Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 72 (2004) 129–143

The Journal of Systems and Software

www.elsevier.com/locate/jss

# A controlled experiment investigation of an object-oriented design heuristic for maintainability

Ignatios Deligiannis [a,*], Ioannis Stamelos [b,*], Lefteris Angelis [b], Manos Roumeliotis [c], Martin Shepperd [d]

[a] Department of Information Technology, Technological Education Institute, P.O. Box 14561 GR, T.K 54101, Thessaloniki, Greece

[b] Department of Informatics, Aristotle University of Thessaloniki Informatics, Aristotle University Campus, 54124, Thessaloniki, Greece

[c] Department of Applied Informatics, University of Macedonia, 54006 Egnatia 156, Thessaloniki, Greece

[d] Design, Engineering and Computing, Bournemouth University, Royal London House, Bournemouth, BH1 3LT, UK

Received 15 November 2002; received in revised form 5 June 2003; accepted 10 July 2003

**Abstract**

The study presented in this paper is a controlled experiment, aiming at investigating the impact of a design heuristic, dealing with the 'god class' problem, on the maintainability of object-oriented designs. In other words, we wish to better understand to what extent a specific design heuristic contributes to the quality of designs developed. The experiment has been conducted using undergraduate students as subjects, performing on two system designs using the Coad & Yourdon method. The results of this study provide evidence that the investigated design heuristic: (a) affects the evolution of design structures; and (b) considerably affects the way participants apply the inheritance mechanism.

*Keywords:* Object-oriented; Maintainability; Experiment; Design; Heuristics

## 1. Introduction

In the last decade or more, the object-oriented (OO) paradigm has gained a broad acceptance within the software community, the industry, and the software development organizations of any size. Hence, OO methodologies, languages and development environments, have been developed supporting this technology. It has mainly been made popular by C++, and now even more so by Java. Jones (1994) has noticed the rapid growth of OO technology, since 1994. Even recently, a survey carried out by Goodley (1999), has indicated the continuing growth in the popularity of Java as an OO development language.

Much of the literature asserts that substantial gains, such as increased understandability, productivity, quality, ease of modification, reuse, should accrue from using OO analysis, design, coding, and reusable components. Nevertheless, these benefits are mostly based on intuition and not on empirical evidence. Intuition may provide a starting point, but it needs to be backed up with empirical evidence. According to Basili and Burgess (1995), experimentation has shown that intuition about software in many cases is wrong. Jones (1994) as well, has identified a lack of empirical evidence to support the claims accredited to OO technology, like improved productivity and quality, defect removal efficiency, and even more so reusability.

For this reason, over recent years, there has been a growing interest in empirical evaluation. In a review (Deligiannis et al., 2002), examining how experimentation has been carried out in the OO technology, the evidence does not support the claim that OO techniques always provide the benefits accredited to them. A case study performed by Hatton (1998) on corrective maintenance issues, indicated a number of concerns whether OO technology has met its claims. Considering the performance and strategies of programmers new to OO

* Corresponding authors. Tel./fax: +30-2310-791295 (I. Deligiannis); Tel.: +302310998227; fax: +302310998419 (I. Stamelos).

*E-mail addresses:* igndel@it.teithe.gr (I. Deligiannis), stamelos@csd.auth.gr (I. Stamelos), lef@csd.auth.gr (L. Angelis), manos@uom.gr (M. Roumeliotis), mshepper@bmth.ac.uk (M. Shepperd).

technology it was found that OO concepts were not easy to learn and use quickly (Hillegersberg et al., 1995).

Clearly, more empirical research is needed to investigate these claims; examining in particular the efficacy and effectiveness with which the various OO features are applied. One particular area that warrants thorough investigation is considered the impact of design heuristics on one quality factor, namely maintainability. The reasons leading us to this decision are: (a) the maintenance phase is considered the most costly phase of system life cycle (Meyer, 1997; Hatton, 1998). Subsequently, investigations aiming to reducing it should be of increased interest for the empirical research community; and (b) design heuristics, concentrating cumulative knowledge mainly based on experience and practice (Booch, 1995), might provide useful assistance to achieve this goal.

The study presented in this paper is a controlled experiment, carried out using students as participants, which investigates the impact of a single design heuristic on quality of OO designs, captured by two important quality factors of design assessment, namely *understandability*, and *maintainability*. The paper is structured as follows. Section 2 describes the details of the experiment. Section 3 summarizes the data collected and presents the data analysis results. Section 4 identifies and discusses possible threats to the validity of the study. Finally, Section 5, presents the conclusions and future research targets. The structure of the paper follows the structure of a Briand et al.'s paper (2001).

