Garzás J, García F, Piattini M. Do rules and patterns affect design maintainability? JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 24(2): 1– Mar. 2009

Do Rules and Patterns Affect Design Maintainability?

Javier Garzás[1], Félix García[2], and Mario Piattini[2]

[1] *Kybele Consulting, Madrid, Spain*

[2] *Alarcos Research Group – Institute of Information Technologies and Systems, Department of Information Technologies and Systems — Escuela Superior de Informáica, University of Castilla-La Mancha, Paseo de la Universidad, 4 13071, Ciudad Real, Spain*

E-mail: javier.garzas@kybeleconsulting.com; {Felix.Garcia, Mario.Piattini}@uclm.es

Received February 29, 2008; revised September 21, 2008.

Abstract At the present time, best rules and patterns have reached a zenith in popularity and diffusion, thanks to the software community's efforts to discover, classify and spread knowledge concerning all types of rules and patterns. Rules and patterns are useful elements, but many features remain to be studied if we wish to apply them in a rational manner. The improvement in quality that rules and patterns can inject into design is a key issue to be analyzed, so a complete body of empirical knowledge dealing with this is therefore necessary. This paper tackles the question of whether design rules and patterns can help to improve the extent to which designs are easy to understand and modify. An empirical study, composed of one experiment and a replica, was conducted with the aim of validating our conjecture. The results suggest that the use of rules and patterns affect the understandability and modifiability of the design, as the diagrams with rules and patterns are more difficult to understand than non-rule/pattern versions and more effort is required to carry out modifications to designs with rules and patterns.

Keywords design patterns and rules, maintainability, software quality, object-oriented design

1 Introduction

For many years now, design principles, heuristics, bad smells, best practices and patterns have been associated with elements for improving software maintainability[1−3]. Principles, heuristics, bad smells, and so on have the same common structure, there being no substantial difference between them, as they all have the structure and form of a rule — they posit a condition and offer a recommendation[4]. Hence, in the context of the present work we use two terms: rule (to group elements such as principles, heuristics, bad smells, best practices, etc.) and pattern. The main differences between these concepts are that patterns are more formalized than rules and pattern descriptions are always broader. Patterns propose solutions to problems whereas rules are recommendations which are, to a great extent, based on the use of natural language[4,5]. Maintainability and rules and patterns are popularly synonymous with well-designed software. A quality design that has a high-level maintainability could thus have a clear effect on the productivity of developers. The effectiveness of maintenance is, furthermore, greatly hindered by poor design[6].

These seemingly obvious statements are questioned by many professionals. Glass states that well-designed systems will be improved more rapidly and have a longer life and will hence, paradoxically, consume more maintenance[7]. Reibing[8] comments that if we have two similar designs, A and B, for the same problem, B using design patterns and A not using design patterns, B should have a higher quality than A, but if we apply "classic" object-oriented design metrics to both designs, the metrics will tell us that design A is better. This is related to the fact that patterns generally increase the complexity of an initial design in order to ease future enhancements[9].

The relationship between maintainability and rules and patterns, then, is not clear, nor are their positive or negative effects. In the present state of affairs, there is a serious lack of hard scientific evidence for the usefulness of rules or of rules and patterns in many situations[10].

Regular Paper

This research has been partially supported by the following projects: MECENAS (Junta de Comunidades de Castilla-La-Mancha, Consejeria de Educacion y Ciencia) under Grnat No. PBI06-0024, ESFINGE (Dirección General de Investigación of the Ministerio de Educación y Ciencia) under Grant No. TIN2006-15175-C05-05 and IDONEO (Junta de Comunidades de Castilla-La-Mancha, Consejeria de Educacion y Ciencia) under Grant No. PAC08-0160-6141.

There is no firm and formal theory that links rules or rules and patterns to these software quality concepts[2] so it is necessary to build an empirical body of knowledge with which confirm these assertions. The question, therefore, is this: what effect do rules and design rules and patterns have on maintainability?

With regard to the empirical validation of design pattern effects on maintainability, most of the related work in literature has tackled this from the perspective of code (see Section 2). Nevertheless, software models are currently taking on key importance, with the appearance of the Model-Driven Engineering paradigm (MDE)[11]. The main goal of this is to ensure that the core artifacts in the software engineering processes will be models rather than code, so that designs are expressed and managed in the manner of models, with a much higher level of abstraction than the code. This has two main advantages: i) it reduces complexity during the design phase when complexity does not need to be taken into account, due to the specific technology of implementation, and ii) it obtains generic systems, since it is possible to create software with more general functionalities that give solutions to a greater quantity of cases or situations. Yet, as we shall see, the potential benefits of using models are significantly greater in software than in any other engineering discipline. Model-driven development methods were devised to take the advantage of this opportunity, and the accompanying technologies have matured to the point where they are generally useful[12].

As a consequence, the potential effects which rules and patterns may have on design models have become topic of interest, as the quality of the models can directly affect the quality of the code produced. This paper proposes to seek empirical evidence about whether using rules and design rules and patterns are beneficial for software design maintainability as this is evaluated through the understandability and modifiability of the software design. We conducted an experiment and a replica to compare performances of designs with rules and rules and patterns to alternatives in which they were not used. The remainder of this article is organized as follows. Section 2 provides an overview of related work. In Section 3 the experimental settings are described. Section 4 summarizes data analysis and its interpretation. Section 5 outlines the limits of the experiment. Finally, conclusions are drawn and future work is set out in Section 6.

