82	79.31	16.21	76.43	20.81
PAEffc	4.09	1.29	4.40	1.49	4.25	1.46	4.24	1.18
PMEffv	84.38	2.72	83.09	2.80	83.19	19.92	84.29	20.02
PMEffc	4.89	0.17	4.94	0.21	4.76	1.53	5.07	1.13
Satisf	4.13	1.06	4.55	1.23	4.38	1.11	4.27	1.26

4 Data Analysis

The descriptive statistics corresponding to the measures used in this experiment are presented in Table 6.

All the analyses described in this section were performed using PASW (Predictive Analytics SoftWare, [32]) v18.

4.1 Scale Reliability

The first step of the analysis was the validation of the reliability of the *Satisf* scale used in the experiment by means of a Cronbach's Alpha test. This test yielded a value of 0.942, in which all the elements contributed more than 0.3 to the general constructor. The resulting value indicates that the internal consistency of the 11 scale items is very high. Therefore, we proceeded to calculate its mean and use it as a global measure of Satisf.

4.2 General Issues

The kind of statistical analysis used in a two-way, mixed model ANOVA design requires that the dependent variables meet the following requirements: *(1)* normality of the variables (the scores for each condition should be normally distributed around the mean); *(2)* homogeneity of the variance (each population, i.e., scores of the textual notation and scores of the graphical notation, should have the same error variance); and *(3)* sphericity of the covariance matrix (which ensures that the F ratios are adjusted to the F distribution).

In the following analyses all the assumptions have been checked. In case of violation of any of these assumptions, the Greenhouse & Geisser's adjustment

to the ANOVA analysis has been applied. The design of the experiment is balanced (i.e., similar number of observations in all the cells), which contributes to the robustness of the analyses.

To perform the analyses we have applied a two-way mixed-design ANOVA ($\alpha = 0.05$), in which Not is the fixed intra-subject factor and App is the random inter-subject factor. In order to facilitate the reading, Table 6 summarizes the means and standard deviations of the measured variables, while Table 7 summarizes the results of the analyses. For all the analyses, the first step has consisted in checking the interaction Not*App. Unless explicitly stated, such interaction has been not significant (see Table 7), which means that we have been able to safely examine the main effects of the two independent variables (Not and App) with no need of qualifying the results. Also, in all the cases, the App used has revealed itself as not significant, despite the small differences found in the variable means (see Table 6). For the sake of space, from here on we will therefore center on the analysis of the Not variable, which is the central focus of this paper. Interested readers on the description of the whole table are referred to the technical report that accompanies this paper, located at the following URL: `https://suma.dlsi.ua.es/ooh4ria/TR_MTG.pdf`.

4.3 RQ1: Objective Performance of the Notations

In order to answer to RQ1, five hypotheses have been analysed (see Section 3.2.2), namely: HAAPrec, HAACov, HAAEffc, HAMEffv, HAMEffc.

4.3.1 Actual Analysability Precision

The actual precision of the subjects when they used the textual notation of the OOH4RIA domain model for analysability tasks (M = 79.04, SD = 20.43) is slightly higher than their precision when using the graphical notation (M = 75.70, $SD = 19.43$). However, this difference is not significant ($F(1, 76) = 2.55$, $MSE = 770.38$, $p > 0.05$, $\eta^2 = 0.033$)

4.3.2 Actual Analysability Coverage

For the Not variable, the actual coverage of the subjects when they use the textual notation for analysability tasks ($M = 73.08$, $SD = 22.98$) is higher than the coverage when they use the graphical notation ($M = 65.90$, $SD = 17.69$). This difference is significant ($F(1, 76) = 9.009$, $MSE = 2134.86$, $p < 0.05$, $\eta^2 = 0.106$).

These results are illustrated in Figure 1. In this chart, the proximity of the lines corresponding to each application shows that the type of the application was not significant for the results. The influence of the Not variable can be appreciated from the line slopes (a lack of influence would mean approximately flat lines). Finally, the fact that the slope of both lines goes in the same direction means that the interaction Not*App is not significant. The same clues apply to interpret the rest of the charts in this paper.