## 2. Description of the experiment

This research builds upon the results of a previous observational study (Deligiannis et al., 2003), which investigated the impact of an OO design heuristic, namely "god class problem" (it is described in Section 2.1), with respect to the maintainability. It was also aiming at investigating the impact of a design heuristic on the maintainability of OO designs, as well as the relationship between that OO design heuristic and metrics, i.e. whether we are able to capture a specific design heuristic by applying a suitable subset of design metrics. The results provided evidence that: (a) the investigated design heuristic considerably affects the performance of the participants; (b) it affects the evolution of design structures; and (c) there is a considerable relationship between that design heuristic and metrics so that it could be feasible to conduct an assessment by using appropriate metrics.

### *2.1. "God class problem"*

The term 'heuristic' is widely used in almost every field of science. We found two meanings of this term in the literature, namely 'rule of thumb', and 'trial and error'. The term 'rule of thumb' refers to the accumulated and distilled knowledge gained from the experience, while the term 'trial and error' refers to *unselective search* that encompasses a high risk to fail (Glass, 2002). The computer science field has traditionally defined heuristics as rules of thumb. However, we consider that both terms are interrelated in that knowledge and experience could not been gained without prior 'trial and error'. From both meanings it is implied that heuristics are not hard and fast rules, since a heuristic violation in itself may not represent a problem, perhaps due to differences in context.

There is a plethora of OO design heuristics proposed in the literature (Lorenz and Kidd, 1994; Booch, 1995; Firesmith, 1995; Jacobson and Christerson, 1995; Riel, 1996) related to a multitude of software aspects. Kirsopp and Shepperd (2001) distinguish them into *syntactic*, *semantic*, and *hybrid*. To this list of categories we would like to add a new one—*structural*—mainly based on structural aspects of a system. There are two basic kinds of structures (Coad and Yourdon, 1991; Rumbaugh et al., 1991; Booch, 1994). *Generalization–Specialization*, in which some abstractions adopt and extend the properties of others, thus forming inheritance hierarchies, and *Whole–Part* structures, where one abstraction contains or is composed of other abstractions (composition), or where one abstraction plays a master role to another's detail role (aggregation). To justify our suggestion, we consider it would be helpful to mention a typical example of a heuristic belonging to this category, proposed by Riel (1996):

> Objects which share lexical scope, i.e. those contained in the same containing class, should not have 'uses' relationships between them.

Actually this heuristic specifies the manner the internal objects of a containment class should communicate to each other. The author suggests that they should not directly communicate to each other, rather the container class should manage their communication. The reasoning behind is reusability.

Heuristics are aimed at enhancing software quality. However, when they are violated there is a risk of producing complex and monolithic design structures. In this study, we are focusing on a single but important design heuristic. According to Riel (1996), there is one very distinct area where the OO paradigm can drive designs in a dangerous direction. This is called the 'god class problem' and deals with poorly distributed system intelligence. It is caused by a common error among process-oriented developers in the process of moving to the OO paradigm. These developers attempt to capture the central control mechanism, so prevalent in the process-oriented paradigm, within their OO design. The result is the creation of a 'god' object that performs most of the work. Riel considers we should work toward the avoidance of such classes. This is described as follows:

Do not create god classes/objects in your system. Be very suspicious of a class whose name contains Driver, Manager, System, or Subsystem.

We consider it belongs to the structural category, since it mainly captures structural design properties like: cohesion, internal and external coupling. According to Allen and Khoshgoftaar (1999) cohesion is based on intramodule coupling, normalized to the interval zero and one. However, a "god class" might as well share the other heuristic categories, mentioned above. A striking example found in the literature, identifiable by its extraordinary number of 172 methods, was examined in a case study (Chaumun et al., 2002). Syntactic category differs from the structural in that it mainly captures syntactic aspects of a design, as seen in the following examples:

Usually when a service requires more than three parameters something is wrong (Coad, 1991).
Do not use protected data, it weakens data hiding. Use protected access functions instead. All data in the base class should be private (Riel, 1994).

Our motivation for the selection of the investigated heuristic is the following: (a) it captures both structural and modular design aspects; (b) it is frequently violated as it is an easy trap for designers with a 'procedural' mindset (Riel, 1996); and (c) it includes some control mechanism. Nevertheless, building control objects is encouraged by some authors (Jacobson and Christerson, 1995; Riel, 1996). According to the object classification stated by Jacobson and Christerson (1995), we should use three kinds of objects: *interface*, *control*, and *entity* objects. Control objects take care of the most complex behaviour. However, in cases where control objects try to do too much of the work themselves, instead of putting the responsibilities where they belong, they become more complex and harder to maintain.