2 Related Work

In the last few years, most empirical studies of patterns have analyzed the usefulness of design patterns in program code maintenance. The hypothesis is that the use of the most commonly referenced design patterns should promote adaptable and reusable program code[13].

Prechelt *et al.*[14] carried out a controlled experiment with professionals to analyze whether the use of patterns could aid software maintenance. Several software maintenance scenarios which employed various design patterns were used, and designs with patterns were compared to simpler alternatives. In most of the maintenance tasks, positive effects were obtained from using a design pattern. The pattern's inherent additional flexibility was achieved without the need for more maintenance time, or maintenance time was actually reduced in comparison to the simpler alternative. The authors emphasize that unless there is a clear reason for preferring the simpler solution, it is probably wise to choose the flexibility provided by the design pattern, since unexpected new requirements often appear. Bieman *et al.*[13] conducted an industrial case study to identify the observable effects of the use of design patterns in early versions on changes that occur as the systems evolve. As a result, in four of the five systems analyzed, classes which played roles in design patterns were more prone to changes than other classes. Masuda *et al.*[15] carried out a case study in which a software system was redesigned by using design patterns. The quantitative evaluation of the effectiveness of applying design patterns show that the use of design patterns improved the system's flexibility and extensibility.

Vokàc[16] performed a case study in which a weekly evolution and maintenance (over three years) of a large commercial product were analyzed. Significant differences in defect rates among the patterns were found, ranging from 63% to 154% of the average rate between the patterns. These results lead to the conclusion that, as well-known design patterns have widely different sizes, complexities, and applicability, the use of patterns in itself does not guarantee that there will be very few defects.

Elish[17] discussed, with examples, the impact of four structural design patterns on the stability of class diagrams. The examples indicated that the design patterns studied had a positive impact on class diagram stability, although empirical validation was not provided, as this lies within the scope of the future work of the study.

A further important perspective discussed in literature is whether design patterns are actually utilized by maintainers. Regarding this aspect, Ng *et al.*[18] empirically studied whether maintainers utilize deployed design patterns, when they do so, and which tasks

they most commonly perform. The experiment was conducted with 215 undergraduate students, who were asked to perform three kinds of maintenance tasks on an existing software code: T1, adding a new class as a concrete participant; T2, modifying the interfaces of a participant; T3, introduction of a new client. As a result of this experiment, the utilization of deployed design patterns was significantly related to the delivery of less faulty code (independently of the task performed, T1 being that which was most frequently applied). The authors of [19] investigated the benefits of applying program refactoring in order to introduce additional design patterns, regardless of the work experience of maintainers, for which an open source system deployed with Composite, Decorator, Factory Method and Observer patterns is used. The conclusion of the empirical study is that the time spent by inexperienced maintainers on a refactorized version is much shorter than that spent by experienced subjects on the original version when perfective maintenance is carried out, but that the correctness of their delivered programs does not significantly differ.

Finally, other aspects of patterns have been researched in literature, such as the usefulness of pattern documentation in making program maintenance easier, which has been tackled in [20, 21]. The assumption is that maintenance can be carried out faster and with fewer errors if design patterns are explicitly documented. In [22] some design restrictions in pattern deployments are proposed to achieve proper encapsulation, obtaining as a result that the approach facilitates program changes subject to multiple design concerns.

To conclude this point, although several studies concerned with the empirical validation of pattern maintainability exist, the building of a body of knowledge around the potential benefits of design rules and patterns is still at an initial stage. The empirical validation presented in this paper mainly differs from the related literature in that we have raised the abstraction level (design instead of code). To the best of our knowledge, with the exception of the preliminary research presented in [8], there is no empirical validation of design maintainability in which the experimental tasks are performed on software designs instead of the source code of programs. Therefore, our aim in the present work is to analyze the influence of the application of rules and rules and patterns on design understandability and modifiability.

3 Description of the Experimental Settings

As the main goal of this work is to ascertain whether the use of design rules and rules and patterns can make the design easier to understand and modify, our research question can be stated as:

Does the use of rules and rules and patterns improve the maintainability of design?

From the above question, the general goal of our empirical study is derived as follows:

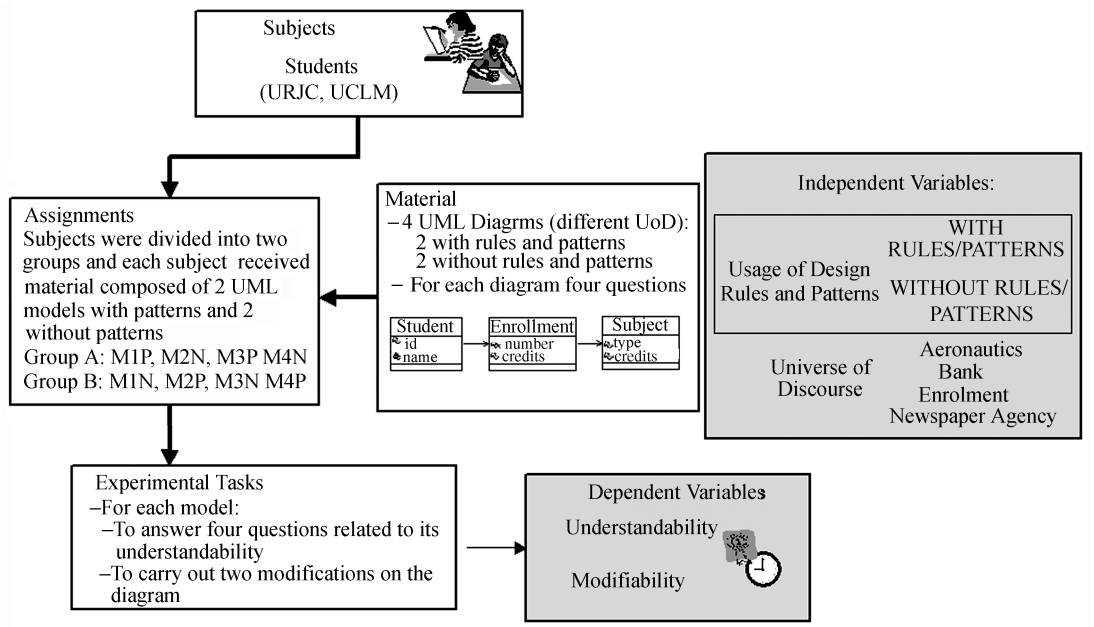


Fig.1. Overview of the experimental plan.

Analyze design diagrams with and without rules and patterns *with the purpose of* comparing them, *with regard to* their maintainability, *from the point of view of* the designers *within the context of* undergraduate students of Software Engineering.

Fig.1 shows an overview of the experimental plan.

3.1 Subjects

The subjects of the first experiment were 10 undergraduate students from the 3rd course of computer science at the Rey Juan Carlos University (URJC, Madrid, Spain). For the purpose of confirming the results, a replica was conducted in the Department of Technology and Information Systems at the University of Castilla-La Mancha (UCLM, Ciudad Real, Spain), in which 37 students from the final year (5th year) of the M.Sc. participated. The students in both experiments were familiar with UML and design rules and patterns, as these were topics which had been taught on their software engineering course. Additionally, a short training session was provided for them, in which the design rules and patterns and UML class diagram constructs were commented and an example of the tasks to be performed in the experiment was explained. Given the above, we consider that the level of experience they brought to the experiment was acceptable.

3.2 Material

The material was prepared by the experiment designers and was composed of:

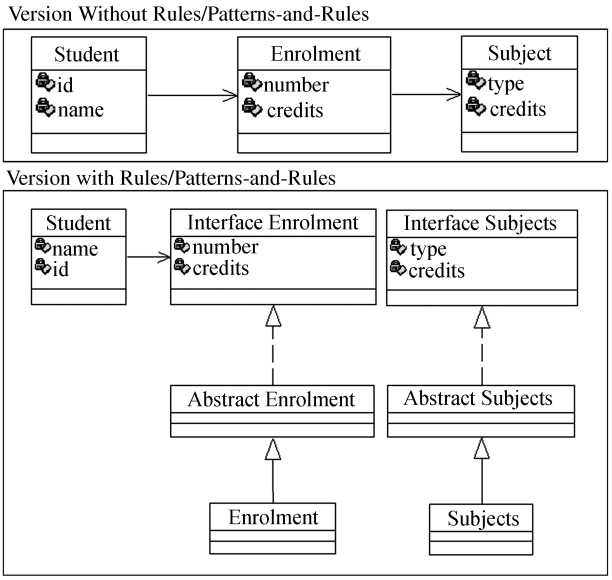
• Four Universes of Discourse (UoD) (Aeronautics, Bank, University and Newspaper Agency). For each UoD, two semantically equivalent UML diagrams were prepared: one including rules and rules and patterns, and the other not containing them. Hence, a total of eight UML class diagrams were prepared. The rules and patterns included in the diagrams were representative of the related literature and were selected according to the experimental design restrictions and our own professional and academic experience. (A more detailed description of the rules and patterns used in the experimental material is shown in Table 1.)

• Each diagram included a *Questionnaire* composed of *four questions* (answer yes/no) related to the model and *two new requirements* which are involved in making some modifications to the diagrams. For each UoD the same set of four questions and new requirements was provided.

In order to carry out the experiment we obtained two sets of experimental material by grouping the diagrams in the following manner:

Table 1. Rules and Patterns Included in the Experimental Material

UoD	Patterns	Rules
Aeronautics	The State Pattern[2]	IF there is no abstract class between an interface and its implementation THEN create an abstract class with an implementation by default between the interface and the class that implements it[23]. "IF a change in an interface has an impact on many clients THEN create specific interfaces for each client (also known as Interface Segregation Principle (ISP)[24]).
Bank	Composite and State Pattern[2]	–
University	–	IF there are dependencies on concrete classes THEN these dependencies should be on abstractions. This depends on interfaces or abstract classes rather than on concrete elements. This rule is also known as a dependency inversion rule[24], or "programming to an interface, not an implementation. "IF there is no Abstract class between an Interface and its Implementation" THEN create an abstract class with an implementation by default between the interface and the class that implements it. This eliminates the possibility of a duplicate code: all subclasses have a default implementation, such that each class does not need to create one[23].
Newspaper Agency	Chain of Responsibility[2]	IF there are dependencies on concrete classes THEN these dependencies should be on abstractions. This depends on interfaces or abstract classes rather than on concrete elements. This rule is also known as a dependency inversion rule[24], or "programming to an interface, not an implementation." "IF there is no Abstract class between an Interface and its Implementation" THEN create an abstract class with an implementation by default between the interface and the class that implements it. This eliminates the possibility of a duplicate code: all subclasses have a default implementation, such that each class does not need to create one[24].



QUESTIONNAIRE (University)
WRITE DOWN THE STARTING TIME
(INDICATING HH:MM:SS): _____

Please answer the following **questions** related to the diagram.