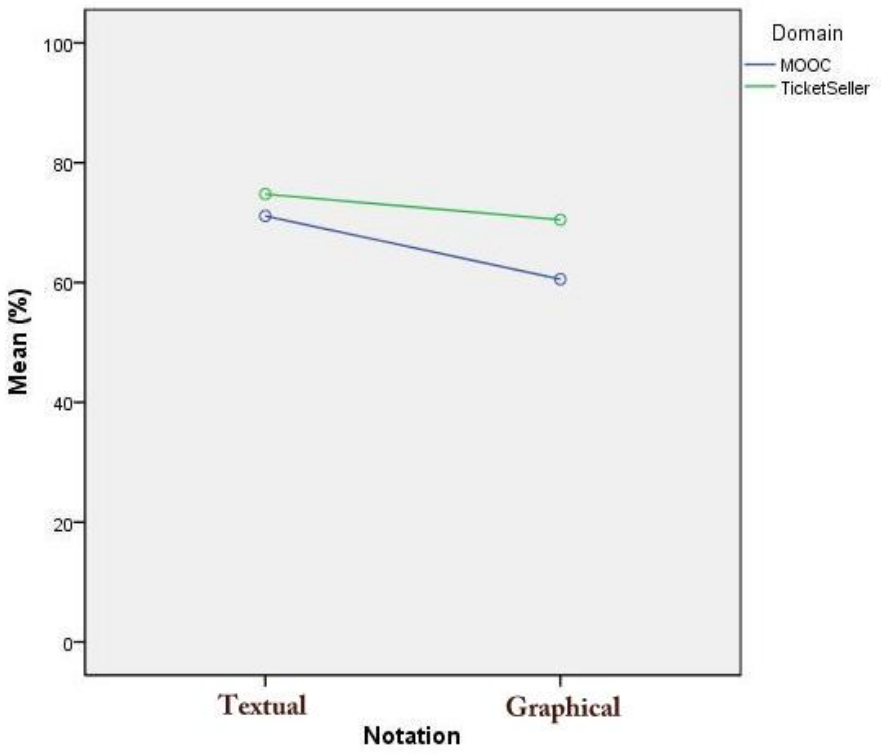


Fig. 1 Actual Analysability Coverage

4.3.3 Actual Analysability Efficiency

For the Not variable, the actual efficiency of the subjects when they use the textual notation ($M = 9.79$, $SD = 5.94$) is almost equal to their efficiency when they use the graphical notation ($M = 9.38$, $SD = 5.86$). This difference is not significant ($F(1, 76) = 0.399$, $MSE = 9.474$, $p > 0.05$, $\eta^2 = 0.005$).

4.3.4 Actual Modifiability Effectiveness

The HAMEffv hypothesis is the only case in which the interaction Not*App is significant ($F(1, 65) = 9.483$, $MSE = 1600.813$, $p < 0.05$, $\eta^2 = 0.127$, see Table 7). Therefore we must treat the individual effects of Not and App with precaution.

For the Not variable, the actual effectiveness of the subjects when they use the textual notation for modifiability tasks ($M = 84.04$, $SD = 19.72$) is slightly higher than their effectiveness when they use the graphical notation ($M = 83.16$, $SD = 20.19$). However, the difference is not significant ($F(1, 65) = 0.288$, $MSE = 48.574$, $p > 0.05$, $\eta^2 = 0.004$).

