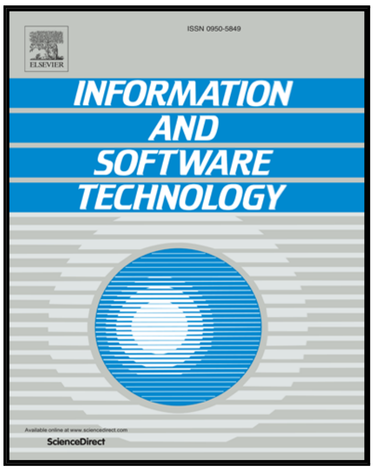


Impact of model notations on the productivity of domain modelling: an empirical study

Cristina Cachero, Santiago Meliá, Jesús M. Hermida

PII: S0950-5849(18)30251-9
DOI: https://doi.org/10.1016/j.infsof.2018.12.005
Reference: INFSOF 6082

To appear in: *Information and Software Technology*

Received date: 21 May 2018
Revised date: 7 December 2018
Accepted date: 10 December 2018

Impact of model notations on the productivity of domain modelling: an empirical study

Cristina Cachero*, Santiago Meliá, Jesús M. Hermida[1]

Departamento de Lenguajes y Sistemas Informáticos. Universidad de Alicante. Spain

Abstract

Context: The intensive use of models is a cornerstone of the Model-Driven Engineering (MDE) paradigm and its claimed gains in productivity. However, in order to maximize these productivity gains, it is important to adequately select the modeling formalism to be used. Unfortunately, the MDE community still lacks empirical data to support such choice.
Objective: This paper aims at contributing to filling this gap by reporting an empirical study in which two types of domain model notations, graphical vs. textual, are compared regarding their efficiency and effectiveness during the creation of domain models.
Method: A quasi-experiment was designed in which 127 participants were randomly classified in four groups. Then, each group was randomly assigned to a different combination of notation and application. All the participants were students enrolled in the 6th semester of the Computer Engineering degree at the University of Alicante. The statistical procedure applied was a two-factor multivariate analysis of variance (two-way MANOVA).
Results: The data shows a statistically significant effect of notation type on the efficiency and effectiveness of domain modelling activities, independently from the application being modelled.
Conclusions: The joint examination of our results and those of previous studies suggests that, in MDE, different tasks call for different types of notations. Therefore, MDE environments should offer both textual and graphical notations, and assist developers in selecting the most suitable one depending on the task being carried out. In particular, our data suggest that domain model creation tasks are better supported by graphical notations. To augment the validity of the conclusions of this paper, the experiment should be replicated with different subject profiles, notations, domain model sizes, tasks and application types.

*Corresponding author. Tel.:+34 965903400
Email addresses: `ccachero@dlsi.ua.es` (Cristina Cachero), `santi@dlsi.ua.es` (Santiago Meliá), `jhermida@dlsi.ua.es` (Jesús M. Hermida)
[1]Currently working at the European Commission, Joint Research Centre (JRC), Ispra, Italy.

Keywords: MDE, Empirical research, Notations, Quasi-Experiment

1. Introduction

Human abilities to process information are highly sensitive to the external representation in which the information is presented to the senses [1]. One reason for this is that external representations of a problem tend to constrain internal representations, and therefore an inappropriate external representation of a problem may lead to a non-optimal mental model, thus hampering its solution [2]. In fact, minor changes in how information is presented to readers may have a dramatic impact on their performance in comprehension and problem solving [1, 3–5].

The Software Engineering (SE) community, aware of this reality, proposes the use of different modelling languages for different purposes in order to increase developers' productivity. These languages act at different levels of abstraction, and they provide different external representations (language notations), which often integrate graphics with text in order to improve the coding experience [2]. Languages whose notation relies on such integration are often referred to as Visual Programming Languages (VPLs), as opposed to pure Textual Programming Languages (TPLs).

Unfortunately, the existing SE language proposals are largely under- evaluated [6]. Most languages have been introduced based on the best knowledge and intuition of their creators, who have largely based their language design decisions on meta-cognitive beliefs [7], that is, on how they think that the use of advanced user interface technology can facilitate the mental processes involved in software construction. In contrast, the use of theoretical models to back the VPL creation process has been traditionally either ignored or considered non-essential [8], the explicit involvement of different user types in the language construction process has been nearly non-existent [6], and their empirical evaluation is scarce for the moment [6, 9]. Such lack of scientific support introduces a degree of uncertainty regarding the conditions under which a VPL can be used successfully [1], which is translated into practitioners often experiencing some practical difficulties when adopting new languages [6].

This situation introduces even more uncertainty when using modern MDE [10] environments, which heavily rely on the construction and maintenance of different models for different views of the system [11]. The claimed benefits of the paradigm rely on how these modelling languages can harness the accidental complexity of software systems [6]. In this context, it is of paramount importance to provide sound empirical evidence of such benefits and, more concretely, of the gains in software developers' performance attributable to the use of the proposed MDE models and techniques in different contexts of use [12]. Empirical data should be related to the evaluation of the languages at the three language definition levels: semantics, abstract syntax (meta-model) and concrete syntax (notation). This paper focuses on the latter, that is, the notation level. In this level, probably due to the meta-cognitive beliefs among the SE community that have been previously mentioned, most MDE environments largely rely on graphical representations of models as the main notation, with text acting as a secondary notation. This contrasts with the fact that graphical notations have proven to be inferior to textual notations for certain tasks [7, 13]. In order to clarify this apparent contradiction, and give each notation type its place in the MDE arena, ascertaining *when and how graphical notations -as opposed to its textual counterparts- can be beneficially used within the context of the complex cognitive activities required by software development?* [2] is as relevant today as it was twenty years ago.

This paper addresses this research question and presents an empirical study that compares the performance (efficiency and effectiveness) of junior software developers while performing a typical domain model development task with two different notations (concrete syntaxes), one graphical and one textual, included in the OOH4RIA MDE methodology [14].

The paper is organized as follows: Section 2 introduces the state of the art regarding the impact of domain modelling notations on domain modelling activities. Section 3 briefly describes the theoretical model that has guided the formulation of the hypotheses for our study. Section 4 presents the study design (objectives, research questions, variables, hypotheses and sensitivity analysis) and its execution. Section 5 describes the data analysis on the efficiency and effectiveness of the notations during domain modelling activities and the threats to its validity. Finally, Section 6 discusses the main results of the study and draws some conclusions and future lines of research.

2. Related Work

Graphical notations differ from textual ones in the way the information is encoded within the language and subsequently processed by the human mind. While textual languages encode information using sequences of characters (uni-dimensional or linear), graphical languages use spatial arrangements together with textual elements (bi-dimensional or spatial) [5]. As a consequence, textual representations can only be processed sequentially, while visual representations allow for a parallel processing by the visual system [5].

Existing research has discussed the relative strengths and weaknesses of both types of notations based on two competing arguments. The research community assumes that diagrams are easy for humans to be understood because their spatial arrangement allows for a more efficient information processing than the linear order of text [2, 15]. Some authors explain this fact by the shift that diagrams allegedly make of some of the processing burden from the cognitive system to the perceptual system, which is faster and frees up scarce cognitive resources for other tasks [5]. However, the symbols of a graphical notation are, in general, harder to comprehend than the textual ones. Therefore, readers of graphical notations must be trained in order to achieve the claimed performance gains provided by the use of such notation. [1, 13, 16].