## 2.2. The experiment

Since this study concerns a controlled experiment, we consider emphasis should be placed on two points. First, experimental settings should be as realistic as possible applying to those in practice. Second, planning, operation, and analysis should be according to those proposed in the literature (Wohlin et al., 2000).

The present study is considered to be a controlled experiment. Key motivator for using formal experimentation is that the results can be more easily generalised than those of an observational or a case study. It concentrates on investigation of the same design heuristic, this time additionally capturing an OO feature, the inheritance mechanism. Inheritance is a fundamental feature in OO programming languages; it is supposed to make systems easier to design and make software components easier to reuse (Bieman and Xia Zhao, 1995). The rationale behind the decision to include this feature was that Bieman and Kang (1995), in a case study they carried out, found significant relationship between inheritance and cohesion. While, in the observational study mentioned above, it was found that cohesion significantly contributed to the 'god class' structure.

Our decision to concentrate on the 'god class problem' was based on the fact that a 'god class' mostly suffering from low cohesion and high coupling, two particularly important quality properties of a design structure, could also be used in an inheritance hierarchy, exploiting the benefits provided by this mechanism, namely reusability and extensibility. Therefore, our interest was to examine the impact of the specific heuristic with respect to inheritance on maintainability of OO designs. Clearly, this choice highlights the emphasis we place on a class structure as well as its dependencies. According to Booch (1995), it is crucial to identify sound classes, as well as their dependencies, which reveal the implication of change—a change in one class will potentially change dependent classes. The importance of dependencies on system maintenance and change is also stressed by Jacobson and Christerson (1995).

The investigated hypotheses (Section 2.4) are stated on the basis that a system designed according to design heuristics is more understandable and maintainable, than a system violating them. In this study, only one heuristic is violated, applied in a specific part of the design, rendering the results more controllable and stronger. Briand et al. (2001) suggest the necessity of investigation of each and every design principle independently, which could lead to better understanding the impact of each principle on various design aspects.

## 2.3. Assessment

Apart from their guidance role in development, heuristics provide an additional means for design assessment. According to Kirsopp and Shepperd (2001) they can be used in two distinct ways: *stand-alone* and *comparative*. In order to assess the benefits provided by the investigated heuristic, as well as its impact on the quality factors—understandability and maintainability—in which this study is most interested, quantitative and qualitative analysis are also required.

Whitmire (1997) argues that, objective criteria depend upon data gathered through measurement. He states that measurement is a powerful and useful design tool and sustains that a valid measurement context can be built from any of the following three points of view:

*strategic*, *tactical* and *technical*. These points of view form one basis of a framework for measurement. The other basis for the framework is a set of four basic objects for which measurement data can be collected: *processes*, *products*, *resources*, and *projects*. Furthermore, each class of object has two types of attributes: *internal* and *external* (Fenton and Pfleeger, 1997).

Within this measurement framework, this study is most closely related to a *technical point of view*, and to object *products*. For products the following external attributes are measured: *understandability* and *maintainability*. Understandability is captured by a *debriefing questionnaire* where the participants express their personal opinions regarding architectural aspects of the systems, as well as the modification task (Fig. 3, Table 1) and is subjective in nature. Maintainability assessment is performed by the following means: (i) by assessing three basic criteria for OO design quality assessment (product internal attributes): *completeness*, *correctness*, *consistency*; and (ii) by considering *conformance* to design heuristics. All of them provide objective measurements.

The first three criteria, suggested by Laitenberger et al. (2000), aim at assessing the quality of design diagrams. These provide a means for systematic design evaluation. Each one of these, according to the authors, can be operationalised as follows:

| | |
|---|---|
| *Completeness*. | A diagram or a set of diagrams is complete if no required elements are missing. |
| *Correctness*. | A diagram or a set of diagrams is correct if it is judged to be equivalent to some reference standard that is assumed to be an infallible source of truth. |
| *Consistency*. | A diagram or a set of diagrams is consistent if there are no contradictions among its elements. |

Based on these criteria, Laitenberger et al. performed an empirical investigation comparing two inspection techniques, Checklist, and Perspective Based Reading (PBR), finding PBR to be more effective in evaluating design diagrams in defect detection. Hence, the PBR technique was chosen for evaluating the above criteria in the present experiment (Fig. 1). From the PBR technique, a number of elements were used, that best conform to the delivered solutions, in our case Appendix A.