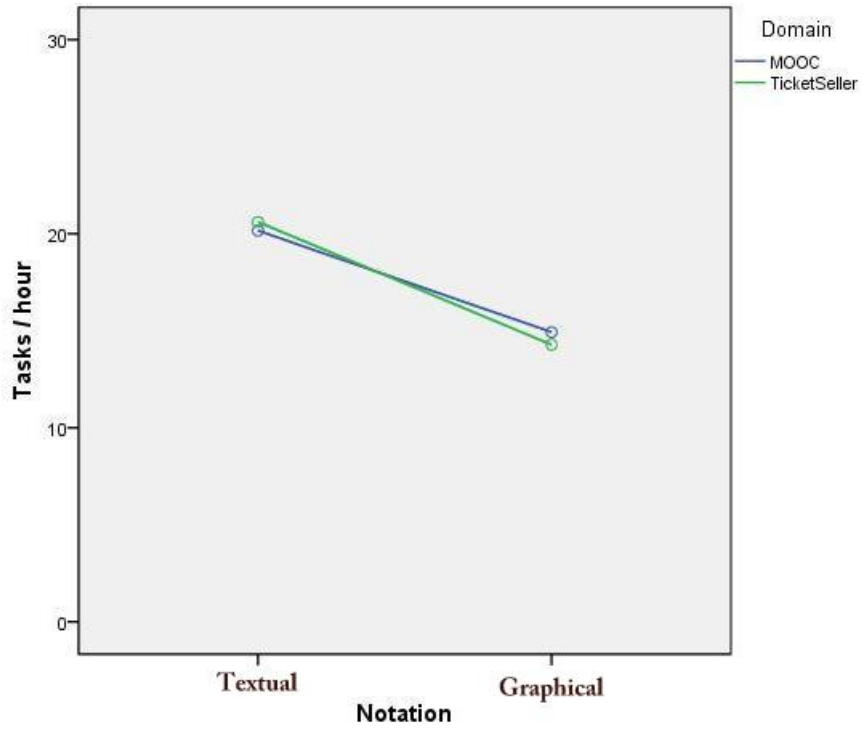


Fig. 2 Actual Modifiability Efficiency

4.3.5 Actual Modifiability Efficiency

For the Not variable, the actual efficiency of the subjects when they use the textual notation of the OOH4RIA domain model to support the maintainability tasks ($M = 20.39$, $SD = 8.66$) is higher than their efficiency when they use the graphical notation ($M = 14.59$, $SD = 8.17$). In this case, the difference is significant ($F(1, 65) = 14.906$, $MSE = 1114.743$, $p < 0.05$, $\eta^2 = 0.187$).

The obtained results are illustrated in Figure 2.

4.4 RQ2: Subjective Performance of the Notations

In order to answer to RQ2, four hypotheses have been analysed (see Section 3.2.2), namely: HPAEffv, HPAEffc, HPMEffv, and HPMEffc.

4.4.1 Perceived Analysability Effectiveness

For the Not variable, the effectiveness perceived by the subjects when they use the textual notation for analysability tasks ($M = 79.65$, $SD = 17.21$) is slightly higher than the effectiveness perceived when they use the graphical

notation ($M = 76.84$, $SD = 18.82$). However, the difference is not significant ($F(1, 55) = 2.003$, $MSE = 327.16$, $p > 0.05$, $\eta^2 = 0.035$).

4.4.2 Perceived Analysability Efficiency

For the Not variable, the efficiency perceived by the subjects when they use the textual notation of the OOH4RIA domain model for analysability tasks ($M = 4.09$, $SD = 1.29$) is slightly lower than the efficiency perceived when they use the graphical notation ($M = 4.40$, $SD = 1.49$). However, this difference is not significant ($F(1, 55) = 1.80$, $MSE = 2.23$, $p > 0.05$, $\eta^2 = 0.032$).

4.4.3 Perceived Modifiability Effectiveness

For the Not variable, the effectiveness perceived by the subjects when they use the textual notation of the OOH4RIA domain model for modifiability tasks ($M = 84.38$, $SD = 2.72$) is slightly higher than the perceived effectiveness when they use the graphical notation ($M = 83.09$, $SD = 2.80$). The difference is not significant ($F(1, 55) = 0.225$, $MSE = 44.121$, $p > 0.05$, $\eta^2 = 0.004$).

