Computer Science Education

Vol. 00, No. 00, April 2011, 1–25

Research Article

Are Models Easier to Understand than Code? An Empirical Study on Comprehension of ER models vs SQL code

Pablo Sánchez*, Marta Zorrilla, Rafael Duque and Alicia Nieto-Reyes[a]

[a] *Dpto. Matemáticas, Estadística y Computación, Universidad de Cantabria, Santander, Spain*

(*Received 00 Month 200x; final version received 00 Month 200x*)

* Corresponding author.
Address: Facultad de Ciencias, Avda. Los Castros S/N
E39005, Santander (Cantabria, Spain)
Tlf: +34942203941, Fax: +34942201402
Email: p.sanchez@unican.es

ISSN: 0899-3408 print/ISSN 1744-5175 online

DOI:

http://www.informaworld.com

Abstract

Models in Software Engineering are considered as abstract representations of software systems. Models highlight relevant details for a certain purpose, whereas irrelevant ones are hidden. Models are supposed to make system comprehension easier by reducing complexity. Therefore, models should play a key role in education, since they would ease the students' learning process. Although these statements are widely accepted, to the best of our knowledge, there is no empirical evidence that supports these hypotheses (beyond practitioners' personal experience). This paper aims to contribute to fill this gap by performing an empirical study on how well students understand entity-relationship database models as compared to SQL code. Several ER models and their corresponding SQL code (more specifically, the DDL statements required to create such models) were shown to a heterogeneous group of students, who answered different questions about the database systems represented by these artifacts. Then, we analysed the correctness of the answers to check whether the ER models really improved students' comprehension.

Keywords: Entity-Relationship Models; SQL; Empirical Research; Software Models; Abstraction;

1 Introduction

Models in Software Engineering are said to be abstract representations of software systems that highlight relevant details for a certain purpose, whereas irrelevant ones are hidden (Ludewig, 2003; Seidewitz, 2003; Kühne, 2006). Thus, models are supposed to improve system comprehension by means of reducing complexity.

If we accept the previous statements as true, models should be the main artifact in software engineering education. If we can use models to highlight what we want to teach and to hide what is irrelevant in each lecture, why students should be overloaded with other artifacts, which include irrelevant details and increase complexity, instead of just using models focusing on the proper abstraction level?

Nevertheless, to the best of our knowledge, there is little empirical evidence that supports the previous hypotheses. These statements have been simply accepted by the software engineering community without further evaluation. Empirical evidence supporting these statements does not often go beyond practitioners' personal experience.

Although the authors of this paper are fully convinced of the benefits of using models in software engineering education as a mechanism to properly deal with the complexity of software systems, the point of view of the students might be just the opposite. They could argue that other software artifacts, such as plain source code, help them to understand software systems better because they need to know, or they feel more comfortable knowing, the irrelevant details hidden by the software models in order to fully understand how a software system works. If it were so, we should discard the use of models when teaching to improve the learning process of our students, since they are the target audience for our lectures. So, the question we would like to answer is: Are models really easier to understand than code?

This paper aims to contribute to fill this gap by performing an empirical study on how well students understand models as compared to plain code. In this paper, we will focus on database models. The main reason for selecting database models is that it was the kind of model for which

more students could participate in the experiments. Having a minimum number of students who we could use as subjects under study is a mandatory requirement for carrying out empirical studies. More specifically, we have analysed whether students understand Entity-Relationship (ER) models (Chen, 1976; Elsmari & Navathe, 2010) better rather than its corresponding plain SQL (*Structured Query Language*) code (ISO/IEC 9075-1, 2008; Gennick, 2010) (in order to be precise, we would like to point out we used the Data Definition Language (DDL) of the SQL standard).

The conclusions emerging from these empirical studies apply exclusively to database models. Therefore, they do not provide a complete answer to the general question about whether software models are easier to understand than code. Nevertheless, the conclusions presented in this paper provide an interesting and useful contribution to the answer to this question for other kinds of models. For instance, the results might be generalised somehow to the question about whether UML (*Unified Modelling Language*) class diagrams are easier to understand than its plain Java code counterpart.

To analyse empirically whether ER models are easier to understand than their equivalent SQL code, we carried out two similar empirical experiments involving 45 students of a Computer Science and Engineering degree. In each experiment, each student was provided with a description of a database schema either as an ER model or an SQL code. Then, the students received a test with several questions about the database schema. The students had to decide if each statement was true or false. We collected all these tests and we marked them. Finally, we performed a statistical analysis which allowed us to extract some conclusions. The results confirmed that ER models are easier to understand than SQL code. This tendency seems to increase when the time to answer the test decreases.

The rest of this paper describes how these experiments were carried out; it shows more detailed results of the statistical analysis; and discusses these results. Prior to this, Section 2 comments on related work we analysed in order to check there was not an enough number of these experiments

already available in the literature and, therefore, these experiments can provide some value to the community. Section 3 describes in detail how the empirical experiments were organised and carried out. Section 4 explains the statistical analysis performed, comments on the main findings emerging from such an analysis, discusses them, and it also analyses threats to validity for our experiments. Finally, Section 5 summarises the article and outlines the next steps to be taken with relation to this research topic.