## 2.4. Hypotheses

For hypotheses testing the design documents of two functionally equivalent versions of a system, called Tariff Selector System (TSS), have been used. Design A was the heuristic compliant version, while Design B was the heuristic non-compliant one (Fig. 2(a) and (b)). In order to investigate the impact of the specific design heuristic on the two quality factors, standard significance testing was used, the null hypothesis being stated as:

$H_0$. There is no difference between the heuristic compliant and the heuristic non-compliant OO design in terms of understandability and maintainability.

The alternative hypotheses, i.e., what was expected to occur, were then created as:

$H_1$. The heuristic compliant OO design is significantly easier to *understand* than the heuristic non-compliant OO design.

$H_2$. It is *easier to maintain* a heuristic compliant OO design than a heuristic non-compliant OO design.

- $H_{2a}$. Higher *completeness* can be achieved maintaining a heuristic compliant OO design than a heuristic non-compliant OO design.
- $H_{2b}$. Higher *correctness* can be achieved maintaining a heuristic compliant OO design than a heuristic non-compliant OO design.
- $H_{2c}$. Higher *consistency* can be achieved maintaining a heuristic compliant OO design than a heuristic non-compliant OO design.

$H_3$. Further *heuristic compliance* can be achieved maintaining a heuristic compliant OO design than a heuristic non-compliant OO design.

The term ‘maintain’ mentioned above refers to the proper use of the existent or new required design members (e.g., class, attribute, method, association), which are classified as: required (according to the modification task), declared, missed, incorrect (e.g. ‘uses’ or ‘message connection’ when ‘whole-part’ structure is required), and inconsistent (duplicated members or deleted methods—i.e. inherited but not used), (see also Sharble and Cohen, 1993).

## 2.5. Subjects

Twenty students were used as participants to perform maintenance tasks on the two system designs A and B. All were undergraduate students at the second year of their studies of the Department of Informatics at the Aristotle University, Thessaloniki, Greece, enrolled in the class of OO Programming. During the class semester, the students were taught basic OO principles as those suggested by Coad and Yourdon (1991). Each lecture was supplemented by a practical session providing the opportunity to the students to make use of the concepts they had learned.

At about the end of the semester, students were asked to participate voluntarily in an experiment planned to run at the end of the semester, making it clear that they would gain an additional bonus to their final evaluation, in order to motivate them. As a result 28 students agreed to take part. Then they were asked to attend two

Table 1
Debriefing questionnaire

| Questions | | | Group A | | | | | | | | | | Group B | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | *Experience* (1 = poor–5 = professional) | With design documents in general | 2 | 1 | 2 | 1 | 1 | 2 | 4 | 3 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 2 | 3 | 1 | 3 | 2 |
| 2 | | With OO design documents | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 2 | 2 | 1 |
| 3 | | In programming languages | 4 | 4 | 4 | 2 | 4 | 4 | 4 | 4 | 3 | 3 | 5 | 4 | 4 | 2 | 2 | 3 | 4 | 4 | 4 | 4 |
| 4 | *Understandability* (1 = little–10 = high) | How well you *understood* what was required of you | 8 | 8 | 6 | 6 | 10 | 8 | 8 | 8 | 4 | 8 | 8 | 6 | 10 | 8 | 6 | 6 | 8 | 6 | 8 | 8 |
| 5 | | Estimate, in terms of *understandability*, the quality of the design documents | 8 | 7 | 5 | 7 | 8 | 7 | 9 | 7 | 7 | 7 | 8 | 3 | 6 | 9 | 5 | 5 | 7 | 7 | 8 | 8 |
| 6 | | Estimate your overall *understanding* of the design documents | 5 | 9 | 5 | 6 | 8 | 7 | 5 | 8 | 5 | 8 | 5 | 3 | 6 | 8 | 5 | 3 | 7 | 5 | 7 | 8 |
| 7 | | What caused you the most difficulty to *understand* the design documents | | | | | | | | | | | | | | | | | | | | |
| | | (a) Nothing in particular | × | | | | | | | × | | | | | × | × | | | × | | × | |
| | | (b) Cohesion in classes | | | × | | | × | × | | | | | | | | × | × | | | | |
| | | (c) Coupling between classes | | × | | × | × | | | | × | × | × | × | | | | | | × | | × |
| 8 | Performance (1 = low–10 = high) | Estimate the correctness of your solution to the modification tasks | 6 | 8 | 6 | 6 | 8 | 6 | 10 | 8 | 4 | 6 | 6 | 4 | 8 | 6 | 4 | 2 | 10 | 6 | 8 | 10 |
| 9 | | If you could not *complete* all the tasks, please indicate why | | | | | | | | | | | | | | | | | | | | |
| | | (a) Ran out of time | × | | | | | | | | × | | | | | | | | × | | | |
| | | (b) Did not fully understand the system requirements | | | | | | | | | | | | | | | | | | | | |
| | | (c) Did not fully understand the design documents | | | | | × | | | | | | | | | | × | × | | × | | |
| | | (d) Did not fully understand the modification task | | | | | | | | | | | | × | | | | | | | | |
| 10 | | Estimate the overall *difficulty* of the tasks you have been asked to *perform* | 5 | 10 | 7 | 7 | 9 | 2 | 6 | 4 | 4 | 10 | 7 | 9 | 5 | 5 | 10 | 9 | 4 | 6 | 3 | 2 |
| 11 | | What caused the most *difficulty* to perform the *modification* task on the design | | | | | | | | | | | | | | | | | | | | |
| | | (a) Nothing in particular | | | | | | | × | | | | | | | | | | × | | | |
| | | (b) Cohesion in classes | | | × | | | | | | | | × | | × | | × | × | | | × | |
| | | (c) Coupling between classes | × | × | | × | × | × | | × | × | × | | × | | × | | | | × | | × |
| 12 | | Estimate your *difficulty* to locate where to *accommodate* the new functionality | 4 | 2 | 8 | 3 | 7 | 3 | 5 | 5 | 5 | 6 | 2 | 9 | 7 | 5 | 8 | 5 | 4 | 5 | 4 | 3 |
| 13 | | Are you a mobile phone user (yes/no)? | n | n | y | y | y | y | y | y | n | y | y | y | y | y | y | n | n | y | n | y |