1. Is it possible for a student to discover all the subjects which exist at the University? _____
2. Is it possible for a student to discover all the subjects in which s/he is enrolled? _____
3. Is it possible for a subject to know all the students who are enrolled in it? _____

NEW REQUIREMENTS (University)
2) What **modifications** would you make to satisfy the following requirements?

1. Two different types of enrolments are to be considered; one for degree subjects and the other for summer courses.

WRITE DOWN THE STARTING TIME
(INDICATING HH:MM:SS): _____
...
WRITE DOWN THE FINISHING TIME
(INDICATING HH:MM:SS): _____

2. Masters courses are also to be considered, so they can be managed in the same way as subject enrolments.

Fig.2. Example of experimental material: UML diagrams for UoD enrolment (groups A and B versions).

• *Group* A: Diagram 1 (Aeronautics without rules and patterns), Diagram 2 (Bank with rules and patterns), Diagram 3 (University without rules and patterns), Diagram 4 (Newspaper Agency with rules and patterns).

• *Group* B: Diagram 1 (Aeronautics with rules and patterns), Diagram 2 (Bank without rules and patterns), Diagram 3 (University with rules and patterns), Diagram 4 (Newspaper Agency without rules and patterns).

Fig.2 illustrates the UoD "Enrolment" in these two design versions.

3.3 Variables

The dependent variables of our empirical study were the following maintainability sub-characteristics:

• The architectural design *understandability*, which we measured through the *time* spent by the subjects on answering the four questions, the *number of correct answers* and the *efficiency*, i.e., the relationship between the number of correct answers and the time they spent on answering them.

• The architectural design *modifiability*, which was measured by the *time* the subjects spent on carrying out the necessary modifications, the *score* of the modifications, and the *efficiency* of those modifications.

Our main independent variable was the *use or non-use of rules and patterns* in the diagrams. For a complete analysis of the experimental data, the UoD was also considered as an independent variable.

3.4 Hypotheses

The experiments were planned with the purpose of testing the following hypotheses, mainly:

• *Null Hypothesis,* $H_{0\mathrm{Pu}}$: the use of rules and patterns does not improve the understandability of the architectural design.

• *Alternative Hypothesis,* $H_{1\mathrm{Pu}}$: the use of rules and patterns improves the understandability of the architectural design.

• *Null Hypothesis,* $H_{0\mathrm{Pm}}$: the use of rules and patterns does not improve the modifiability of the architectural design.

• *Alternative Hypothesis,* $H_{1\mathrm{Pm}}$: the use of rules and patterns improves the modifiability of the architectural design.

In addition to the above objectives, the testing of the following set of hypotheses was a secondary goal of the empirical study:

• *Null Hypothesis,* $H_{0\mathrm{UoD}}$: there are no differences in understandability or modifiability due to the Universe of Discourse of the diagram.

• *Alternative Hypothesis,* $H_{1\mathrm{UoD}}$: there are differences in understandability or modifiability due to the Universe of Discourse of the diagram.

• *Null Hypothesis,* $H_{0\mathrm{PxUoD}}$: there are no differences in understandability or modifiability due to the combined effect of Pattern/Rules usage and Universe of Discourse of the diagram.

• *Alternative Hypothesis,* $H_{1\mathrm{PxUoD}}$: there are differences in understandability or modifiability due to the combined effect of Pattern/Rules usage and Universe of Discourse of the diagram.

3.5 Experimental Tasks and Treatment

We applied an experimental design of confounded factorial with interaction (see Table 2). In each experiment, the subjects were randomly divided into two groups: Group A, whose members received material set A, and Group B, who were given material set B. Each subject was required to answer the questionnaires and to carry out the modifications of each diagram. In total, each subject received four diagrams of four different UoD (two of them designed by applying rules and patterns and the other two without rules and patterns).

Table 2. Experimental Design

	Universe of Discourse			
	Aeronautics	Bank	University	Newspaper Agency
With Rules and Patterns	Group B	Group A	Group B	Group A
Without Rules and Patterns	Group A	Group B	Group A	Group B

4 Data Analysis and Interpretation

Once the task had been carried out, we collected the forms filled out by the subjects, checking that they were complete. As all the forms were indeed complete, all subject results were considered. Once the results were obtained, we first performed a descriptive analysis of the data. Tables 3 and 4 show the main descriptive statistics for the two experiments that were carried out.

Table 3. Understandability Results: Descriptive Statistics

Modifiability Measures	Experiments							
	1st Experiment (URJC)				2nd Experiment (UCLM)			
	Without Rules and Patterns		With Rules and Patterns		Without Rules and Patterns		With Rules and Patterns	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
Time (s)	261.7	154.274 57	251.050 0	182.689 12	130.635 1	82.304 91	182.837 8	113.586 97
Score of Modifications	1.25	0.786	1.05	0.686	1.44	0.773	1.32	0.622
Efficiency	0.003 717	0.002 689 3	0.006 515	0.005 773 8	0.011 100	0.012 320 6	0.010 345	0.069 646

Table 4. Modifiability Results: Descriptive Statistics

Understandability Measures	Experiments							
	1st Experiment (URJC)				2nd Experiment (UCLM)			
	Without Rules and Patterns		With Rules and Patterns		Without Rules and Patterns		With Rules and Patterns	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
Time (s)	238.8	179.170 19	243.6	116.511 31	92.081 1	58.077 36	129.837 8	80.541 66
No. of Right Answers	3.35	1.137	2.95	1.146	3.68	0.599	3.20	0.682
Efficiency	0.024 245	0.017 649 2	0.015 575	0.011 073 1	0.056 780	0.036 962 6	0.034 261	0.020 455 9

The results of the first experiment show that the time the subjects employed in understanding and answering the questions that were related to the diagrams in which rules and patterns were not used was slightly shorter than in the non-pattern diagrams. Moreover, the correctness of answers was 13.5% better (0.4 positive difference) and the efficiency was improved by 55% (0.008 67 better score) with regard to the use of rules and patterns in diagrams.