2 Related Work

First of all, before carrying out our empirical studies, we checked whether similar studies have been carried out and reported in the literature. We found little work done in this area, and, to the best of knowledge, no similar work has been done using students as the target audience and with a educational purpose. This section comments on related work we have found in the literature.

Several researchers have looked at the effect of models on database development. For example, Batra et al. (1990) compared the users' performance on the task of database design using the ER model and the relational model. The design task was divided into various subtasks based on the elements of the ER model (e.g., unary and binary relationships). For most of the tasks, the ER model led to significantly more accurate results. Subjects also perceived the ER model to be slightly easier to use. Later, they carried out a similar study but this time oriented to model user views (Batra & Antony, 1994). The results the results allowed the conclusion that for novice designers the entity-relationship model was the appropriate choice. The task of database design was also investigated by Jarvenpaa & Machesky (1989), using the relational model and a binary ER model. Again, performance was better with the ER model.

Later, Chan & Wei (1993) and Chan et al. (1994) tested experimentally that the database users' performance in tasks such as data modelling, query writing and query comprehension was higher when users worked at the conceptual level using ER models than when they worked at the logical

level using SQL. The results showed that database users working at the conceptual level using ER models and an entity-relationship query language known as KQL produced results a 15.4% more accurate than users using SQL. Moreover, ER users produced these results in a 57.8% of the time required by the users working with SQL. In another empirical study, Chan et al. (1997) investigated the effect of entity-relationship versus relational models, and textual versus visual query languages, on database users. This study was extended later, including Object-Oriented models (Chan et al., 2005) and showing similar results.

All these studies seem to support our hypothesis. Nevertheless, they are mostly based on database design and query, and they evaluate issues such as accuracy or development/design time. In this paper, we have simply focused on comprehension, this is, we wanted to analyse whether models are useful mechanisms to transfer knowledge. We were not interested at all on how well or how fast students can produce models. Finally, we would like to highlight these studies were carried out approximately 20 years ago. Therefore, it makes sense to repeat them to check that which was true 20 years ago, it is still true, since students' skills change over the time.

Next sections describe how our empirical experiments were carried out and the results obtained.

3 Description of the Experimentation Procedure

The experiments reported in this paper were built to test a hypothesis: Entity-Relationship (ER) models are easier to understand than the corresponding SQL code. The procedure to check this hypothesis was to provide students with descriptions of a database schema either as an ER model or as SQL code. Then, we analysed statistically how these students understand each one of them. As null hypothesis, we considered that the marks obtained with the ER model and the SQL code are equally distributed. We test is against the following alternative hypothesis: the distribution of the marks obtained with the ER model is larger than the one obtained with the

SQL code. The experimentation procedure was comprised of the following steps:

(1) First of all, an ER model and its corresponding SQL code were created.

(2) Next, a set of questions to be answered as *true* or *false* were elaborated. These questions should be answered based on just the syntactic structure of either the ER model or the SQL code, i.e., no knowledge about the domain of the database schema had to be required.

(3) Then, a set of students, carefully selected for representing a wide spectrum of student types, was arranged to meet in specific dates. Half a group was given the ER model of the database system and the other half of the group was given the SQL code counterpart.

(4) The students had several minutes - depending upon the database schema size and complexity - to examine the ER model or the SQL code. A blank sheet of paper was given to each student, just in case the student needed to do some sketching.

(5) After this period, the questions about the database schema were given to the students.

(6) Finally, the questionnaires were collected, marked and the results statistically analysed.

We repeated this process twice, carrying out two experiments. Next, we comment on relevant issues that were taken into account in each step of this experimentation procedure.

3.1 *Database Schema Selection*

First, we looked for small to medium size database schemas with around 15-20 entities, where each entity were involved in around 3-5 relationships. We opted for this size because it is large enough for a database system not being considered trivial, and small enough to be handled by students - our target - in a reasonable frame of time. Moreover, the number of relationships is large enough to add some complexity to the understanding process, but small enough to not make the understanding task an artificially complex challenge.

To create the ER model and the corresponding SQL code, we decided we should base our experiments on industrial case studies. Therefore, we asked some companies for help. We got

different material from four Small-Medium Enterprises (SMEs), named Nuclenor [1], Predictia [2], Semicrol [3] and Suomitech [4]. Based on this material, we opted for using a database system of a software for storing and managing information about the maintenance jobs performed in a nuclear power plant. We selected this case study because of the following reasons:

(1) We were familiar the database because we had participated in its design;
(2) The database schema had the size we were looking for - neither too small to be considered as trivial nor too large to be considered as artificially complex or unfeasible to be handled by students.
(3) We assumed the domain was unfamiliar to the students. This would help us to avoid students being able to answer some of the questions using the knowledge they had about the domain. Therefore, an unknown domain would help to avoid some bias being introduced into the experiments.

Based on the material provided by this company, we created an ER model and its corresponding SQL counterpart. The database schema was initially too large, but it could be split up in two separated and weakly related parts. These two parts would be used to carry out two separate experiments. Thus, we wanted these parts to be weakly connected since it should help us to avoid that some learning acquired during the first experiment could introduce some bias in the second experiment. The first part of this database schema had 16 entities, 18 relationships and one generalisation. The second part had 17 entities, 17 relationships, two generalisations and 4 weak entities. The code for the first part was 6 pages long and the second one was 4 pages long.