Other differences between textual and graphical notations compiled from the existing literature are the following:

- Graphical notations can facilitate the comprehensibility of the problem [3, 17] since they usually facilitate the capture and representation of the mental model of the system structure [18]. Their higher level of abstraction may make the programming task easier even for professional programmers [19].
- Graphical notations may contribute to avoiding errors during the modelling activities, also due to their support to spatial reasoning [20].
- Graphical notations allow for a richer secondary notation [2]. Closeness of elements, colors or symbols may supply extra information over and above the information explicitly represented by the model [17]. This proves specially important for expert developers [3].
- Graphical notations help recognition, at least of individual elements (easier to read and discriminate) [17], allowing for a simpler location and access of the desired information.
- Graphical notations are less accurate than textual ones, since a lot of information is embedded in the symbols of the language, which could result in some ambiguity in the interpretation of the reader [3].
- Textual notations are more efficient in spatial terms (more information in the same space) [21]. Otherwise stated, graphical notations offer a relatively low screen density compared to textual notations (Deutch limit) [2].
- Textual notations are more platform- and tool- independent than their graphical counterparts [21] and allow for an easier consistency check.
- Graphical notations are more sensitive to bad layouts (textual pretty printers are much more effective than graphical ones [3])

Table 1 summarizes the main benefits (+) of each notation type commented above.

Table 1: Benefits of textual and graphical notations in software modelling

Textual notation	Graphical notation
(+) Accuracy	(+) Problem comprehensibility (better representation of mental models)
(+) Spatial efficiency / Higher screen density	(+) Spatial reasoning
(+) Platform and tool independency	(+) Information accessing (easier to read/discriminate individual elements)
(+) Consistency check	(+) Richer secondary notation
(+) Lower training required	
(+) Better layout tool support	

It is important to note how this compilation of results is largely focused on differences between textual and graphical languages that are operating at different levels of abstraction (e.g. comparison between General Purpose Languages

(GPLs) and Domain Specific Languages (DSLs) [22]). Oftentimes, it is this different level of abstraction, and not only the graphical or textual character of the language, what may help to explain some apparently contradicting claims in the literature. For example, accuracy is considered to be a potential hindrance of graphical languages [3]. However, in our experience, it may become an issue only if the textual and graphical languages operate at different levels of abstraction, in which case accuracy will usually favor the notation that operates at the lower level. In the same line of thought, efficiency in spatial terms seems to favor graphical notations [15] only if its level of abstraction is higher, while, if both languages operate at the same level of abstraction (for example, when both notations rely on the same underlying meta-model), textual notations make a better use of space [21].

Another important conclusion that has been largely overlooked by the MDE community is that cognitive effectiveness is not an intrinsic property of graphical representations, but something that must be designed into them [6, 15]. Unfortunately, such design (see e.g. [6, 23]) is the exception rather than the rule in MDE notations included in MDE environments. Next, the state of the art of MDE domain modelling notations, which are the object of our study, is presented.

2.1. Domain Modelling Textual vs Graphical Notations in MDE

In MDE, both textual and graphical notations can be used to define different models representing different views of the system under development. Among these views, this experimental study centers on the domain model, which is a core element for the representation of the business logic and the persistence layer of the software applications.

Domain modelling notations in MDE follow two main trends. On the one hand, graphical notations define all the elements in terms of nodes (i.e., domain entities) and links among the nodes (i.e., relationships among the entities), clearly influenced by the Unified Modelling Language (UML) and Entity-Relationship (ER) model, two *de-facto* standards. On the other hand, regarding the textual representation of domain models, the Object Management Group (OMG) has defined the Human-Usable Textual Notation (HUTN) as a Model-Driven Architecture (MDA) standard for general textual modelling [24]. However, there is no standard for the specific representation of domain models. Table 2 shows examples of MDE solutions, together with the languages, be them GPLs or DSLs [22], that they propose to model the application domain.

Table 2: Excerpt of MDE approaches and support for domain modelling notations

Approach	Textual notation	Graphical notation
OOH4RIA	Propietary DSL	UML
OO-Method	-	UML
RadarC	-	UML
WebML	-	UML, ER
UMPLE	Propietary DSL	UML

2.2. Empirical research comparing textual and graphical notations

To the best of our knowledge, there are few empirical studies that compare textual and graphical notations at a similar level of abstraction. In Software Engineering, abstraction is a cognitive means by which, in the process of solving a problem, developers convert the original problem into a simpler one (i.e., more abstract) [25, 26]. In this process of conversion, developers can focus their efforts on specific concerns and address them at a certain detail level. The levels of abstraction and detail of a model are related and constrained by the notation (language) used in the definition. While the level of abstraction refers to the structure of the knowledge acquired and represented in the model (i.e., more general or specific), the level of detail refers to the number of elements (and their properties) used in the representation. Detailed models contain most of the elements and interactions thought to exist in the system being modelled [27].

As part of a broader study, Whitley [2] compiled the empirical evidence up to 1996 regarding the use of VPLs vs. pure TPLs. She concluded that graphical notations can provide better organization and can make information explicit, which favors its performance (notably, its comprehension), more so as the size or complexity of the problem grows. She also claimed that notations are not probably superior in an absolute sense: rather, they are good in relation to specific tasks (match-mismatch and cognitive-fit hypotheses) and individual differences, such as experience [17]. In [28], professional software engineers were asked to validate models presented in three formats: a semi-structured language, a diagrammatic notation and a fully structured textual notation. The results indicated that the semi-structured natural language resulted in the highest efficiency and effectiveness. Also closely related to the study presented in this paper, in [29] it was empirically assessed how the use of a textual vs. a graphical domain-model notation has an impact on the maintainability of the models, with the textual notation outperforming its graphical counterpart.

Some authors have performed such textual vs graphical comparison based on theoretical arguments instead of empirical data. As an example, in [30] the authors informally compare three object modelling notations: Alloy (graphical), UML (graphical) and Z (textual) regarding several parameters such as expressiveness, treatment of functions, contexts, classes and types, and relationships.

There is also a line of research whose focus is on the comparison between different graphical notations (see e.g. [31], [32]). Notably, in [33], the authors extended a DSL and compared the original version against the extended one. They showed how the extended language allowed users to work faster without their effectiveness being hampered.

Another related area with much ampler research results is the the comparison of DSLs *vs.* GPLs (usually defined at different levels of abstraction) for different tasks. Whitley [2] compiles a good summary of of such studies up to 1996. Scanniello *et al.* [34] also provide a good synthesis of a set of studies that compare the effects of using UML domain models produced in the requirements analysis phase against working directly with source code for the comprehensibility and modifiability of such code. A meta-analysis performed on such studies concludes that UML models do not influence neither the comprehensibility of the source code nor its modifiability. In [35] the authors compare the Pheasant DSL vs C++ for query construction. They conclude that Pheasant is more effective, more efficient and generates more satisfaction among its users. In [36] it is shown how the use of models (vs. direct coding) has a positive impact on the maintainability of software applications. Also, in [37] the authors present a family of experiments that show how users achieved higher efficiency and effectiveness in comprehension tasks when using a DSL. Furthermore, in [9] it is proven that these results are independent from the use of an Integrated Development Environment (IDE). Another good example of this line of research is presented in [38], where a family of experiments show that users solve program comprehension tasks more accurately and efficiently with a DSL than with a GPL. Some of these studies have gone one step further and have tried to explain the empirically found improvements in efficiency and effectiveness of program comprehension using DSLs vs using GPLs by better values in a subset of cognitive dimensions [39]. These studies show how the most influential dimensions for differences between DSLs and GPLs are closeness of mappings, diffuseness, error-proneness, role expressiveness and hard mental operations, all of them favoring the use of DSLs over GPLs [40]. Given the importance of such cognitive dimensions as potential explaining factors for the empirical results of MDE experiments, their main characteristics are presented next.