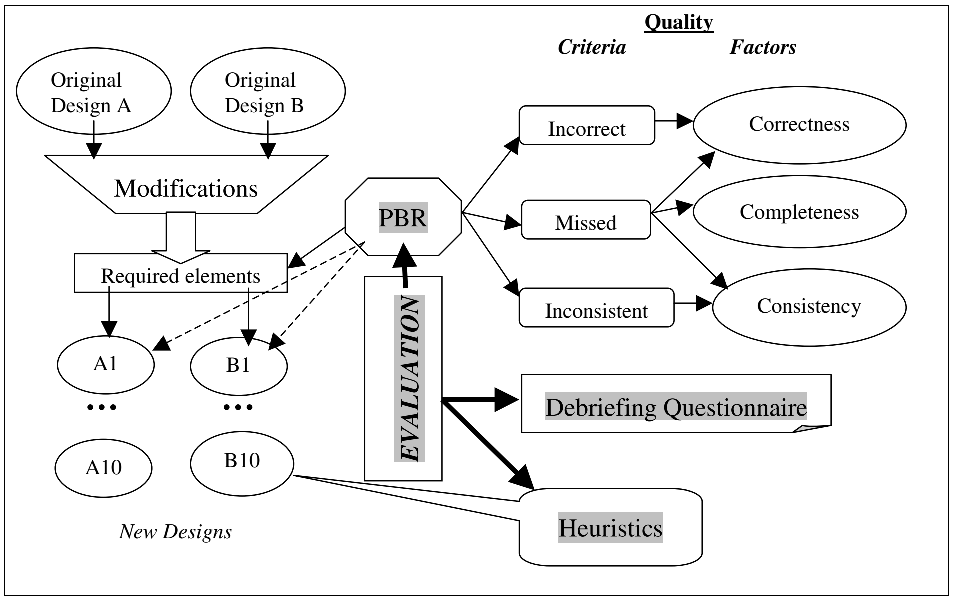

Fig. 1. The evaluation model.

additional lectures where some related concepts to the experiment were taught. A few days before the experiment took place, they were asked to undertake a small test in order to evaluate their ability to participate in the experiment. They were given a small system consisting of a number of classes, with no relationships between them. The students were asked to design the class relationships according to a written requirements text. We were interested to check whether they could efficiently perform on inheritance, aggregation, association and cardinality issues. Finally, 20 of them were chosen to participate, randomly assigned to two groups, A and B. In the rest of the paper we call them as A1–A10 and B1–B10 for convenience, according to the system they worked with. Ten were given the version compliant to the design heuristic (Design A). The other 10 were given the non-compliant version (Design B), which violates the design heuristic under investigation. They had no prior practical experience on design documents, apart from what they had learned during the semester, as shown on Table 1. It is also worthwhile to note that most of the participants are mobile phone users, thus quite familiar with the whole functionality of the 'Tariff Selector System', used for this study (Table 1).