Since the experiment was about database schema comprehension, we tried to avoid making the ER model and the corresponding SQL part more complex than strictly required. Therefore, in the case of the ER model, we tried to avoid line crossing as much as possible and we tried to group related entities in the same area of the model, taking care each model had a clear

[1] http://www.nuclenor.org/
[2] http://www.predictia.es/
[3] http://www.semicrol.es
[4] http://www.suomitech.com/

entry point and it could be easily read. To create the SQL code, we directly generated it from a relational model using SQL Server Rankins et al. (2010). For the ER to relational mapping, we used the classical algorithm to transform an entity-relationship model into a relational one. More specifically, we used the version of such an algorithm provided by Elsmari & Navathe (2010). We used this version of the algorithm since this is the one taught to the students, and, therefore, the one that students know best. Finally, we would like to point out we checked carefully that all questions asked could be properly answered using both the ER model and the corresponding SQL code.

Since this code was generated code, we *cleaned* it to make it more human-readable. For instance, we placed constraints related to foreign keys inside the code block corresponding to the creation of each table in order to keep grouped all code related to a same table. We also made some names more meaningful to human beings and we formatted the code properly. It should be noted that the SQL code is shown following this style to the students during the lectures, so it should also be presented in such a way to the students during the experiment.

3.2 *Elaboration of Questions*

To analyse how the students understand SQL code or ER models, we developed a set of questions about these artifacts. These questions had to be short questions that could be marked as *true* or *false*, in order to avoid intermediate values which made the statistical analysis more complex. To avoid students answering randomly, we required them to write a brief argumentation beside the selected option that justified such a selection. So, if the justification was right, the answer would be marked as right, otherwise it would be marked as wrong.

For both experiments, we created 15 questions about relationships between entities, 5 questions about generalisations, 5 questions about attributes, 5 questions about weak entities - when these weak entities were presented in the database schema - and 5 questions about if a set of data could be inserted in the database schema or there would be some problems instead. We show

some examples of these questions using the excerpt of the database schema depicted in Figure 1 (the corresponding SQL code is shown in Appendix A):

- **About relationships between entities:** A `Procedure` is related to exactly one `Function Machine`.
- **About generalisations:** All `Installations` can have associated a `PumpTestDefinition`.
- **About attributes:** Each `Procedure` has an `id`, which is unique for each `Procedure`.
- **About weak entities:** Each `PumpTestDefinition` is identified by one `Procedure` and one `Functional Machine`.
- **About data insertion:** To create a new `PumpTestDefinition` is enough with knowing the `id`, `rev`, `group`, `type`, `speed_ref`, `desc_press` and `press_diff` is enough to create .

We wanted to make the complete ER models, the corresponding SQL code and the questionnaires publicly available in order to allow replication of these experiments. However, due to legal constraints and security issues, we cannot make this material public. Indeed, we and the students were required to sign a confidentiality agreement. In case the reader is interested in replicating these experiments, we encourage him or her to contact us, in order to check whether it would be possible to sign an agreement between the reader and the company that provided us the case study.

Finally, we selected randomly 20 questions for the first experiment and 30 for the second one. The reason for this variation was the students did the first quicker than we initially planned and we wanted to analyse how they reason under certain time pressure. More details about this issue will be given in Subsection 3.4.

3.3 *Selection of the Students*

A key step when carrying out our experiments was the selection of the subjects under study. Since we wanted to analyse the influence of models on the students' learning process, our subjects had to be students.

To gather the subjects for our experiments, we disseminated a call for participation between the students of the second and fourth years of the Computer Science and Engineering degree, which is a five years degree. To motivate them and to ensure they did their best during the experiments, we did the following actions:

(1) The students were required to sign a commitment and confidentiality contract - the confidentiality part was required by the companies that provided us the case studies. By signing the contract, each student agreed to attend the experiments prepared to do his/her best, using exclusively his/her skills and knowledge.

(2) We agreed to give the students some extra points toward the final mark in one of the following subjects: Data Structures, Database Systems, Advanced Database Systems and Software Engineering II - these are the subjects the authors of this paper are currently teaching. The final mark of each one of these subjects is in the range between 0 and 10. Each student was granted a minimum 0.15 extra points in one of these subjects by simply participating in these experiments. In addition, each student could obtain a maximum of 0.5 points depending on the number of correct answers. Therefore, each student can obtain between 0.15 and 0.5 points to be added to his or her final mark in one of the mentioned subjects.

We would like to point out the participation in these experiments was offered to the students as an extra activity that was complementary to their academic duties. All students were free to participate in these experiments and they could also withdraw at any time without any consequence on their final marks. Thus, any student could obtain the maximum score in the subjects previously listed without having to participate in these experiments. Nevertheless, inspired by Hanenberg (2010), we decided we had to compensate our students for their effort in order to ensure they would do their best. Hanenberg, in his experiments, paid their students in function of both the quantity and the quality of the work done. Due to financial issues, we could not pay our students with money, so we decided to compensate them with some extra points in

their final marks depending upon both the quality and the quantity of the work done.