4.4.4 Perceived Modifiability Efficiency

For the Not variable, the efficiency perceived by the subjects when they use the textual notation of the OOH4RIA domain model for the modifiability tasks ($M = 4.89$, $SD = 0.17$) is approximately equal to the efficiency when they use the graphical notation ($M = 4.94$, $SD = 0.21$). This difference is not significant ($F(1, 55) = 0.036$, $MSE = 0.55$, $p > 0.05$, $\eta^2 = 0.001$).

4.5 RQ3: Satisfaction of the Notations

Finally, for the Not variable, the satisfaction of the subjects when they use the textual notation for maintainability tasks (be them analysability or modifiability tasks) ($M = 4.13$, $SD = 1.06$) is slightly lower than the satisfaction of using the graphical notation ($M = 4.55$, $SD = 1.23$). However, the difference is not significant ($F(1, 55) = 3.531$, $MSE = 5.652$, $p > 0.05$, $\eta^2 = 0.060$).

If we look back at table 6, we can observe how, for the two notationally significant variables (AACov and AMEffc), the textual notation was associated with better performance.

4.6 Threats to the Validity of the Study

The analysis of the threats to the validity of the study assesses the circumstances under which our experiment is applicable and offers benefits, and the ones under which our experiment may not be representative. Cook distinguishes four types of threats [5]: conclusion, internal, construct and external threats. Subsequently, we briefly discuss them.

Table 7 Summary of results.

Var	**p-values**			**Effect size (eta-squared)**		
	Not*App	Not	App	Not*App η^2	Not η^2	App η^2
AAPrec	0.577	0.114	0.385	0.04	0.033	0.01
AACov	0.209	**0.004**	0.085	0.021	**0.106**	0.039
AAEffc	0.159	0.53	0.188	0.026	0.005	0.023
AMEffv	**0.003**	0.593	0.397	**0.127**	0.004	0.011
AMEffc	0.716	**¡0.0001**	0.944	0.002	**0.187**	0
PAEffv	0.285	0.163	0.507	0.021	0.035	0.008
PAEffc	0.647	0.185	0.969	0.004	0.032	0
PMEffv	0.566	0.367	0.821	0.006	0.004	0.01
PMEffc	0.698	0.850	0.318	0.003	0.001	0.018
Satisf	0.502	0.066	0.58	0.008	0.060	0.006

The threats to the **conclusion validity** refer to issues that affect the capacity of drawing a correct statistic conclusion about the relationships between the treatments (*i.e.*, the type of notation and application) and the outcome of the experiment (*i.e.*, efficiency, effectiveness and satisfaction of developers). In order to mitigate these threats, in our study, we have performed a sensibility analysis that, given our sample size, ensures a power = 0.8 for effect sizes larger than 0.16 (analysability), 0.17 (modifiability) and 0.19 (satisfaction). Furthermore, we have verified the main assumptions of the statistic tests and we have applied the adjustments required to fulfil them. We would like to pinpoint that the use of 5- or 7-item Likert scales is a controversial topic in the scientific community. However, the use of ANOVA, which has a high degree of robustness regarding the ordinality and non-normality of the scales [31], significantly mitigates this threat. Moreover, we have checked the reliability of the only scale used in the experiment, i.e., the satisfaction scale. Last but not least, we have created an experimental package that standardises the application of treatments to subjects. The use of a homogeneous group of subjects and the absence of unexpected events during the stage of data gathering also contribute to the conclusion validity of our experiment.

The threats of the **internal validity** refer to the possibility of the existence of hidden factors, i.e., out of control in the experiment, which may provide alternative explanations for the result. In order to mitigate this risk, in our experiment, all the subjects except two participated in the study, so that there is no selection bias apart from the one inherent to the quasi-experiments. The subjects applied both treatments with different tasks, so that the biases of maturity, demoralisation and compensatory rivalry are limited. Moreover, two lecturers supervised the experimental process in order to limit the interaction bias.