3. Theoretical Model: The Cognitive Framework for Notations

In MDE, decisions on which type of notation to use for a given task in a given context should be based on objective criteria [16]. Given the myriad of notations and the impossibility to compare them one to one for every possible usage context, it is necessary to link empirical results to more theoretical arguments that may help explain and generalize them beyond the particular notations being tested. To this purpose, many characteristics of the notations with potential explaining power have been proposed, including cognitive dimensions [39, 41–45], expressive power [46], formal analysis capabilities [47], terseness [48], cognitive load [49], aesthetics [50], perceptual characteristics of the symbol set [5, 51] and usability [4], to name a few. From them, the Cognitive Dimensions Framework (CDF) [39] and the Physics of Notations Framework (PNF) [5] stand out. The PNF centers on decoding efficiency (comprehensibility) and is only applicable to graphical notations [52]. On the contrary, the CDF centers on encoding efficiency (productivity) and it can be applied to both graphical and textual notations in either an interactive (computer-based, which may include graphical packages, word-processors and IDEs) or a non-interactive (static representations on paper, on a board, etc.) context [52, 53]. For these two reasons, in this paper the CNF has been chosen to characterize the languages being compared.

The CDF adopts a cognitive effectiveness viewpoint, which can be defined as the *speed, ease and accuracy with which a representation can be processed by the human mind* [15]. It defines a set of constructs that is listed in Table 3. It is expected that different values of these constructs for different notations may at least partly explain differences in the language usability, defined as *the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use* [54]. The usability measurement of modeling languages can be performed informally (see e.g. [44]) or more formally, e.g. through the use of questionnaires (see e.g. [55]) or even a whole quality assessment framework [56]. Also, it is important to note how the impact of the values of these dimensions may vary depending on the particular task being performed [44] or the individual characteristics of the developer[57]. Last but not least, some of the characteristics of the language may be mitigated by design manoeuvres, either isolated or part of an IDE [44].

Based on the definition of these dimensions, it seems only natural to assume that not all of them are equally impacted by the notation being used: abstraction, diffuseness, premature commitment, closeness of mapping, provisionality and role expresiveness seem to be more related to the abstract syntax of the language (*Low* influence of notation, see Table 3), while hidden dependencies, secondary notation, viscosity, visibility, consistency, error proneness, hard mental operations and progressive evaluation may be more directly affected by the notation (*High* influence of notation, see Table 3).

4. Experimental design

In the context of SE, a controlled experiment can be defined as *a randomised experiment or quasi-experiment in which individuals or teams (the experimental units) conduct one or more SE tasks for the sake of comparing different populations, processes, methods, techniques, languages or tools (the treatments)* [58]. In March 2016, a quasi-experiment was conducted at the University of Alicante. A quasi-experiment is a type of controlled experiment in which the subjects are not selected randomly. In this way, it is possible to study cause-effect relationships in scenarios like ours, in which the cost of the random selection of subjects, i.e. software developers, is very high [59].

4.1. Objectives and context definition

Following the structure of the Goal-Question-Metric (GQM) template [60], the purpose of this study was to assess the effect (efficiency and effectiveness) of the use of a textual vs. the use of a graphical notation for the representation of domain models from the point of view of junior software developers individually modelling a software system. The context of the study is that of a 6th semester undergraduate student enrolled in the Computer Engineering degree at the University of Alicante. The design of the experiment is based on the experimentation framework proposed by Wohlin *et al.* [61].

The two chosen notations are part of the OOH4RIA MDE proposal [14]. OOH4RIA proposes a complete development process based on a set of models and transformations that allow to go from conceptual models to code. Specifically, the central model is the domain model that allows to represent different

Dimension	Description	Influence
Abstraction	types and availability of abstraction mechanisms in the language	Low
Hidden dependencies	Important links between entities are not visible	High
Secondary notation	Extra information is provided in means other than form syntax	High
Diffuseness	Verbosity of language	Low
Premature commitment	Constraints on the order of doing things	Low
Viscosity	Resistance to change	High
Visibility	Ability to view components easily	High
Closeness of mapping	Closeness of representation to domain	Low
Consistency	Similar semantics are expressed in similar syntactic forms	High
Error proneness	Notation invites mistakes	High
Hard mental operations	High demand on cognitive resources	High
Progressive Evaluation	Work-to-date can be checked at any time	High
Provisionality	Degree of commitment to actions or marks	Low
Role expressiveness	Purpose of a component is readily inferred	Low

Table 3: CDF Dimensions and influence of notation

entities and their relations of the real world independently of their implementation. From the domain model, the OOH4RIA proposal defines a set of model-to-text transformations to obtain the business logic and the persistence layers of software applications. In this experiment, the reason for selecting OOH4RIA is two fold: on the one hand, the MDE community does not favor any specific MDE environment over the other, so, to the best of our knowledge, the selection of one or other is a highly subjective choice. On the other hand, our research question required an experimental environment that provided support for both a graphical and a textual notation for domain models that were informationally equivalent [1]. The OOH4RIA approach fulfills these requirements and provides, for the definition of domain models, two possible notations. The first is a graphical notation that turns the language into a VPL. The second is a pure textual notation that turns the language into a pure TPL. With this selection it has been ensured that, in the study, the abstract syntax and the semantics of the DSL remain fixed, so that, if differences in performance are found, they can be attributed to the concrete syntax (notation) used. The hypothesis in this sense is, based on the related literature and the CDF, that such differences exist.

The research questions (RQ) addressed in this study were designed to be answered using quantitative data. The questions are the following:

- RQ1: Is the *productivity* of the software developers, understood as a linear composite of effectiveness and efficiency, affected by the use of a graphical or a textual notation during the definition of the domain model of a software application?
- RQ2: Is the *effectiveness* of the software developers affected by the use of a graphical or a textual notation during the definition of the domain model of a software application?
- RQ3: Is the *efficiency* of the software developers affected by the use of a graphical or a textual notation during the definition of the domain model of a software application?

4.2. Design of the experiment

In this study, it was planned to gather data from 134 subjects, who were 6th semester students of the Computer Engineering degree at the University of Alicante. The subjects were randomly assigned to two factors: Notation (graphical or textual) and System (Ticket Seller or Hotel Manager). As mentioned before, both types of notation share the same abstract syntax and semantics, while differing in their concrete syntax or notation. It should be noted that the experiment was designed to be run in a class session. During that session, which was scheduled 2 hours, the students had to be able to finish the domain model proposed. For this reason, the complexity of the systems had to be adjusted to that time restriction. The final complexity of the two proposed systems was as follows:

- The TicketSeller system has 8 classes, 24 attributes, 11 operations and 10 relationships.
- The HotelManager system has 6 classes, 25 attributes, 6 operations and 7 relationships.