The participating students learn about ER models and SQL in the spring semester of the second year, in a Database Systems course. They have an optional course, in which not all students enroll, about Database Design and Administration in the fall semester of the third year and another optional course on Advanced Database Systems in the spring semester of the fourth course.

After disseminating the call for participation, 22 second-year students and 33 fourth-year students registered to participate in the experiments. Finally, due to several reasons, as health problems or schedule incompatibilities, some of these students did not attend the experiments, thereby reducing the number of participants to 45, 20 from the second year and 25 from the fourth year. Twenty-three (23) of the students used ER models and the other 22 SQL code. More specifically, 10 second-year students used ER models and the other 10 SQL code. From the 25 fourth-year students 13 of them used ER models and the other 12 SQL code.

We classified these students according to their academic performance into the following levels: *outstanding*, *good*, *normal*, *bad* and *very bad*. For this classification we used an objective indicator, their academic marks, and a subjective indicator, the impression we had as their teachers about the commitment and skills of each student. Then, we distributed them into two groups randomly, but ensuring both groups were balanced in number of students from each and also balanced in number of students of each academic level. This is, both groups had the same number of outstanding second-year students, good second-year students, normal second-year students, and so on. One of these groups would do the experiments using the ER description of the database schema and the other one would use the SQL code.

3.4 *Execution of the Experiment*

Once the students were classified and selected, we found the gap, after a quick but complex poll, in the students' schedule to carry out these experiments. These experiments took place outside

the lecture time. The students were not told about the structure or contents of the experiments thereby reducing bias in the results.

The initial intention was that they did not have enough time to answer all the questions comfortably. The reason for this is that we wanted to check whether the students were able to provide a proper answer to one question in a reasonable time frame. There is no doubt that having enough time, most of them would be able to provide satisfactory answers to most of the questions, either using the ER model or the SQL code.

In the first experiment, the students had 15 minutes to review either the ER model or the SQL code. Then, they received a test with 20 short questions and they had 30 minutes to answer them. Most of the students finished before this time elapsed. Thus, in the second experiment, they had 10 minutes for the review, the number of questions were 30 and they had 20 minutes to answer this group of questions. Only two students finished within the specified time.

Finally, we collected all the tests, marked them and analysed the results statistically to check whether our initial hypothesis held. The next section describes the results of the statistical analysis carried out.

4 Statistical Analysis of Results

This section describes the statistical analysis performed with the gathered data. It also discusses threats to validity.

4.1 *Statistical Analysis*

Once we processed the results of the tests, we analysed them to provide some statistical evidence about whether ER models are easier to understand than SQL code.

As already commented, a total of 45 students participated in these experiments. They were divided randomly in two balanced groups. We selected these two samples independently and using convenience sampling (Wohlin et al., 1999).

First of all, we did a descriptive analysis of the data, whose results are shown in Table 1 and depicted in Figure 2. This table shows what was the lowest mark for the ER and SQL experiments, the value for the first quartile, the median, the mean, the value for the third quartile and the standard deviation. It can be observed that the students who worked with the ER models got higher marks than those who worked with SQL code. Particularly, the median value[1] of the first ones is 6.78 (not a bad mark) while it is 5.17 for the second ones (just pass). Therefore, it seems that ER models are easier to understand than SQL code.

To confirm this phenomenon happens independently of which year of the degree the students are in, we have also done the previous descriptive analysis separately for the students in the second year and in the fourth year. The results are shown in Table 2 and illustrated in Figure 3. It can be can observed that in both kinds of students (second-year students and fourth-year students), the marks obtained when students work with the ER model are higher than those obtained when they work with the SQL code. It should also be noticed that the marks are higher for fourth-year students than for second-year students. Particularly, we can observe that the median mark for the SQL code is pass (5.42) for the fourth-year students and just below pass (4.92) for the second-year ones. The reason for this might be that second year students were recently taught about SQL, so they were more used to dealing with SQL code than the fourth-year students. Therefore, this phenomenon might indicate that skills developed on ER models remain longer in time than skills related to SQL code management.

Finally, we analysed the results of the two experiments separately. The reason for this is that the students were more constrained in time in the second experiment. Therefore, we wanted to analyse how they reason under certain time pressure. Table 3 shows the descriptive statistics for each experiment separately. These statistics are graphically depicted in the boxplots of Figure 4.

It can be noticed that in both cases ER marks are higher than SQL marks, but this difference is clearly higher in the second experiment. Therefore, this seems to confirm our hypothesis about

[1]We would like to point out we refer to the median instead of the mean because as the distribution of the data is unknown we prefer to refer to a robust measure of centrality.

ER models being easier to understand than SQL code when the time is constrained. Moreover, the deviation in the first experiment is clearly higher than in the SQL case. This might mean that whereas the average student understands ER models well, there are more variations in the SQL case.