### 2.6. Experimental material

The application domain used for the system was a Tariff Selector System (TSS); a small but realistic (i.e. not fictitious) application, in the sense that it could be easily implemented and used as a real application. The system implemented a mobile phone tariff selector system to aid prospective buyers of telephones, in determining which tariff best matches their needs. Each system documentation was six pages long and included the requirements documents and the class design diagrams. Design A consists of 18 classes while Design B of 17. Design A is considered to be the 'heuristic compliant' one (Fig. 2(a)), due to its decentralized structure. It was designed according to the above-mentioned design heuristic. Extra care was also taken, to be consistent with some other properties for well-designed classes, namely coupling, cohesion, sufficiency, completeness, and primitiveness (Booch, 1994). Design B, considered as the 'heuristic non-compliant' one, was designed according to Design A but violating the design heuristic under investigation (Fig. 2b). The basic concept was to build a class playing a central role into the system, as described above. Two independent professionals examined both designs and asserted their correctness and consistency.

The only difference between the two designs was that one class in Design B, the 'god class', captured the central functionality of the system. That class constituted of two entire classes and a method from a third class, all taken from Design A. In Design A the same functionality was split into three classes, each capturing a coherent functionality. One of these three classes still implemented control functionality, however being consistent with a few related heuristics, namely (i) "A class services should be related to each other and to the class in an intuitively satisfying way" (Coplien, 1992) and (ii) "Avoid having a large, central object which controls most of the functionality" (Riel, 1994). It is worthwhile to mention here, that in a previous study (Deligiannis et al., 2003) it was shown that, by applying a subset of three metrics suites (Chidamber and Kemerer, 1994; Lorenz and Kidd, 1994; Abreu and Melo, 1996), it could