Variables

To conduct the experiment, two independent variables (IV) or factors were defined, each with two levels/values:

- Notation (Not): categorical variable, inter-subject, with two possible values: textual and graphical. Not was defined as our focal variable, *i.e.*, the independent variable of primary interest.
- System (Sys): categorical variable, inter-subject, with two possible values: TicketSeller or HotelManager. Sys was included as a moderator variable, *i.e.*, a variable which can further define/refine the relationship between the focal variable and the dependent variables.

Their combination resulted in four different treatments (corresponding to the four possible value combinations of Not and Sys).

The set of dependent, or measurable, variables (DV) were defined as follows:

- PAtr: percentage of attributes correctly defined, out of the total number of attributes in the model [0..100]
- POp: percentage of operations correctly defined, out of the total number of operations in the model [0..100]
- PRel: percentage of relationships correctly defined, out of the total number of relationships in the model [0..100]
- PCard: percentage of cardinality constraints correctly defined, out of the total number of cardinality constraints in the model [0..100]
- T: Time needed to complete the domain model (in seconds).

Finally, given the fact that all the DV are related to the productivity concept, the Productivity construct has been defined as a linear composite of the PAtr, POp, PRel, PCard and T DV.

Hypotheses

Based on the literature review presented in Section 2, and the research questions and the variables presented above, the following null hypotheses (and the corresponding alternative hypotheses) were defined:

- HProductivity0: The use of the graphical or textual notation of OOH4RIA for domain modelling does not affect the productivity of the software developers (i.e., the percentage of elements correctly identified and the development time), independently from the system being developed.
- HEffectivenessAttr0: The use of the graphical or textual notation of the approach OOH4RIA for domain modelling does not affect the percentage of attributes correctly detected, independently from the system being developed.
- HEffectivenessOp0: The use of the graphical or textual notation of the approach OOH4RIA for domain modelling does not affect the percentage of operations correctly detected, independently from the system being developed.
- HEffectivenessRel0: The use of the graphical or textual notation of the approach OOH4RIA for domain modelling does not affect the percentage of relationships correctly detected, independently from the system being developed.
- HEffectivenessCard0: The use of the graphical or textual notation of OOH4RIA for domain modelling does not affect the percentage of cardinality constraints correctly defined, independently from the system being developed.

- HEfficiency0: The use of the graphical or textual notation of OOH4RIA for domain modelling does not affect the time needed to complete a model, independently from the system being developed.

Sensitivity analysis

Finally, in order to validate our experiment design, a sensitivity analysis of the main univariate effects was performed, both for Not and Sys. Given that our subject number is fixed (limited by the number of students enrolled in the course) and cannot be modified, this analysis was used to determine the size of the effect that can be detected with our design, for the variables Not, Sys and their interaction. Using ANOVA (ANalysis Of VAriance), the size of the effect is measured using the eta-squared statistic (η^2). In the three cases, the result was 0.25 (average effect size [62]) with an analysis power (1-β) of 0.8 [62]. This means that our design, besides limiting to α=0.05 the probability of Type I errors (rejecting the null hypothesis when it was actually True), controls the risk of Type II errors (i.e., risk of accepting the null hypothesis when it should be rejected) to 0.2 if the effect size is at least 0.25.

4.3. Experiment Execution

This quasi-experiment was conducted during a two-hour session of the Software Design course. The subjects had been previously trained in the use of both notations (graphical and textual) during two hours each. In the MDE field, IDEs are normally used for modelling. However, as presented in Table 2, few MDE proposals support both a textual and a graphical notation for the same abstract syntax of the domain model. From the ones that do support both (such as is the case of the OOH4RIA editor), the aids provided by the textual and graphical editors (syntax highlighting, auto-complete functions, continuous validation, etc.) differed. Therefore, it was decided to ask the subjects to create the domain models on paper, in order to control for this potentially confounding factor.

During the session, the subjects did not receive any feedback on their performance. Furthermore, two lecturers supervised the experiment in order to avoid any possible interaction among the subjects. For ethical reasons, subjects were explicitly asked for permission to treat their anonymised and aggregated data in this experiment. 127 subjects (out of 134) accepted. Table 4 shows the final distribution of the subjects by treatment.

Table 4: Final distribution of subjects by treatment

	Textual	Graphical	Total
Ticket Seller	30	33	63
Hotel Manager	32	32	64
Total	62	65	127

All the measures were calculated based on the manual evaluation of the tasks, carried out by the three authors of this paper. Before the evaluation, in order to avoid misalignments among evaluators, a set of evaluation templates were elaborated using Google Forms [2] [3]. Subsequently, the templates were refined by collaboratively evaluating five assignments for each system. Finally, the templates were validated using five more assignments, which were evaluated independently by the three authors. The specificity of the criteria achieved during the first stage of the template evaluation made possible to reach a 100% level of agreement for this second set. This result allowed us to divide the evaluation of the remaining assignments among the three evaluators. The complete versions of the models (textual and graphical) used in this experiment are available as part of the experimental package [4].

5. Data Analysis and Results

To analyse the data, the software SPSS Statistics v.23 was used. Table 5 shows the descriptive statistics corresponding to the measures used in this study.

Table 5: Descriptive statistics for the five measures associated with the dependent variables. (Avg: average; SD: standard deviation)

DV	**Textual**		**Graphical**		**Ticket Seller**		**Hotel Manager**	
	Avg	SD	Avg	SD	Avg	SD	Avg	SD
PAtr	76.47	16.34	84.77	9.90	81.82	11.68	79.64	17.28
POp	44.03	29.70	58.77	26.99	42.86	28.50	60.16	28.26
PRel	39.49	29.77	71.69	16.72	55.24	26.39	56.70	31.81
PCard	21.70	25.31	42.14	29.28	33.42	28.49	30.92	29.98
T	3701	798	2964	974	3123	931	3520	960

5.1. Test selection, assumptions, interaction and main effects

To perform the statistical data analysis, a two-factor Multivariate ANalysis Of VAriance (two-way MANOVA) was applied [63] (α=0.05) with two independent variables – Not and Sys – and five dependent variables – PAttr, POp, PRel, PCard and T –. The two-way MANOVA is an extension of the two-way ANOVA in which two or more dependent variables are combined to measure something (in our case, the overall productivity of the analysts in the creation of domain models).

Before the analysis, all the assumptions that ensure the applicability of the MANOVA in our study were verified. A scatterplot of the dependent variables by group detected absence of linearity. Although the violation of this assumption does not prevent the application of the test, it does reduce its power. The Pearson coefficient showed no evidence of multicolinearity ($|r| < 0.9$). After the inspection of the boxplots, several univariate outliers and two extreme points were detected. The review of the associated data tuples showed that they were not errors in the data input or in the measure, but genuinely exceptional values.

[2]Hotel Manager Evaluation form: https://goo.gl/PR67uV
[3]Ticket Seller Evaluation Form: https://goo.gl/9E831d
[4]http://mde.dlsi.ua.es/oohria/labPackages/ExpPackageIST2018.zip

Therefore, they were kept in the analyses. With the calculation of the Mahalanobis distance, the existence of multivariate outliers was discarded. Violations of the normal distribution of the scale in the values of the dependent variables PAttr, POp and PRel in several groups were detected. In order to correct this aspect, given that all the variables showed a light negative asymmetry, a *reflect and square root* transformation was performed, consisting in adding one to the scale maximum, substracting the value of each variable and finally calculating the square root [64]. This transformation substantially improved the distribution of the three variables, which passed the normality test (verified by means of the z-score for asymmetry and kurtosis, in the range ± 2.58 for all the cases) even though with no linearity. In addition, the verification of the compliance of the homogeneity of the covariance matrices was done by means of the Box's M test ($\rho > 0.001$), and the homogeneity of the variances was checked using the Levene's variance homogeneity test ($\rho > 0.05$).