The replica at UCLM also confirmed a better performance; understanding diagrams with no rules and patterns usage, and in this case the differences were more significant. In fact, the time was improved by 37.7 seconds on average (58% less time), the correctness of answers was 0.48 (a 15 % improvement) and the effectiveness score went up to 0.022519 (65%).

With regard to modifiability, in the first experiment the effort required to carry out the tasks was slightly superior in those diagrams in which no rules and patterns were involved, but the level of correctness of answers was higher. The efficiency in this case was in favor of the use of rules and patterns. However, these results were not confirmed by the experiment at UCLM, where there were no important differences in usage or non-usage of patterns with regard to scores and efficiency. A greater difference was noticed only in relation to effort, i.e., the subjects spent less effort on diagrams which did not include rules and patterns.

The next step was to obtain insight into whether these differences were statistically significant. We therefore performed an ANOVA statistical test (significance level $\alpha = 0.05$) in order to analyze the interaction between the independent variables under study when the measurement of the dependent variables is repeated. The statistical results for the hypotheses-testing in relation to the efficiency measure of the dependent variables are summarized in Tables 5 and 6.

As we can observe in Tables 5 and 6, in relation to the dependent variable of understandability, we obtained evidence that:

• Design diagrams with rules and patterns were more difficult to understand than non-rule/pattern versions. These differences were confirmed as statistically significant ($p = 0.006$ and $0.000 < \alpha$, $H_{0\,\mathrm{Pu}}$ can be rejected in both experiments).

• The UoD also had an influence on the understandability efficiency, i.e., subjects obtained different understandability performances when they worked with different UoDs ($p = 0.000 < \alpha$, $H_{0\,\mathrm{UoD}}$ can be rejected in both experiments).

• There were no significant differences between the subjects in the two groups A and B. This is an indicator that the group to which the subjects belonged did not affect their easiness in understanding the task.

Table 5. ANOVA Results for Understandability Efficiency

		Sum of Squares	df	Mean Squared	F	Significance Level
Experiment URJC	Subject (Group)	0.002	8	0.000	2.367	0.046
	Error	0.003	26	0.000(b)		
	UoD	0.003	3	0.001	8.313	0.000
	Error	0.003	26	0.000(b)		
	Rules and Patterns	0.001	1	0.001	8.806	**0.006**
	Error	0.003	26	0.000(b)		
	Group	0.000	1	0.000	0.008	0.931
	Error	0.002	8	0.000(a)		
	Interaction	0.015	1	0.015	55.421	0.000
	Error	0.002	8	0.000(a)		
Experiment UCLM	Subject (Group)	0.036	35	0.001	1.844	0.009
	Error	0.059	107	0.001(b)		
	UoD	0.035	3	0.012	21.347	0.000
	Error	0.059	107	0.001(b)		
	Rules and Patterns	0.019	1	0.019	33.608	**0.000**
	Error	0.059	107	0.001(b)		
	Group	0.000	1	0.000	0.211	0.649
	Error	0.036	35	0.001(a)		
	Interaction	0.306	1	0.306	300.614	0.000
	Error	0.036	35	0.001(a)		

Note: a=MS (subject (group)). b=MS (error)

Table 6. ANOVA Results for Modifiability Efficiency

		Sum of Squares	df	Mean Squared	F	Significance Level
Experiment URJC	Subject (Group)	0.000	8	4.39×10^{-5}	2.391	0.044
	Error	0.000	26	1.84×10^{-5}(b)		
	UoD	0.001	3	0.000	10.128	0.000
	Error	0.000	26	1.84×10^{-5}(b)		
	Rules and Patterns	2.15×10^{-5}	1	2.15×10^{-5}	1.173	**0.289**
	Error	0.000	26	1.84×10^{-5}(b)		
	Group	2.25×10^{-5}	1	2.25×10^{-5}	0.513	0.494
	Error	0.000	8	4.39×10^{-5}(a)		
	Interaction	0.002	1	0.002	40.664	0.000
	Error	0.000	8	4.39×10^{-5}(a)		
Experiment UCLM	Subject (Group)	0.006	35	0.000	2.535	0.000
	Error	0.007	107	6.28×10^{-5}(b)		
	UoD	0.002	3	0.001	12.296	0.000
	Error	0.007	107	6.28×10^{-5}(b)		
	Rules and Patterns	1.19×10^{-5}	1	1.19×10^{-5}	0.189	**0.664**
	Error	0.007	107	6.28×10^{-5}		
	Group	2.03×10^{-5}	1	2.03×10^{-5}	0.128	0.723
	Error	0.006	35	0.000(a)		
	Interaction	0.017	1	0.017	106.663	0.000
	Error	0.006	35	0.000(a)		

Note: a=MS (subject (group)). b=MS (error)

In relation to modifiability, the differences between the use or non-use of rules and patterns were not statistically significant in the efficiency of either experiment. Only the different levels of effort in the modifications were found to be statistically significant.