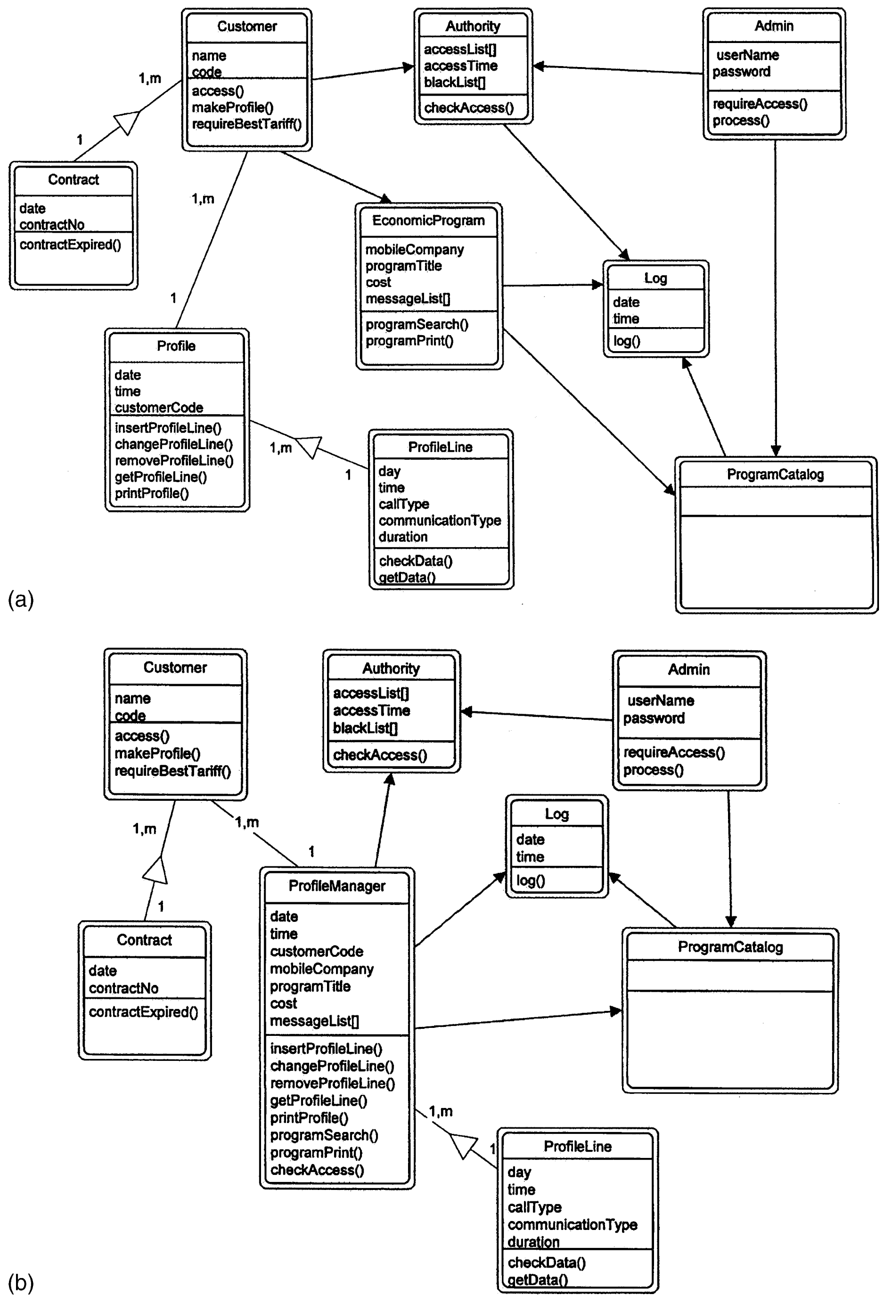

Fig. 2. (a) Design A ('heuristic compliant') (b) Design B ('heuristic non-compliant').

be feasible to locate a part of a design where violation of the heuristic under investigation has taken place.

### 2.7. Experimental tasks

There were two tasks to be performed by the participants. First, they were given the documentation set in order to apply a modification task (Fig. 3). This task was designed to match as much as possible a maintenance task in a real situation. The required modification was to add new functionality mostly affecting the part of the system under investigation.

Namely, a mobile user (customer) would be given the ability to request, after his contract expired, a new 'best tariff', based on calls he had done during the past year. This would serve as a consultation to his decision, before signing a new contract, suggesting the most beneficial company and tariff. Thus, a new modified class diagram was expected from each participant, along with additional information explaining their solution.

The total elements to be added were approximately 15 attributes, and 14 methods. A considerable part of them were already present in their designs. Although not explicitly mentioned, it was implied that applying

**Modification task**

**Asking for a tariff order based on an expired contract's performed calls**

When a customer's contract expired, after one year, he could request for a new best tariff this time based on his performed calls during the past year. Thus, a new contract could be signed according to his past mobile usage.

To fulfill such a request the system had first to request from the Mobile Phone Company's Billing system, the customer's performed calls for the specific contract, and then create a new 'Call profile' based on them from which the best tariff should be built. Also, its cost as well as the cost of the performed calls should be provided, in order to make feasible a comparison.

Fig. 3. The modification task.

inheritance these elements could be reused. Alternatively, they could be implemented explicitly, possibly causing duplication of their usage. Inheritance was a reasonable choice since the previous functionality and the new required differed in that the old one built a 'user profile' by the user himself, while the new one built a 'call profile' from the users' performed calls.

The second task was to complete a debriefing questionnaire, which captured, (i) their personal details and experience, (ii) their understanding of the system, (iii) their understanding of the modification task, (v) the structure of the design, and (vi) their motivation and performance, e.g., how accurate and complete they thought their solution was, what had caused them the most difficulty (Table 1).

## 2.8. Procedures

A week before the experiment proper took place, the volunteering participants were asked to perform a short test in order to collect the most capable of them to participate in the experiment. They were told that only a number of them, the most correctly performing, would participate in the experiment. The test questions were related to various design concepts, e.g., inheritance, aggregation, composition (Deligiannis et al., 2002), that were applied within the experiment designs. Also, a dry run was performed using two Ph.D. students, in order to test the experimental settings before the experiment proper took place.

The experiment was performed in a classroom, where each participant had plenty of space to examine all the available documentation. They were randomly placed in the work places. Each participant received a documentation set different than the one examined by the person sitting next to him. This was performed to eliminate plagiarism, although it was not a real concern.

The participants were told verbally that there were different designs being worked upon, but they were not told anything about the nature of the study, i.e., what hypotheses were being tested. However, they were told to complete their tasks correctly, exploiting efficiently their time. They were given a maximum of one and a half hours to complete all the tasks. They were also told that any questions could be directed towards the two experiment monitors. Nevertheless, questions were not answered if thought to assist participants' performance, only explanations were provided to them, avoiding affecting their performance. After completing their tasks, they called one of the experiment monitors to collect their documentation. This was necessary in order to mark their performance time as well as avoiding any further changing. Then they were given a debriefing questionnaire to complete, expressing their subjective opinion concerning a number of issues as described above.