The interaction effect between Not and Sys in the combined variables was not significant – $F(5, 119) = 0.403$, $\rho = .846$, Wilks' $\Lambda = .983$, partial $\eta^2 = .017$–. However, the analyses detected a significant main effect in both Not – F(5, 119) = 15.030, $\rho < 0.001$, Wilks' $\Lambda = .613$, partial $\eta^2 = .387$ – and Sys – F(5, 119) = 4.245, $\rho = 0.001$, Wilks' $\Lambda = .849$, partial $\eta^2 = .151$ – over the combined DVs. Based on these results, the first hypothesis, HProductivity0, can be rejected, and thus affirm that *the model notation in OOH4RIA globally affects the efficiency and effectiveness of the developers.* Subsequently, a two-way ANOVA univariate analysis was carried out in order to study the univariate effects of the variables Not and Sys over each dependent variables separately.

5.2. Univariate Effect of Notation and System

The performed analysis detects a statistically significant main effect of the Not over all the dependent variables. In addition, it detects a statistically significant main effect of Sys over POp and T. Such effects are shown in Table 6.

Table 6: Univariate effects in Notation and System (two-way ANOVA)

IV	DV	F(1,123)	p	partial n2
Not	PAttr	8.891	**0.003**	0.067
	POp	6.204	**0.014**	0.048
	PRel	43.73	<**0.001**	0.262
	PCard	15.475	<**0.001**	0.112
	T	22.363	<**0.001**	0.154
Syst	PAttr	0.283	0.596	0.002
	POp	14.029	<**0.001**	0.102
	PRel	0.956	0.330	0.008
	PCard	0.236	0.628	0.002
	T	6.525	**0.012**	0.050

Based on these results, the hypotheses related to the RQ2 can be rejected, concerning effectiveness, and affirm that *the percentage of attributes, operations, relationships and cardinality constraints correctly defined is affected in*

OOH4RIA by the use of a specific notation. The system modelled in the exercise (Hotel Manager or Ticket Seller) also affects POp and T measures, suggesting a difference in complexity among both systems despite its similar size. In particular, efficiency is significantly lower for the Hotel Manager system, which, counterintutively, is slightly smaller than the Ticket Seller, indicating that model size may be misleading as a complexity measure. Last, there is no interaction in the effects, which implies that the direction of the effect is homogeneous and, in our case, favors the graphical notation.

Figures 1a, 1b, 1c and 1d illustrate the results in a set of charts. In the figures, the proximity of the lines corresponding to each system indicates that the Sys variable does not affect the results. The influence of the Not variable can be appreciated in the slope of the lines. Finally, the slope of both lines (similar direction and inclination) shows in a visual manner that there is no interaction of the variables Not*Sys. The same visual indications are applicable to Figure 1e, which represents the average time used by the subjects to model each system with each notation.

Finally, Table 6 also provides the information required to answer RQ3 and to reject the hypothesis HEfficiency0. Our analysis shows how *subjects are significantly more efficient when they use the graphical notation (* vs. *using the textual one)* as well as when they model the TicketSeller system (*vs.* modelling the HotelManager one).

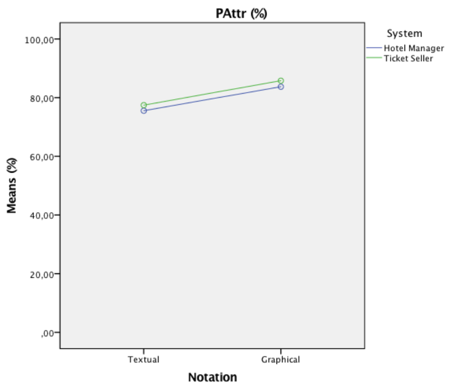
The results show how the two measures most notably affected by the notation used are the definition of model relationships and model cardinalities; for these two domain modelling subtasks, developers are significantly less efficient and effective when using the textual notation.

Next a tentative explanation of these facts according to the CDF introduced in Section 3 is presented.

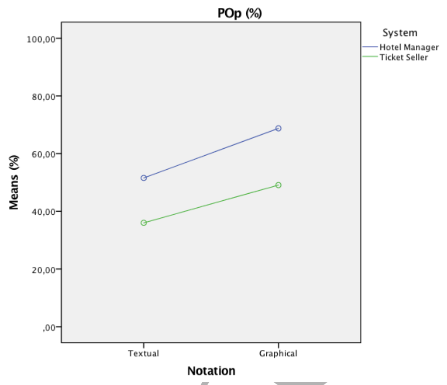
5.3. CDF Analysis of the Notations of the Empirical Study

Section 3 explained how the CDF can be used to come up with plausible explanations for differences in productivity of different notations [52]. However, in this study, not all the dimensions of the CDs framework are equally relevant. Since the study compares the notations (while the abstract syntax of the language is constant), the values for the dimensions of abstraction, closeness of mapping, diffuseness, premature commitment, consistency, provisionality and role expressiveness remain constant. The other dimensions, i.e., hidden dependencies, secondary notation, viscosity, visibility, error proneness, hard mental operations and progressive evaluation, of the two OOH4RIA notations (for the task of domain model creation) were evaluated following the procedure defined in [65]. Table 7 presents the main results of such evaluation.

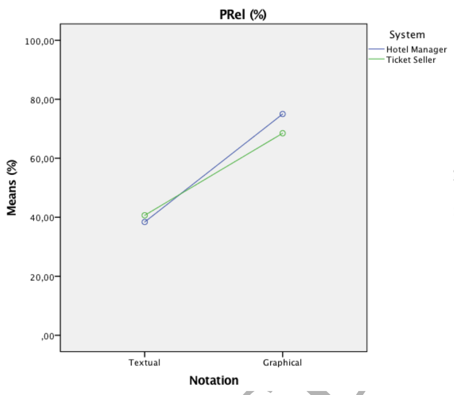
As it can be seen in Table 7, for the domain model creation task, the textual and graphical notations of OOH4RIA differ in some important dimensions. Regarding the *Hidden dependencies* dimension, all the relationships of the domain are explicitly represented in the graphical notation (low value). In the case of the textual notation, some relationships are implicit due to how the inheritance relationship between classes/concepts is expressed in the notation (medium value).