Finally, in order to analyze the influence of the interaction of UoD and usage of rules and patterns on understandability and modifiability efficiency, we performed a second and specific ANOVA analysis, with a significance level α fixed to 0.05. As a result, no significant differences of understandability and modifiability efficiency were found to arise from the combined influence of the two independent variables: for understandability (first experiment, $p = 0.183 > \alpha$; second experiment $p = 0.062 > \alpha$: $H_{0\,\mathrm{PxUoD}}$ can be rejected in both experiments); for modifiability (first experiment, $p = 0.480 > \alpha$; second experiment $p = 0.887 > \alpha$: $H_{0\,\mathrm{PxUoD}}$ can be rejected in both experiments). Figs. 3 and 4 show the profile plots for the understandability efficiency in experiments 1 and 2, and Figs. 5 and 6 show profile plots for the modifiability efficiency. These are line plots which show whether there are interactions between the UoD (body of the plot) and the usage or non-usage of rules and patters (X axis) with regard to the understandability and modifiability efficiency (Y axis) respectively. As we can observe in the box plots of Figs. 3∼6, there is no interaction effect between the usage or non-usage of rules and patterns and the UoD. This is confirmed by the parallel lines which appear in all the profile plots, i.e., the levels of usage or non-usage of patterns and rules are constant throughout the levels of UoD. Figs. 3∼6 also demonstrate that the use of rules and patterns produced a worse deficiency, irrespective of the UoD.

This is a good insight into the usefulness, or otherwise, of rules and patterns for the improvement of

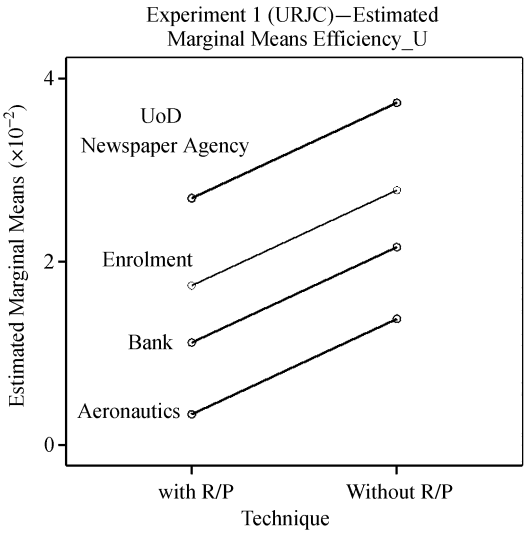


Fig.3. Box plot diagram of the interaction of UoD × usage of rules and patterns in the first experiment for understandability efficiency.

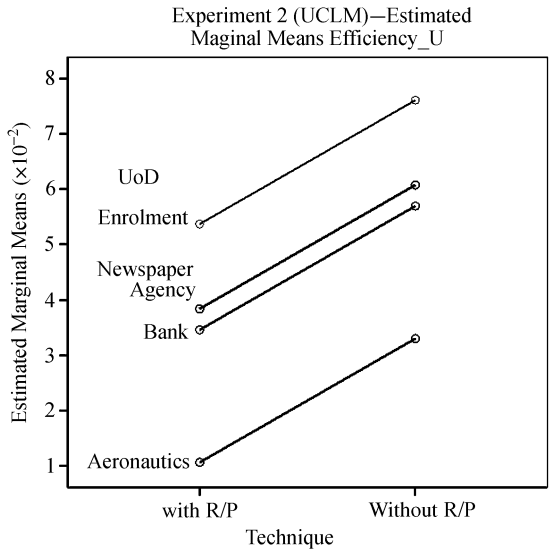


Fig.4. Box plot diagram of interaction of UoD × usage of rules and patterns in the second experiment for understandability efficiency.

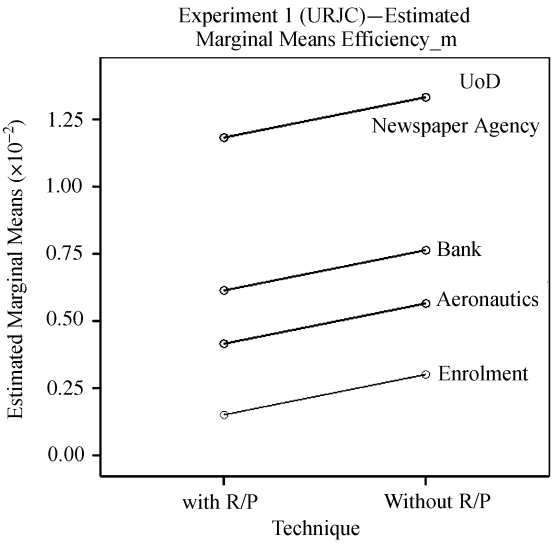


Fig.5. Box plot diagram of interaction of UoD × usage of rules, and rules & patterns in the first experiment for modifiability efficiency.

understandability and modifiability for all UoDs. It can also be deduced that different scores were obtained for each UoD, which, furthermore, indicates that the application domain may also affect the easiness of understanding and modification, as is confirmed by statistical analysis in which the significance level for UoD with regard to understandability and modifiability efficiency was 0.0 in both the experiments (UCLM and URJC) (Tables 5 and 6).