The threats to the **construct validity** refer to the risks that the treatments do not correctly capture the theoretical causal construct, or that the dependent measures do not correctly capture the theoretical effect construct. In our case, the application of the treatments to more than one system limits the mono-operation bias, while the change of order applied to the treatments limit the effect of the possible interaction between treatments. We did not discuss the hypothesis of study before conducting the experiment and we provided explicit instructions to the facilitators to keep neutral and do not express their preferences between treatments. The main construct validity threat in this experiment is the mono-method bias: the theoretical constructs (efficiency, effectiveness and satisfaction) were measured with only one measure type. This was partially mitigated by choosing measures that are frequently employed in literature for the same purposes.

Finally, the threats to the **external validity** refer to the capacity of generalisation of the results to the general population. Our sample (i.e., 3rd year students of the Computer Engineering degree) is a weak representative of the population of software developers in business environments. In addition, the conditions of the test under which we conducted the experiment (in a classroom, under exam conditions) do not reflect the conditions of a working environment. Unfortunately this is a common situation for this type of experiments [29]. Furthermore, due to time constraints, we defined simplified versions of the applications, which do not reflect the complexity of some of the systems developed in the industry. We partially mitigated this risk by using partial domain models that were part of real projects. Last but not least, the use of two specific concrete syntaxes (UML vs a typical OO language syntax), with specific language characteristics, prevents us from extending the conclusions to textual vs graphical languages in general. Such generalization will only be possible after several replications of the study with different exponents of both notation types have been made.

5 Discussion

Our data analysis (see Table 7) shows how, for junior software developers and small domain models, the use of a textual notation significantly improves the analysability coverage: subjects using a textual notation found 73.08% of the total errors, while they only found 65.90% when using a graphical notation. Moreover, they perceived this improvement, although somehow attenuated: they thought they had correctly found 79.65% of the errors with the textual notation, and 76.84% with the graphical notation. These results are aligned with the claims made by authors such as [16,14,12,36], as it was discussed in Section 2. Our contribution in this sense has been to provide empirical data that now support such claims.

The data also shows that subjects were significantly more efficient performing modifications over the textual notation (20.39 successfully completed tasks per hour) rather than over the graphical notation (14.59 tasks success-

fully completed per hour). This finding is also aligned with claims made by some authors [12,36]. Nevertheless, subjects felt slightly more efficient with the graphical notation (4.94 over 7) than with the textual notation (4.89 over 7). It also draws the attention that subjects, when expressing their global satisfaction with the notations, still prefer a graphical one: for the Satisf variable the differences between notations are almost significant (p=0.06, barely over the 0.05 α threshold, see Table 7). Otherwise stated, data reveals how junior developers' intuition may drive them towards using a graphical notation, despite the fact that textual ones show slight benefits in objective terms. This fact is aligned with Meyer's claims [30] about the superior attractiveness of graphical notations.

We believe that this may be partly due to the way the Software Engineering curricula at universities fosters the use of graphical modelling languages for the definition of software domain models, while giving -at least in our case- little attention to textual alternatives. We might reconsider the reasons why we prefer to teach graphical models, such as UML class diagrams, over textual models, which, under some conditions, can improve the effectiveness and efficiency of designers, as shown by our data. This consideration has been also introduced by other authors such as Hutchinson *et al.* [13].

Our study also reveals that, given the relatively small explanatory power (effect size) of the notation variable in maintainability gains (see Table 7), even if AACov and AMEff are significantly affected by the notation used, maintainability gains seem not to be big enough to justify on their own a change of notation in software development processes. This somehow qualifies the advantages of textual languages over graphical ones reported in literature: although we agree with all the authors mentioned in Section **??** in that they exist, now we can say that, according to our data, their impact -at least with the kind of developers and projects used in our study- is quite small.