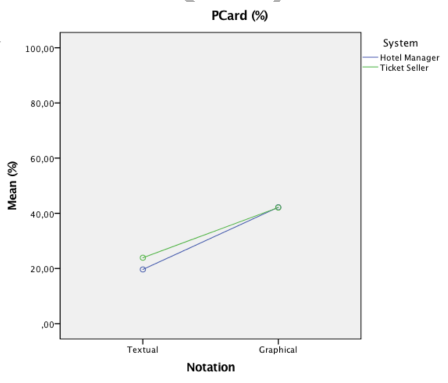
(a) PAttr



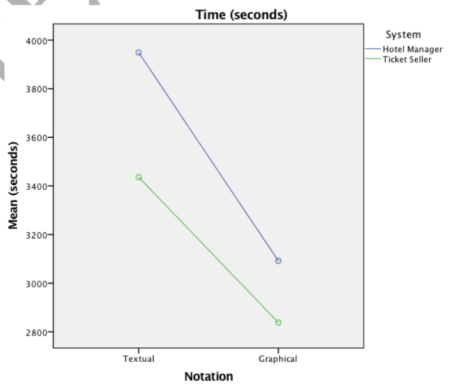
(b) POp



(c) PRel



(d) PCard



(e) PT

Figure 1: Univariate effects of Notation and System over PAtr, POp, PRel,PCard and T

Dimension	OOH4RIA Textual	OOH4RIA Graphical
Hidden dependencies	**Medium**	**Low**
Secondary notation	**Low**	**High**
Viscosity	Medium	Medium
Visibility	**Medium**	**High**
Error proneness	**Medium**	**Low**
Hard mental operations	**High**	**Low**
Progressive Evaluation	High	High

Table 7: Cognitive dimensions of the OOH4RIA Notations for the Creation of Domain Models

Therefore, with the textual language, readers must keep in mind some relationships between classes that are not present in the visible segment of text, which may lead to misinterpretations of the model.

Regarding the *Secondary notation* dimension, the OOH4RIA graphical notation does not represent all the knowledge captured by the model, but requires an auxiliary notation (textual, form-based) to represent some properties of the classes, attributes, operations and constraints (high value). Instead, all the elements and properties are represented in the textual notation (low value). Therefore, regarding this dimension, the graphical notation is at a disadvantage since, to understand a basic piece of knowledge (e.g., a constraint), developers may need to read two different notations. This can be a problem specially in medium/large models.

In the *Visibility* dimension, the graphical notation (high value) captures and presents more knowledge to the readers than the textual notation (medium value) in the same representation space, i.e., it provides a higher knowledge density. For small/medium models, this can be an advantage for readers because they can have a complete view of the model (and the problem) with less effort. However, understanding large models requires more effort independently from the notation. In this case, readers can have an advantage using the textual notation, since it can better split the represented knowledge. Regarding the *Error proneness* dimension, the graphical notation is not considered as error prone (low value) since it is an extension of the notation for UML class diagrams (well-known in the domain of Software Engineering). Such familiarity of the developers with the the UML class diagram should reduce the learning curve of this notation, as well as facilitate its use. Instead, the textual notation is specific for the OOH4RIA domain model. Although it is a simple notation and there are several high-level programming languages that use similar structures, the fact that developers are not as familiarized with this specific notation may cause some problems with its use, in particular with the definition of contexts and relationships. Regarding the *Hard mental operations* dimension, the models using both notations require some effort from the readers, especially when understanding the inheritance relationships between classes and its implications. In particular, the textual notation (high value) requires a general effort to understand the logic of the relationships between classes and keep it in mind. This is due to the fact that relationships are defined independently from the classes and may not even be represented in a text area nearby. On the other hand, in the graphical notation (low value), with medium and small systems, the classes and their relationships are represented in the same representation space, which reduces the effort to memorize and to comprehend the system under development. Finally, we do not appreciate substantial differences between the two notations regarding Viscosity and Progressive evaluation, despite them being also potentially influenced by notation.

5.4. Threats to the validity of the study

Given the design of the study, the main threats to its validity are external, i.e., they affect the capacity of extrapolating the results to the general population. Our student sample is a limited representation of the population of software developers in MDE environments. However, our need for a broad sample that was able to detect at least an average effect size (see the sensitivity analysis in Section 4) made very difficult the recruitment of such a high number of professionals. This is a common situation in experiments of this type [66]. Also, graphical modelling techniques have been empirically proven to provide better support for expert modellers than for student modellers [67]. Based on these findings, our hypothesis (whose validation requires a replication of this study) is that the positive impact of the graphical notation over our subjects would have been bigger had expert modelers been enrolled instead. Another external threat is the the setting of the experiment (modelling on paper, in the context of a university course), since it does not reflect the conditions of a work environment. As mentioned before, using an IDE, while increasing the external validity, also meant including a potential additional source of variability [44]. For this reason, in the experiment design this variable was removed. The results presented in this paper can therefore be regarded as a baseline against which future data involving the use of the OOH4RIA IDE can be compared and questions such as 'to what extend does the use of a given IDE mitigates/increments the impact of the notation used?' can be answered. In this sense, some related results presented in [9] suggest that such use should not significantly change the direction of the results. Also the complexity of the systems defined is lower than the average complexity of the systems used in the industry, due to time constraints in the experiment (a 2-hour session). Finally, the model notations are the ones included in the OOH4RIA IDE. Although the graphical and textual notations of OOH4RIA are the result of more than 10 years of experience in the development of MDE applications in the industry, they are still two possible notations among many others, which makes impossible to generalise our conclusions to other existing notations.

Regarding the conclusion validity of this study, all the assumptions that enable the use of the chosen statistical test were verified. Moreover, an analysis of the sensibility that ensures a power of 0.8 for the main effects with a medium effect size (0.25) was performed.

The main threats to the construct validity were mitigated by avoiding the discussion of the hypothesis of the experiment beforehand and by using more than one system.

Finally, the threats to the internal validity of the study were mitigated by asking each subject to develop one single system (thus avoiding learning effects, lack of time and maturity) and controlling the interaction among subjects during the experiment. Another internal threat is the possible different level of familiarity of students with the two assigned systems. This risk has been mitigated by choosing two systems that are commonly used as examples in SE courses (and therefore assumed to be equally familiar for the students). Also, students were randomly assigned to each system and notation.

6. Discussion, conclusions and future work

This paper has compared the impact of two modelling notations, a graphical and a textual one, both part of the OOH4RIA environment, regarding the efficiency and effectiveness of junior software developers for the task of domain model creation. To this aim, several measures have been defined over the number of attributes, operations, relationships and cardinality constraints that developers need to define in domain models, as well as over the time needed to develop the models.

The data gathered in the current study demonstrates that it is statistically significant the improvement of a graphical notation regarding to the textual notation, for the development tasks of small-sized domain models with juniors software developers. The sensitivity analysis carried out in the study also qualifies the non-significant results. In terms of the theory of equivalence of representations [1], this means that the graphical and textual notations of the OOH4RIA approach are semantically equivalent (that is, the same information can be conveyed from both representations of the system) but computationally unequivalent (creating graphical domains model requires less computation than creating textual ones).

The results of efficiency (errors with the textual notation against errors with the graphical one), together with the types of errors that were more affected by the notation (relationships and their cardinalities), are a particularly relevant contribution of this paper. The vast majority of subjects was able to finish the assignment within the 2-hour time boundary, which allows us to assume that the lack of time did not influence the results. In addition, the models developed during the experiment are relatively simple, so that the lower number of attributes, operations, relationships and cardinality constraints detected correctly cannot be due to the tiredness of the subjects, which a more complex system could have produced.

It is important to note how these results contrast with the ones obtained in a previous study on maintainability of MDE models using either textual or graphical notations [29], in which the textual notation improved both the efficiency and the effectiveness of the subjects during certain tasks of analysis and modification of domain models. Such contrasting findings (textual notations being superior for model maintainability tasks and graphical notations being superior for model creation tasks) corroborate the claim made by several authors that the cognitive load of the different languages may impact differently depending on the particular task [2, 44].