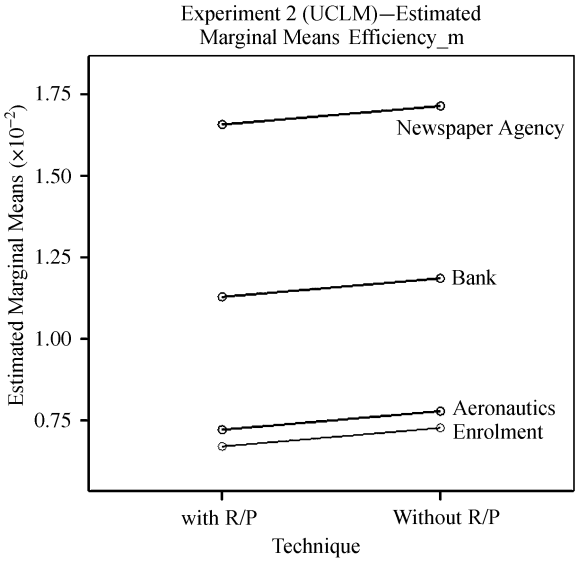


Fig.6. Box plot diagram of interaction of UoD × usage of rules, and rules & patterns in the second experiment for modifiability efficiency.

5 Experimental Validity Discussion

The various issues which might have threatened the validity of the results were analyzed during the planning process of the experiments.

Construct Validity. With regard to construct validity, the measures of the dependent variables aim at assessing the easiness with which the subjects understood and modified the provided designs. These measures (time, correctness, efficiency) are commonly used in studies which involve cognitive tasks[25], which is the nature of the current research, and are usually applied in empirical studies in which these dependent variables are evaluated[26,27]. Since these measures were obtained from the data collected on forms filled in by the subjects, the measurement may have lacked accuracy, i.e., the subjects may have failed to record the exact time at which they completed the task. To palliate this threat, we carried out a training session to address the correct use of the material and the process of collecting measurements in both experiments. We provided the subjects with an example of a completed form and explained the process carefully during the prior training session.

Internal Validity. The aspects that could threaten the internal validity were tackled in the following manner.

• *Persistence Effects.* The experiment was carried out with subjects who had never done a similar experiment before, thus avoiding persistence effects.

• *Learning Effects.* The universe of discourse of each

of the diagrams included in the material for each subject was different, and the subjects were given the diagrams in random order which alleviated this threat.

• *Knowledge of the Universe of Discourse* among UML models. The diagrams were from different universes of discourse but were sufficiently general and well-known to be familiar to the subjects. Consequently, knowledge of the domain did not affect the internal validity.

• *Fatigue Effects.* The average duration of the experiments was an hour, so fatigue effects did not appear.

• *Subject Motivation.* The subjects were motivated to participate, since the experiments were potentially beneficial to them. These beneficial features were related to both their preparation for the final examinations of the course (remembering that rules and patterns and UML are topics included in their regular syllabus) and to their daily work as future professionals.

• *Plagiarism* and *Influence Among and Between Subjects* were controlled by supervising the experimental runs.

External Validity. The material and tasks used were designed by bearing in mind the time restrictions, and were set at the level of an average student. However, the subjects were not professional designers and we would obviously have expected much better results if the subjects had been more experienced. The limited difficulty of the tasks and the different UoD's do, however, make the students suitable experimental subjects. Despite the above remarks on the students' lack of expertise, these individuals are much easier to work with than certain others. Further replications of these experiments using people already working in the field will, nevertheless, be considered. The purpose of those replications will be to confirm the results obtained. The diagrams of the material were relatively simple, due to the type of experimental settings and the amount of time available. In the future, more realistic diagrams (representative of real projects) must be used.

6 Concluding Remarks

The concept of quality has several interpretations. For software design, elements such as patterns or rules are associated with design maintainability quality. The relationship between maintainability and rules, and rules and patterns is not clear, however, and the positive or negative effects of these on maintenance quality is a controversial subject. In order to study this in detail, we broke up the maintainability concept into understandability and modifiability, and the design was that of the software artifact we were interested in evaluating.

In this paper, two experiments have been described. They were conducted to compare performances of designs with rules and patterns to alternatives in which these were not used. The experiments showed that design diagrams with rules and patterns were more difficult to understand than non-rule/pattern versions. In relation to modifiability, the differences between the use or non-use of rules and patterns were not statistically significant in the efficiency of either experiments. Only the differences at the levels of effort required in the modifications were found to be statistically significant.

We found evidence that design rules and patterns do indeed affect the design understandability, measured through efficiency. Designers, in making the decision to use patterns in a specific context, have to consider, on the basis of these results, whether the future improvements that patterns can provide to the design outweigh the added difficulties of understanding it. These difficulties in the understanding of UML designs in which patterns and rules are used may motivate specific extensions of the UML notations which will overcome this issue, as in the research of [28].

The research presented in this work therefore constitutes a good starting point for building up a complete body of knowledge concerning this issue. Furthermore, as the MDE paradigm has reinforced an emphasis on the importance of models, the positive and negative effects of using rules and patterns in design models may greatly contribute to building better applications from good quality models.

Confirming the results obtained, one focus of future research will be the replication of this empirical study in new settings, particularly those of industry, in which more realistic material and tasks will be considered.

References

[1] Agerbo E, Cornils A. How to preserve the benefits of design patterns. In *Proc. Conference on Object Oriented Programming, Systems, Languages, and Applications* (*OOPSLA'98*), Vancouver, Canada, Oct. 18–22, 1998, pp.134–143.

[2] Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1995.

[3] Coplien J, Schmidt D. Pattern Languages of Program Design. Addison-Wesley Publishing Company, 1995.

[4] Garzás J, Piattini M. An ontology for micro-architectural design knowledge. *IEEE Software,* 2005, 22(2): 28–33.

[5] Pescio C C. Principles versus patterns. *IEEE Computer,* 1997, 30(9): 130–131.