For all the hypotheses that could not be rejected, the sensitivity analysis performed (see Sect. 3.2.3) indicates that the effect size of notation over these dependent variables is relatively small (lower than 0.16 for AAPrec and AAEffc, lower than 0.17 for AMEffv, and lower than 0.19 for PAEffv, PAEffc, PMEffv, PMEffc, Satisf). This is also a useful piece of information for practitioners: even if junior developers detect slightly more errors and correct errors quicker with textual notations, is this enough to justify the cost and effort involved in changing the notation in which a software project is modelled? This question is even more tricky if we take into account that these gains need to be qualified by the subjective greater satisfaction of the developers with the graphical notation, which, given the nature of the development work, may hamper the achievement of the purported gains in case of forcing such change.

6 Conclusions & Future Work

This study compares the impact of a textual and a graphical notation on the analysability and modifiability of domain models. For this purpose, we have

defined a set of variables that allowed us to measure the efficiency, effectiveness and satisfaction of junior developers with each notation. Subjects in the experiment performed significantly better both for analysability coverage and modifiability efficiency with a textual notation, while none of the measures was significantly in favor of graphical notations. Despite this, subjects showed a slight preference towards the graphical notation of the domain models used in the experiment. The most relevant threat to the validity of our results is that the quasi-experiment was performed with students and small models (external threat). Our next step will therefore be to replicate the experiment with more complex models and more experienced software developers. Also important, we plan to perform a set of replications in which other textual and graphical languages (based on different DSL's) are tested. By replicating and sharing the results of these studies, we hope that in the future it will be possible to analyze the relationship between the cognitive dimensions of the different languages used in this family of experiments (which are affected by the cognitive principles applied for their design [28]) and maintainability.

References

1. L. Angyal and L. Lengyel. Synchronization of textual and visual representations of evolving information in the context of model-based development. In *EUROCON 2009, EUROCON'09. IEEE*, pages 420–425. IEEE, 2009.
2. A. Blackwell and T. Green. Notational systems–the cognitive dimensions of notations framework. *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science. Morgan Kaufmann*, pages 103–134, 2003.
3. N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan. Types of software evolution and software maintenance. *Journal of software maintenance and evolution: Research and Practice*, 13(1):3–30, 2001.
4. J. Cohen. Statistical power analysis. *Current directions in psychological science*, 1(3):98–101, 1992.
5. T. D. Cook, D. T. Campbell, and A. Day. *Quasi-experimentation: Design & analysis issues for field settings*. Houghton Mifflin Boston, 1979.
6. W. J. Dzidek, E. Arisholm, and L. C. Briand. A realistic empirical evaluation of the costs and benefits of uml in software maintenance. *Software Engineering, IEEE Transactions on*, 34(3):407–432, 2008.
7. E. Evans. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.
8. 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*, 39(2):175–191, 2007.
9. I. O. for Standardization. ISO/IEC FCD 25010: systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality model. http://www.iso.org, 2011.
10. A. Forward. *The convergence of modeling and programming: facilitating the representation of attributes and associations in the umple model-oriented programming language*. University of Ottawa, 2010.
11. M. Gogolla, F. Büttner, and M. Richters. Use: A uml-based specification environment for validating uml and ocl. *Science of Computer Programming*, 69(1):27–34, 2007.
12. H. Grönniger, H. Krahn, B. Rumpe, M. Schindler, and S. Völkel. Text-based modeling. In *4th International Workshop on Software Language Engineering*, 2007.
13. J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of mde in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 471–480, New York, NY, USA, 2011. ACM.