In light of the CDF dimension values assigned to both the textual and graphical notations (see Section 3), it can be inferred that these dimensions do not affect in the same way domain model maintenance and domain model creation tasks. The OOH4RIA graphical notation shows less hidden dependencies, higher visibility, lower error proneness and easier mental operations, which seem to be important for developers' productivity. However, for domain model maintenance tasks, such graphical advantages are shadowed by the need for a secondary notation. Our hypothesis in this sense (which will need to be further examined in future experiments) is that spatial reasoning, while very important for model creation, plays a secondary role in model maintenance. Given the fact that both are important cognitive activities that need to be supported by modelling languages [6], it seems only sensible for the MDE community to provide support for both notations in their MDE environments. Unfortunately, this is not the common practice in the discipline, as it was shown in Table 2.

The main threats to the validity of this study are 1) the fact that the experiment was conducted with students and with no IDE (on paper) and 2) the size of the models used, relatively small. From them, the use of novice modellers is the most worrying, since existing literature points at the idea that novices may be specially sensitive to the visual form of notations used [68]. Therefore, our next step will be to replicate this experiment using the OOH4RIA IDE, with more complex models and more experienced developers. Another line of work is the replication of this study with other examples of graphical and textual notations (other DSLs), ideally differing in other cognitive dimensions in order to better understand how they influence the performance of developers.

Acknowledgements

This study was partially funded by the Spanish Ministry of Economy, Industry and Competitiveness under the project TIN2016-78103-C2-2-R "Plataforma para la publicación y consumo de datos abiertos para una ciudad inteligente". Special thanks to the subjects of the experiment, who kindly agreed to participate in our study. Also, many thanks to the reviewers who, with their suggestions, greatly contributed to improving the quality of this paper.

References

[1] K. Siau, Informational and computational equivalence in comparing information modeling methods, Journal of Database Management 15 (1) (2004) 73.

[2] K. N. Whitley, Visual programming languages and the empirical evidence for and against, Journal of Visual Languages & Computing 8 (1) (1997) 109–142.

[3] M. Petre, Why looking isn't always seeing: readership skills and graphical programming, Communications of the ACM 38 (6) (1995) 33–44.

[4] R. N. Taylor, A. van der Hoek, Software design and architecture, the once and future focus of software engineering, in: Future of Software Engineering, 2007, pp. 226–243. doi:10.1109/FOSE.2007.21.

[5] D. L. Moody, The "physics" of notations: a scientific approach to designing visual notations in software engineering, in: ACM/IEEE International Conference on Software Engineering, Vol. 2, 2010, pp. 485–486. doi:10.1145/1810295.1810442.

[6] A. Barišić, V. Amaral, M. Goulao, Usability driven DSL development with USE-ME, Computer Languages, Systems & Structures 51 (2018) 118–157.

[7] A. F. Blackwell, Metacognitive theories of visual programming: what do we think we are doing?, in: IEEE Symposium on Visual Languages, 1996, pp. 240–246. doi:10.1109/VL.1996.545293.

[8] K. Siau, Y. Wand, I. Benbasat, The relative importance of structural constraints and surface semantics in information modeling, Information Systems 22 (2-3) (1997) 155–170.

[9] T. Kosar, S. Gaberc, J. C. Carver, M. Mernik, Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments, Empirical Software Engineering (2018) 1–30.

[10] A. R. da Silva, Model-driven engineering: A survey supported by the unified conceptual model, Computer Languages, Systems & Structures 43 (2015) 139 – 155. doi:https://doi.org/10.1016/j.cl.2015.06.001.

[11] P. Mohagheghi, W. Gilani, A. Stefanescu, M. Fernandez, An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases, Empirical Software Engineering 18 (1) (2013) 89–116.

[12] J. Hutchinson, J. Whittle, M. Rouncefield, S. Kristoffersen, Empirical assessment of MDE in industry, in: International Conference on Software Engineering, 2011, pp. 471–480. doi:10.1145/1985793.1985858.

[13] T. R. G. Green, M. Petre, When visual programs are harder to read than textual programs, in: European Conference on Cognitive Ergonomics), 1992, pp. 167–180.

[14] S. Meliá, J. Gómez, S. Pérez, O. Díaz, A model-driven development for GWT-based rich internet applications with OOH4RIA, in: International Conference on Web Engineering, 2008, pp. 13–23. doi:10.1109/ICWE.2008.36.

[15] J. H. Larkin, H. A. Simon, Why a diagram is (sometimes) worth ten thousand words, Cognitive science 11 (1) (1987) 65–100.

[16] A. Ottensooser, A. Fekete, H. A. Reijers, J. Mendling, C. Menictas, Making sense of business process descriptions: An experimental comparison of graphical and textual notations, Journal of Systems and Software 85 (3) (2012) 596–606.

[17] T. R. G. Green, M. Petre, R. K. E. Bellamy, Comprehensibility of visual and textual programs: A test of superlativism against the match-mismatch conjecture, in: Empirical Studies of Programmers, 1991, pp. 121–146.

[18] S. M. Kosslyn, J. R. Pomerantz, Imagery, propositions, and the form of internal representations, Cognitive psychology 9 (1) (1977) 52–76.

[19] B. A. Myers, Taxonomies of visual programming and program visualization, Journal of Visual Languages & Computing 1 (1) (1990) 97–123.

[20] J. C. Spohrer, E. Soloway, Novice mistakes: Are the folk wisdoms correct?, Communications of the ACM 29 (7) (1986) 624–632.

[21] H. Krahn, B. Rumpe, S. Völkel, Monticore: Modular development of textual domain specific languages, in: Objects, Components, Models and Patterns, Springer Berlin Heidelberg, 2008, pp. 297–315.

[22] T. Kosar, S. Bohra, M. Mernik, Domain-specific languages: A systematic mapping study, Information and Software Technology 71 (2016) 77 – 91. doi:https://doi.org/10.1016/j.infsof.2015.11.001.

[23] P. Caire, N. Genon, P. Heymans, D. L. Moody, Visual notation design 2.0: Towards user comprehensible requirements engineering notations, in: IEEE International Requirements Engineering Conference, 2013, pp. 115–124. doi:10.1109/RE.2013.6636711.

[24] OMG, UML Human-Usable Textual Notation, Tech. rep., The Object Management Group (2004).
URL `http://www.omg.org/spec/HUTN/1.0/`

[25] O. Hazzan, J. Kramer, Abstraction in computer science & software engineering: A pedagogical perspective, System Design Frontier Journal 4 (1) (2007) 6 – 14.

[26] B. Liskov, J. Guttag, Program Development in Java. Abstraction, Specification and Object-Oriented Design, Addison-Wesley, 2000.

[27] R. Brooks, A. Tobias, Choosing the best model: Level of detail, complexity, and model performance, Mathematical and Computer Modelling 24 (4) (1996) 1 – 14. doi:https://doi.org/10.1016/0895-7177(96)00103-3.

[28] B. Hoisl, S. Sobernig, M. Strembeck, Comparing three notations for defining scenario-based model tests: A controlled experiment, in: Quality of Information and Communications Technology, IEEE, 2014, pp. 180–189.

[29] S. Meliá, C. Cachero, J. M. Hermida, E. Aparicio, Comparison of a textual versus a graphical notation for the maintainability of MDE domain models: an empirical pilot study, Software Quality Journal 24 (3) (2016) 709–735.

[30] D. Jackson, A comparison of object modelling notations: Alloy, UML and Z, Tech. rep., MIT Lab for Computer Science (1999).