In the second experiment, the standard deviation for the ER models grows higher than in the SQL case. An explanation for this might be that, in the first case, without the time being limited, some students finished earlier than the other students, but the number of right and wrong answers of each student was similar at the end. In the second experiment, we suppose outstanding students were able to complete a larger part of the test in the frame time, whereas the other students complete smaller parts of the test. However, in the SQL case, the standard deviation is lower than in the first experiment. This might mean that outstanding students were not able to take a noticeable advantage to the other students when dealing with SQL code, i.e., reasoning quickly on SQL code seems to be difficult to any kind of student, from an outstanding student to very bad student. Indeed, we would like to highlight that the third quartile for the SQL code is below the first quartile for the ER model, showing clearly how students using the ER model beat the students using the SQL code. Moreover, it should be noticed that the best mark for the SQL code is approximately the median value for the students using the ER model. This means that the most outstanding student dealing with the SQL code performed more or less equally than an average student using the ER model.

To check whether there is a significant difference between the understanding of ER models and SQL code we formulated it as a hypothesis that we could check with a hypothesis test. Thus, our null hypothesis was:

H_0 : The marks obtained with the ER model and the SQL code are equally distributed.

And we will test it against the alternative hypothesis:

H_a : The distribution of the marks obtained with the ER model is larger than the one obtained with the SQL code.

Denoting by F_m the distribution function corresponding to the ER model and by F_c the one corresponding to the SQL code, we can rewrite these hypothesis as:

$$H_0 : F_m(x) = F_c(x) \text{ for all } x.$$

$$H_a : F_m(x) \leq F_c(x) \text{ where the strict inequality occurs for at least one } x.$$

Note that this alternative hypothesis denotes that the marks for the ER models tend to be larger than the marks observed for the SQL code.

For testing these hypothesis we used the two-sample permutation test, see Higgins (2003), a non-parametric test which does not require assumptions, such as happens for example in the t-test. The obtained p-value is $3.942 \cdot 10^{-5}$. Therefore, we can reject the null hypothesis. We have also done the test separately for the students in the second and fourth year, obtaining respectively a p-value of 0.006245 and of 0.000651. Although these two p-values are higher than the previous one (the sample sizes are smaller) they are well below the intended level. Finally, we applied this test to the first and the second experiment individually, obtaining a p-value of 0.002293 and $1.474 \cdot 10^{-5}$ respectively.

Thus, we can reject the null hypothesis in both cases and and we can state the ER models are easier to understand than the corresponding SQL code at level of significance smaller than 0.01. The differences between the medians of the ER models and the SQL code is 1.73 for the general case, 1.76 for the second-year students, 1.67 for the fourth-year students, 1.07 for the first experiment and 2.49 for the second one. So, students seem to be able to reason better using the ER models rather than the SQL code, at least around 1.7 out 10 better in the general case. Nevertheless, this value can vary from 1.0 without time limitations to 2.5 under certain time constraints.

In the next subsection, we comment on threats to validity for the results of our experiments.

4.2 *Threats to Validity*

The reader could argue the reason for the differences between the results for the ER models and the SQL code might be that the students working the ER models had higher skills than the students working with the SQL code. We would like to remind the reader that the groups were balanced in number of students of each year and academic performance. Moreover, these students were assigned randomly to each group. For instance, four outstanding second-year students enrolled in the experiments. So, we selected two of them randomly for the ER group. Consequently, the other two were assigned to the SQL group. Therefore, the groups should have a similar performance.

Other argument against these results could say these differences are a consequence of the kind of questions asked, i.e., a different kind of questions should provide different results. In this respect, we would like to comment that we had focused on the kind of questions students had often to deal with during a lecture. Moreover, we tried to include in the tests questions which should be more easily answered using an ER model and questions which should be more easily answered using the corresponding SQL code. We assume questions about relationships between entities should be more easily answered using ER models, since relationships between entities are more explicit in these models. On the other hand, questions related to primary keys or constraints for inserting a data tuple into a table should be more easily answered using the SQL code counterpart.

For instance, in Figure 1, PumpTestDefinition is a weak entity whose primary key is the union of the primary keys of the entities that identify it, i.e., Procedure and Functional Machine. In addition, Functional Machine is a child entity that inherits its primary key from its parent entity, Installation. Thus, to find the primary key of the PumpTestDefinition entity, we need to examine three entities. In the SQL case, to know such a primary key, it is enough with looking at the code of the PumpTestDefinition table.

It could also be argued the results obtained in this paper apply exclusively to the case study we

have used, i.e., the reader could say the more precise conclusion for this paper should be the ER model of our the electrical power plant is easier to understand than its corresponding SQL code. Other case studies with different peculiarities might provide different results. We would like to comment that, although it would have been desirable to carry out a large set of experiments to check this hypothesis, this was not feasible at all. Performing this kind of experiments requires a large set of resources. First of all, we needed to find students who volunteered to perform these experiments. Then, we had to find a gap in the students schedules - they were enrolled in different courses with different timetables and each one of them had his/her own external duties to address. In addition, we had to book a room in our building for carrying out these experiments - large rooms are often not available. Finally, we had to prepare the material for the experiment, which also consumed a considerable effort and amount of time. Therefore, due to the available resources we were limited to two experiments..