14. D. Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.
15. F. Jouault, J. Bézivin, and I. Kurtev. Tcs:: a dsl for the specification of textual concrete syntaxes in model engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering*, pages 249–254. ACM, 2006.
16. V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K Sjøberg. A systematic review of quasi-experiments in software engineering. *Information and Software Technology*, 51(1):71–82, 2009.
17. S. M. Kosslyn and J. R. Pomerantz. Imagery, propositions, and the form of internal representations. *Cognitive Psychology*, 9(1):52–76, 1977.
18. J. Krogstie. evaluating uml: A practical application of a framework for the understanding of quality in requirements specifications and conceptual modeling. In *Norwegian Informatics Conference (NIK)*, volume 35, page 37, 2000.
19. M. Leotta, F. Ricca, G. Antoniol, V. Garousi, J. Zhi, and G. Ruhe. A pilot experiment to quantify the effect of documentation accuracy on maintenance tasks. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, pages 428–431. IEEE, 2013.
20. F. Leung and N. Bolloju. Analyzing the quality of domain models developed by novice systems analysts. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 188b–188b. IEEE, 2005.
21. M. E. Manso, J. A. Cruz-Lemus, M. Genero, and M. Piattini. Empirical validation of measures for uml class diagrams: a meta-analysis study. In *Models in Software Engineering*, pages 303–313. Springer, 2009.
22. Y. Martínez, C. Cachero, and S. Meliá. Empirical study on the maintainability of web applications: Model-driven engineering vs code-centric. *Empirical Software Engineering*, pages 1–34, 2013.
23. Y. Martínez, C. Cachero, and S. Meliá. Mdd¡ i¿ vs.¡/i¿ traditional software development: A practitioners subjective perspective. *Information and Software Technology*, 55(2):189–200, 2013.
24. S. Meliá, J. J. M. Domene, Á. Pérez, and J. Gómez. Ooh4ria tool: Una herramienta basada en el desarrollo dirigido por modelos para las rias. In *JISBD*, pages 219–222, 2009.
25. S. Meliá, J. Gómez, S. Pérez, and O. Díaz. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. *2008 Eighth International Conference on Web Engineering*, pages 13–23, July 2008.
26. P. Mohagheghi and J. Aagedal. Evaluating quality in model-driven engineering. In *Modeling in Software Engineering, 2007. MISE'07: ICSE Workshop 2007. International Workshop on*, pages 6–6. IEEE, 2007.
27. P. Mohagheghi and R. Conradi. An empirical study of software change: origin, acceptance rate, and functionality vs. quality attributes. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, pages 7–16. IEEE, 2004.
28. D. L. Moody. The physics of notations: a scientific approach to designing visual notations in software engineering. In *Proceedings of the 32nd ACM/IEEE ICSE-Volume 2*, pages 485–486. ACM, 2010.
29. D. L. Moody, G. Sindre, T. Brasethvik, and A. Sølvberg. Evaluating the quality of information models: empirical testing of a conceptual model quality framework. In *Proceedings of the 25th International Conference on Software Engineering*, pages 295–305. IEEE Computer Society, 2003.
30. B. A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123, 1990.
31. G. Norman. Likert scales, levels of measurement and the lawsof statistics. *Advances in health sciences education*, 15(5):625–632, 2010.
32. M. J. Norusis et al. *PASW statistics 18 guide to data analysis*. Prentice Hall Press, 2010.
33. O. M. G. OMG. UML Human-Usable Textual Notation (HUTN), 2004.
34. J. Pardillo and C. Cachero. Domain-specific language modelling with uml profiles by decoupling abstract and concrete syntaxes. *Journal of Systems and Software*, 83(12):2591–2606, 2010.

35. D. E. Perry, A. A. Porter, and L. G. Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 345–355. ACM, 2000.
36. M. Petre. Why looking isn't always seeing: readership skills and graphical programming. *Communications of the ACM*, 38(6):33–44, 1995.
37. G. Scanniello, C. Gravino, M. Genero, J. Cruz-Lemus, and G. Tortora. On the impact of uml analysis models on source-code comprehensibility and modifiability. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(2):13, 2014.
38. G. Scanniello, C. Gravino, M. Risi, and G. Tortora. A controlled experiment for assessing the contribution of design pattern documentation on software maintenance. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, page 52. ACM, 2010.
39. B. Selic. The pragmatics of model-driven development. *Software, IEEE*, 20(5):19–25, 2003.
40. J. C. Spohrer and E. Soloway. Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, 29(7):624–632, 1986.
41. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer, 2012.
42. M. V. Zelkowitz. An update to experimental models for validating computer technology. *Journal of Systems and Software*, 82(3):373–376, 2009.