[31] D. Birkmeier, S. Kloeckner, S. Overhage, An empirical comparison of the usability of BPMN and UML activity diagrams for business users, in: European Conference on Information Services, 2010, p. 2.
URL `https://aisel.aisnet.org/ecis2010/51`

[32] H. C. Purchase, R. Welland, M. McGill, L. Colpoys, Comprehension of diagram syntax: an empirical study of entity relationship notations, International Journal of Human-Computer Studies 61 (2) (2004) 187–203.

[33] F. Häser, M. Felderer, R. Breu, Is business domain language support beneficial for creating test case specifications: a controlled experiment, Information and software technology 79 (2016) 52–62.

[34] G. Scanniello, C. Gravino, M. Genero, J. Cruz-Lemus, G. Tortora, On the impact of UML analysis models on source-code comprehensibility and modifiability, ACM Transactions on Software Engineering and Methodology 23 (2) (2014) 13.

[35] A. Barišić, V. Amaral, M. Goulão, B. Barroca, Quality in use of domain-specific languages: a case study, in: ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools, ACM, 2011, pp. 65–72.

[36] Y. Martínez, C. Cachero, S. Meliá, Empirical study on the maintainability of web applications: Model-driven engineering vs code-centric, Empirical Software Engineering 19 (6) (2014) 1887–1920.

[37] T. Kosar, M. Mernik, J. C. Carver, Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments, Empirical Software Engineering 17 (3) (2012) 276–304.

[38] A. N. Johanson, W. Hasselbring, Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment, Empirical Software Engineering 22 (4) (2017) 2206–2236.

[39] T. R. G. Green, M. Petre, Usability analysis of visual programming environments: a cognitive dimensions framework, Journal of Visual Languages & Computing 7 (2) (1996) 131–174.

[40] T. Kosar, N. Oliveira, M. Mernik, V. J. M. Pereira, M. Črepinšek, D. Da Cruz, R. P. Henriques, Comparing general-purpose and domain-specific languages: An empirical study, Computer Science and Information Systems 7 (2) (2010) 247–264.

[41] T. R. G. Green, Cognitive dimensions of notations, in: Conference of the Human-Computer Interaction Specialist Group on People and Computers, Cambridge University Press, 1989, pp. 443–460.

[42] F. Modugno, T. R. G. Green, B. A. Myers, Visual programming in a visual domain: a case study of cognitive dimensions, in: Conference of the Human-Computer Interaction Specialist Group on People and Computers, Cambridge University Press, 1994, pp. 91–108.

[43] A. F. Blackwell, C. Britton, A. Cox, T. R. G. Green, C. Gurr, G. Kadoda, M. S. Kutar, M. Loomes, C. L. Nehaniv, M. Petre, C. Roast, C. Roe, A. Wong, R. M. Young, Cognitive dimensions of notations: Design tools for cognitive technology, in: Cognitive Technology: Instruments of Mind, Springer Berlin Heidelberg, 2001, pp. 325–341.

[44] T. R. G. Green, A. E. Blandford, L. Church, C. R. Roast, S. Clarke, Cognitive dimensions: Achievements, new directions, and open questions, Journal of Visual Languages & Computing 17 (4) (2006) 328–365.

[45] A. F. Blackwell, Cognitive dimensions of notations: Understanding the ergonomics of diagram use, in: Diagrammatic Representation and Inference, Springer Berlin Heidelberg, 2008, pp. 5–8.

[46] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, Pattern-based analysis of the control-flow perspective of UML activity diagrams, in: Conceptual Modeling, Springer Berlin Heidelberg, 2005, pp. 63–78.

[47] A. Lauder, S. Kent, Precise visual specification of design patterns, in: European Conference on Object-Oriented Programming, Springer Berlin Heidelberg, 1998, pp. 114–134.

[48] S. Taylor, Extreme terseness: Some languages are more agile than others, Extreme Programming and Agile Processes in Software Engineering (2003) 1013–1013.

[49] W. Huang, P. Eades, S.-H. Hong, Measuring effectiveness of graph visualizations: A cognitive load perspective, Information Visualization 8 (3) (2009) 139–152.

[50] C. Ware, H. Purchase, L. Colpoys, M. McGill, Cognitive measurements of graph aesthetics, Information Visualization 1 (2) (2002) 103–110.

[51] K. Figl, J. Mendling, M. Strembeck, J. Recker, On the cognitive effectiveness of routing symbols in process modeling languages, in: Business Information Systems, Springer Berlin Heidelberg, 2010, pp. 230–241.

[52] H. Störrle, A. Fish, Towards an operationalization of the "physics of notations" for the analysis of visual languages, in: Model-Driven Engineering Languages and Systems, Springer Berlin Heidelberg, 2013, pp. 104–120.

[53] T. Green, A. Blackwell, Cognitive dimensions of information artefacts: a tutorial, in: BCS HCI Conference, Vol. 98, 1998, pp. 1–75.

[54] ISO, ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) - Part 11 : Guidance on usability, Tech. rep., The International Organization for Standardization (ISO) (1998).
URL https://www.iso.org/standard/16883.html

[55] A. F. Blackwell, T. R. G. Green, A cognitive dimensions questionnaire optimised for users, in: Psychology of Programming Interest Group, 2000, pp. 137–152.

[56] G. Kahraman, S. Bilgen, A framework for qualitative assessment of domain-specific languages, Software & Systems Modeling 14 (4) (2015) 1505–1526.

[57] S. Clarke, Describing and measuring API usability with the cognitive dimensions, in: Cognitive Dimensions of Notations Workshop, 2005, p. 131.

[58] D. I. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, A. C. Rekdal, A survey of controlled experiments in software engineering, IEEE transactions on software engineering 31 (9) (2005) 733–753.

[59] V. B. Kampenes, T. Dybå, J. E. Hannay, D. I. K. Sjøberg, A systematic review of effect size in software engineering experiments, Information and Software Technology 49 (11) (2007) 1073–1086.

[60] D. E. Perry, A. A. Porter, L. G. Votta, Empirical studies of software engineering: A roadmap, in: Future Of Software Engineering, ACM, 2000, pp. 345–355.
URL http://doi.acm.org/10.1145/336512.336586

[61] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-29044-2.

[62] J. Cohen, Statistical power analysis, Current Directions in Psychological Sciences 1 (3) (1992) 98–101.

[63] J. F. Hair, W. C. Black, B. J. Babin, R. E. Anderson, Multivariate data analysis, Prentice Hall Upper Saddle River, NJ, 2009.

[64] J. Osborne, Notes on the use of data transformations, Practical Assessment, Research and Evaluation 9 (1) (2005) 42–50.

[65] M. Kutar, C. Britton, T. Barker, A comparison of empirical study and cognitive dimensions analysis in the evaluation of UML diagrams, in: Psychology of Programming Interest Group, 2002, pp. 1–14.

[66] D. L. Moody, G. Sindre, T. Brasethvik, A. Solvberg, Evaluating the quality of information models: empirical testing of a conceptual model quality framework, in: International Conference on Software Engineering, IEEE Computer Society, 2003, pp. 295–305. doi:10.1109/ICSE.2003.1201209.

[67] I.-L. Huang, An empirical analysis of students difficulties on learning conceptual data modeling, Academy of Information and Management Sciences Journal 15 (2) (2012) 73.

[68] S. Hitchman, The details of conceptual modelling notations are important-a comparison of relationship normative language, Communications of the Association for Information Systems 9 (1) (2002) 10.