Obviously, two experiments are not enough to confirm or reject this hypothesis with total confidence, i.e., these experiments cannot be used as a proof or a refutation for the hypothesis being tested. As the hypothesis turned out to be confirmed by the experiment, the experiment can be considered as empirical evidence about the veracity of such a hypothesis. Moreover, the previous empirical experiments reported in the Section 2 should not be neglected and, together with the results presented in this paper, be considered as a corpus of empirical evidence about the better understanding of the ER models as compared to their corresponding SQL code.

Finally, we would like to mention that if the experiment had rejected the hypothesis, we would have found a situation where the hypothesis does not hold and more research would be required to determine the circumstances under which models are not easier to understand than code. But it would not be a proof of the negation of the hypothesis being tested, i.e., we could not conclude that SQL code is easier to understand than ER models.

5 Conclusions and Future Work

This article has reported the result obtained after carrying out two empirical experiments about whether students understand better ER models rather than SQL code. The results showed a better performance of ER models as compared to SQL code, mainly when the time to reason was limited. Moreover, the capacity to reason on ER models seems to remain longer in time than the same capacity on SQL code.

The final goal of this study was to provide some empirical evidence about whether models are really easier to understand than plain code. Although we have limited the analysis to ER models and SQL code, we are convinced that similar results would have been obtained using other pairs of model/code, as UML class diagrams and plain Java code. According to the obtained results, we claim models should have a key role in computer science education, since they would help to ease the students learning process. This key role should also be extended to professional software development, because if models are easier to understand, it is reasonable to think they should also be easier - and faster - to construct and manage. Therefore, models should be the key artifacts in the software development process. Consequently, the teaching of model-driven techniques for model management and transformation, such as graph-based model transformation Rozenberg (1997), would be clearly justified.

Before concluding the paper, we would like to highlight that Carver et al. (2003) presented an empirical study which concludes that the results obtained from empirical studies using students as subjects can be extrapolated to professional developers and software engineers. Therefore, although this work has used students, its results can also be applied to professional developers, more or less.

After this experience, we will repeat these experiments with different students and different case studies in order to confirm whether similar results are obtained. We will also carry out similar experiments using other kinds of models, such as UML class diagrams, state machines or Petri nets. We are also planning to do a more detailed analysis in order to know what kind

of questions are more easily answered using ER models and for which ones to use SQL code is more adequate.

A reasonable doubt is whether the better performance of ER models is simply due to the fact of the models being visual. Therefore, to create a visual notation for SQL code would be enough to achieve a level of comprehension similar to EER models. We will explore this issue by creating a visual notation for SQL and comparing EER models and SQL code depicted using this visual notation. Another interesting experiment would be to create a textual notation for EER models and compare it against textual SQL code. Both experiments would give important clues about whether comprehension is improved because EER models are more abstract or simply because they are visual, or maybe because of both issues at the same time.

Acknowledgement(s)

We would like to thank to the companies (Nuclenor, Predictia, Semicrol and Suomitech[1]) which helped us by providing different case studies that served as guide to prepare the material for the experiments. We would also like to thank to all the students who participated in the experiments. We really appreciate their extra effort, collaboration and commitment. Finally, we would also like to give thanks to our colleagues Carlos Blanco and Daniel Sadornil, who gave a hand whenever it was required.

References

Batra, D., & Antony, S.R. (1994). Effects of Data Model and Task Characteristics on Designer Performance: a Laboratory Study. *International Journal of Human-Computer Studies*, *41*(4), 481–508.

Batra, D., Hoffer, J.A., & Bostrom, R.P. (1990). Comparing Representations with Relational and EER Models. *Communications of the ACM*, *33*, 126–139.

[1]All acknowledgements are given in alphabetical ordering

Carver, J., Jaccheri, M.L., Morasca, S., & Shull, F. (2003). Using Empirical Studies during Software Courses. In R. Conradi & A.I. Wang (Eds.), *Empirical Methods and Studies in Software Engineering, Experiences from ESERNET*Vol. 2765, pp. 81–103). .

Chan, H., & Wei, K. (1993). User-Database Interface: The Effect of Abstraction Levels on Query Performance. *MIS Quarterly*, *17*, 441–464.

Chan, H.C., Siau, K., & Wei, K.K. (1997). The Effect of Data Model, System and Task Characteristics on User Query Performance: An Empirical Study. *SIGMIS Database*, *29*, 31–49.

Chan, H.C., Teo, H.H., & Zeng, X. (2005). An Evaluation of Novice End-User Computing Performance: Data modeling, Query Writing and Comprehension. *Journal of the American Society for Information Science and Technology (JASIST)*, *56*(8), 843–853.

Chan, H.C., Wei, K.K., & Siau, K. (1994). An Empirical Study on End-users' Update Performance for Different Abstraction Levels. *International Journal Human-Computer Studies*, *41*(3), 309–328.

Chen, P.P. (1976). The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, *1*(1), 9–36.

Elsmari, R., & Navathe, S. (2010). *Fundamentals of Database Systems*. Pearson Education.

Gennick, J. (2010). *SQL Pocket Guide*. O'Reilly.

Hanenberg, S. (2010). An Experiment about Static and Dynamic Type Systems: Doubts about the Positive Impact of Static Type Systems on Development Time. In Proc. of the 25th Annual Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA), October (pp. 22–35)., Reno/Tahoe (Nevada, USA).