[6] Wiederhold G. What is your software worth? *Communications of the ACM,* 2006, 49(9): 65–75.

[7] Glass R L. Facts and Fallacies of Software Engineering. Addison Wesley, 2003.

[8] Reibing R. The impact of pattern use on design quality. In *Proc. OOPSLA Workshop Beyond Design: Patterns (Mis)used*, Tampa Bay, Florida, USA, Oct. 14–18, 2001.

[9] Bieman J, Jain D, Yang H. OO design patterns, design structure, and program changes: An industrial case study. In *Proc. 17th IEEE International Conference on Software Maintenance* (*ICSM'01*), Florence, Italy, Nov. 6–10, 2001, pp.580–589.

[10] Wendorff P. Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In *Proc. the Fifth European Conference on Software Maintenance and Reengineering* (*CSMR*), Lisbon, Portugal, IEEE Computer Society, March 14–16, 2001, pp.77–84.

[11] Bezivin J, Jouault F, Touzet D. Principles, standards and tools for model engineering. In *Proc. 10th IEEE International Conference on Engineering of Complex Computer Systems* (*ICECCS'2005*), Shanghai, China, June 20, 2005, pp.28–29.

[12] Selic B. The pragmatics of model-driven development. *IEEE Software,* 2003, 20(5): 19–25.

[13] Bieman J, Straw G, Wang H, Munger W, Alexander R. Design patterns and change proneness: An examination of five evolving systems. In *Proc. Ninth International Software Metrics Symposium*, Sidney, Australia, Sept. 3–5, 2003, pp.40–49.

[14] Prechelt L, Unger B, Tichy W F, Brössler P, Votta L G. A controlled experiment in maintenance comparing design patterns to simpler solutions. *IEEE Transactions on Software Engineering,* December, 2001, 27(12): 1134–1144.

[15] Masuda G, Sakamoto N, Ushijima K. Redesigning of an existing software using design patterns. In *Proc. the International Symposium on Principles of Software Evolution* (*ISPSE'00*), Kanazawa, Japan, Nov. 1–2, 2000, pp.165–169.

[16] Vokàc M. Defect frequency and design patterns: An empirical study of industrial code. *IEEE Transactions on Software Engineering,* December 2004, 30(12): 904–917.

[17] Elish M. Do structural design patterns promote design stability? In *Proc. the 30th Annual International Computer Software and Applications Conference* (*COMPSAC'06*), Chicago, USA, Sept. 17–21, 2006, pp.215–220.

[18] Ng T H, Cheung S C, Chan W K, Yu Y T. Do maintainers utilize deployed design patterns effectively? In *Proc. International Conference on Software Engineering* (*ICSE*), Minneapolis, USA, May 19–27, 2007, pp.168–177.

[19] Ng T H, Cheung S C, Chan W K, Yu Y T. Work experience versus refactoring to design patterns: A controlled experiment. In *Proc. SIGSOFT FSE*, Portland, USA, 2006, pp.12–22.

[20] Prechelt L, Philippsen M, Tichy W F. Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *IEEE Transaction of Software Engineering,* 2002, 28(6): 595–606.

[21] Ng T H, Cheung S C. Proactive views on concrete aspects: A pattern documentation approach for software evolution. In *Proc. 27th Annual International Computer Software and Applications Conference* (*COMPSAC'03*), Dallas, USA, Nov. 3–6, 2003, p.242.

[22] Ng T H, Cheung S C. Enhancing class commutability in the deployment of design patterns. *Information & Software Technology,* 2005, 47(12): 797–804.

[23] Garzás J, Piattini M (eds.). Object-Oriented Design Knowledge: Principles, Heuristics, Best Practices. Idea Group Inc, 2006.

[24] Martin R. Agile Software Development, Principles, Patterns, and Practices. Prentice Hall, 2003.

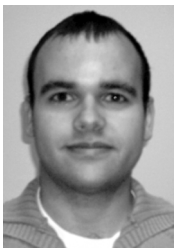
[25] Eysenck M, Keane M. Cognitive Psychology: A Student's Handbook. Lawrence Erlbaum Associates: Mahwah NJ, 2000, p.540.

[26] Genero M, Moody D, Piattini M. Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. *Journal of Software Maintenance and Evolution: Research and Practice,* 2005, 17(3): 225–246.

[27] Patig S. A practical guide to testing the understandability of notations. In *Proc. Fifth Asia-Pacific Conference on Conceptual Modelling* (*APCCM 2008*), Australia, January 2008, pp.49–55.

[28] Dong J. Adding pattern related information in structural and behavioral diagrams. *Information and Software Technology,* 2004, 46(5): 293–300.

Javier Garzás received his M.S. (2000) and Ph.D. (2004) degrees in computer science from the University of Castilla-La Mancha (UCLM). He is currently the Director of Kybele-Consulting and lecturer at Rey Juan Carlos University. His research interests include capability maturity model, object-oriented design, and software process and project management.



Félix García received his M.S. (2001) and Ph.D. (2004) degrees in computer science from the University of Castilla-La Mancha (UCLM). He is currently an associate professor in the Department of Information Technologies and Systems at the UCLM. His research interests include business process management, software processes, software measurement, and agile methods.

Mario Piattini received his M.S. (1989) and Ph.D. (1994) degrees in computer science and M.S. degree in psychology (2001), CISA and CISM (ISACA). He is the founder of Cronos Iberica, S.A., professor at UCM and U3CM, full professor at UCLM, director of the Information Technologies and Systems at UCLM. His current research interests are information systems quality, security and development.