Higgins, J.J. (2003). Duxbury Advanced Series*Introduction to Modern Nonparametric Statistics*. Thomson.

ISO/IEC 9075-1, ISO/IEC 9075-1:2008: Information technology Database languages SQL Part 1: Framework (SQL/Framework). (2008). , Technical report.

Jarvenpaa, S.L., & Machesky, J.J. (1989). Data Analysis and Learning: An Experimental Study

of Data Modeling Tools. *International Journal of Man-Machine Studies*, *31*(4), 367–391.

Kühne, T. (2006). Matters of (Meta-)Modeling. *Software and System Modeling*, *5*(4), 369–385.

Ludewig, J. (2003). Models in Software Engineering. *Software and System Modeling*, *2*(1), 5–14.

Rankins, R., Bertucci, P.T., Gallelli, C., & Silverstein, A.T. (2010). *Microsoft SQL Server 2008 R2 Unleashed*. Sams.

Rozenberg, G. (1997). *Handbook of Graph Grammars and Computing by Graph Transformation*. World Scientific Publishing.

Seidewitz, E. (2003). What Models Mean. *IEEE Software*, *20*(5), 26–32.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., & Wesslén, A. (1999). *Experimentation in Software Engineering: An Introduction*. Kluwer Academics.

Appendix A: SQL (DDL) code

```
CREATE TABLE [dbo].[installation](
        [installation_id] char(10) NOT NULL,
        [installation_type] char(10) NOT NULL
                CHECK ([installation_type] IN ('FUNCTIONAL_MACHINE','SYSTEM')),
        [installation_desc] char(255) NOT NULL,
                CONSTRAINT [PK_INSTALLATION] PRIMARY KEY CLUSTERED
                                                ([installation_id] ASC, [installation_tipo] ASC)
) -- installation

CREATE TABLE [dbo].[system](
        [installation_id] char(10) NOT NULL,
        [installation_tipo] char(10) NOT NULL,
        [system_synonim] char(5) NULL,
        [system_creationDate] date NOT NULL,
        [system_updateDate] date NOT NULL,
        [system_deleteDate] date NULL,
        CONSTRAINT [PK_system] PRIMARY KEY CLUSTERED
                ([installation_id] ASC, [installation_type] ASC)),
        CONSTRAINT [FK_installation_system]
                FOREIGN KEY([installation_id], [installation_type])
                REFERENCES [dbo].[installation] [installation_id], [installation_type])
) -- system
```

```
CREATE TABLE [dbo].[functional_machine](
        [installation_id] char(10) NOT NULL,
        [installation_type] char(10) NOT NULL,
        [funmach_creationDate] date NOT NULL,
        [funmach_updateDate] date NOT NULL,
        [funmach_deleteDate] date NULL,
        CONSTRAINT [PK_FUNCTIONAL_MACHINE] PRIMARY KEY CLUSTERED
          ([installation_id] ASC, [installation_type] ASC),
        CONSTRAINT [FK_installation_functional_machine]
          FOREIGN KEY([installation_id], [installation_type])
          REFERENCES [dbo].[installation] ([installation_id], [installation_type]),
) -- functional_machine

CREATE TABLE [dbo].[procedure](
        [procedure_id] char(20) NOT NULL,
        [procedure_rev] char(3) NOT NULL,
        [procedure_desc] varchar(200) NOT NULL,
        CONSTRAINT [PK_PROCEDURE] PRIMARY KEY CLUSTERED
                ([procedure_id] ASC, [procedure_rev] ASC)
) -- procedure


CREATE TABLE [dbo].[PumpTestDefinition](
        [installation_id] char(10) NOT NULL,
        [instalacion_type] char(10) NOT NULL,
        [proc_ID] char(20) NOT NULL,
        [proc_rev] char(3) NOT null,
        [pump_type] char(1) NOT NULL check ([pump_type] in ('A','B','G')),
        [pump_group] char(1) NULL check ([pump_group] in ('A','B')),
        [pump_speed_ref] decimal(10,2) NULL,
        [pump_desc_press] decimal(10,2) NULL,
        [pump_press_diff] decimal(10,2) NULL,
        CONSTRAINT [PK_PUMP_TEST_DEFINITION] PRIMARY KEY CLUSTERED
                ([installation_id] ASC, [installation_type] ASC,
                 [proc_ID] ASC, [proc_rev] ASC),
        CONSTRAINT [FK_PumpTestDefinition_FunctionalMachine]
                FOREIGN KEY([installation_id],[installation_type])
                REFERENCES [dbo].[functional_machine]
                        ([installation_id],[installation_type]),
        CONSTRAINT [FK_PumpTestDefinition_Procedure]
                FOREIGN KEY([proc_ID],[proc_rev])
```

```
    REFERENCES [dbo].[procedure] ([procedure_id],[procedure_rev]),
) -- PumpTestDefinition
```

Tables and Figures

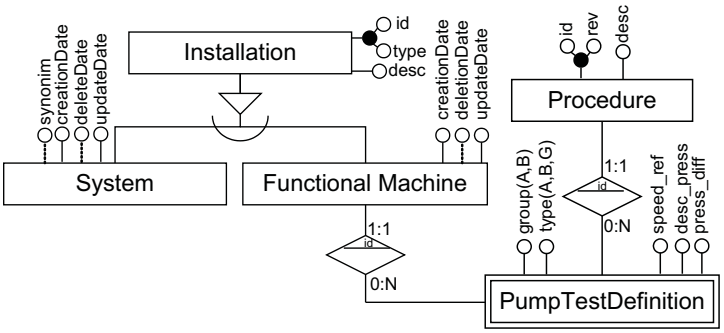

Figure 1. Excerpt of the database schema used for the experiments

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. Dev.
ER model	3.889	5.778	6.778	6.696	7.472	8.889	1.297
SQL code	2.611	4.361	5.167	4.970	5.722	6.333	1.070

Table 1. Descriptive statistics for the ER model and the SQL code.

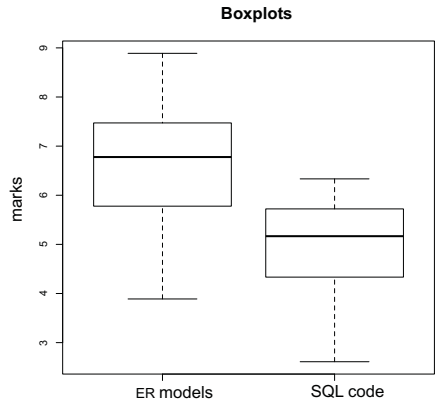

Figure 2. Boxplots of the ER model (left) and the SQL code (right)

Second-year students

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. Dev.
ER model	3.889	5.778	6.111	6.289	6.722	8.889	1.475
SQL code	2.611	3.917	4.917	4.533	5.167	5.944	1.151

Fourth-year students

ER model	5.444	5.833	7.111	7.009	7.500	8.889	1.100
SQL code	3.667	4.819	5.417	5.333	6.167	6.333	0.885

Table 2. Descriptive statistics separated according students' degree year.

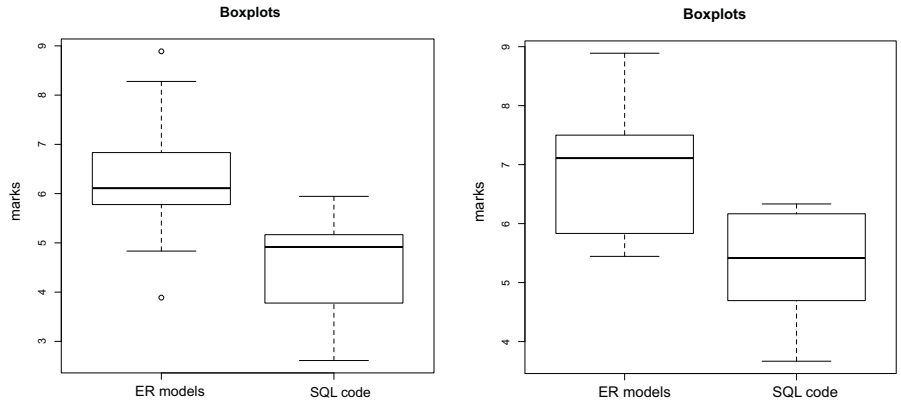

Figure 3. Boxplots corresponding to second-year (left) and fourth-year (right) students.
Inside each graph, boxplots for the ER model (left) and the SQL code (right).

First Experiment (20 questions/30 minutes)

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Std. Dev.
ER model	6.667	7.778	8.333	8.188	8.889	9.444	0.7532
SQL code	3.889	6.667	7.500	7.121	8.333	8.889	1.4612

Second Experiment (30 questions/20 minutes)

ER model	2.667	4.083	5.333	5.439	6.583	8.667	1.812
SQL code	1.333	2.000	2.667	2.952	3.667	5.667	1.107

Table 3. Descriptive statistics separated by experiment.

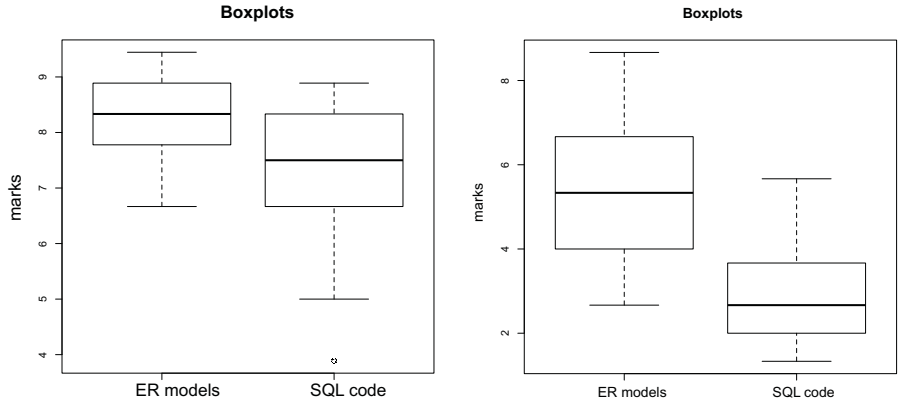

Figure 4. Boxplots corresponding to first (left) and second (right) experiment.
Inside each graph, boxplots for the ER model (left) and the SQL code (right).