## 2.9. Design

We applied statistical analysis on the collected data to interpret the results in order to draw meaningful conclusions from the experiment. The choice of the design affects the data analysis and vice versa. In this study the type of design applied is a *completely randomized balanced design* (Wohlin et al., 2000). The independent variable is the *design heuristic* under investigation. The experiment context is characterized as *multi-test within object study* (Wohlin et al., 2000, p. 43). An advantage of using such a design is that it simplifies the statistical analysis.

## 2.10. Dependent variables

For hypotheses testing a number of dependent variables should be examined as follows:

*Understandability*, is captured by a *questionnaire* where the participants express their personal opinions regarding design aspects and the modification task, and is subjective in nature. It represents the degree of understanding measured by the grades of participants' answers to related questions.

*Maintainability*, is assessed by three basic criteria for OO design quality assessment: *completeness*, namely whether they completed the functionality needed as required by the modification task; *correctness*, based on the former criterion, namely the proportion of the completed members that were also correct; and *consistency*, based also on the completeness criterion, namely the proportion of the completed design elements that were consistent. In order to measure them, we first need to define a few variables (a design element may be a method, an attribute, an association, etc.)

$$\begin{aligned} \mathit{Declared\ elmts}\ (\mathit{DECL}) = {} & \mathit{new\ elmts} + \mathit{inherited\ elmts} \\ & + \mathit{reused\ elmts} \end{aligned}$$

$$\mathit{Required\ elmts}\ (\mathit{REL}) = \mathit{DECL} + \mathit{missed\ elmts}\ (\mathit{MEL})$$

Note, that required elements (REL) depend on requirement definition of the modification task, not on some expected solution.

*Completeness* is measured based on the number of completed design elements, according to the requirements definition. They were: (a) the total number of required elements REL; and (b) the missed elements MEL to complete the tasks:

$$\begin{aligned} \mathit{COMPLETENESS} &= (1 - (\mathit{MEL}/\mathit{REL})) * 100 \\ &= (\mathit{DECL}/\mathit{REL}) * 100. \end{aligned}$$

*Correctness* represents the correct usage of the above described elements, on the basis of the completeness achieved.

$$\begin{aligned} \mathit{CORRECTNESS} &= (1 - (\mathit{incorrect\ elmts}/\mathit{DECL})) \\ &\quad * \mathit{COMPLETENESS} \\ &= ((\mathit{DECL} - \mathit{incorrect\ elmts})/\mathit{REL}) \\ &\quad * 100 \end{aligned}$$

*Consistency* represents the consistent usage of the completed design elements. As INCONSISTENT elements are considered the following: duplicated, redundant (attributes), empty (methods), and inconsistently declared. We also take into account the achieved COMPLETENESS:

$$\begin{aligned} \mathit{INCONSISTENT} = {} & \mathit{duplicated\ elmts} \\ & + \mathit{redundant\ attributes} \\ & + \mathit{empty\ methods} \\ & + \mathit{inconsistently\ declared\ elmts} \end{aligned}$$

$$\begin{aligned} \mathit{CONSISTENCY} &= (1 - (\mathit{INCONSISTENT}/\mathit{DECL})) \\ &\quad * \mathit{COMPLETENESS} \\ &= ((\mathit{DECL} - \mathit{INCONSISTENT})/\mathit{REL}) \\ &\quad * 100 \end{aligned}$$

*Heuristic compliance*. The delivered designs were inspected to decide whether they violated other heuristics. A possible case where some violation could occur, was the one that offered some inheritance opportunity. Namely whether, if they did apply inheritance, they did it so properly. A proper heuristic for such case is that measured by DELETED METHODS:

> Subclasses should not delete features. Subclasses that delete features probably are not specializations of their superclass(es) because they cannot do something that their parent(s) can (Firesmith, 1995).

All but the dependent variable understandability provide objective measures. Each one is operationalised as following: Understandability is measured by the questions 4–7 in the debriefing questionnaire. Maintainability is measured by the three quality criteria, mentioned above. Their measurements are based on examining the required and delivered design members quantified according to the above formulas. Finally, heuristic compliance is measured by the number of deleted members (unused attributes and methods). Such members are considered those inherited while not making sense in the new derived subclass.

## 3. Experimental results

### 3.1. Statistical analysis of the data

For the statistical analysis of the data we used variables related to the answers provided by the students in the questionnaire and variables related to their performance in the experiment. The statistical methods we used to test the hypotheses were: the Student's *t*-test for independent samples and the non-parametric Mann–Whitney (M–W) test in order to test significant differences between the two groups (A and B) for each individual variable. In order to examine differences between the two groups by considering many variables simultaneously, we also used the multivariate method known as discriminant analysis.

Regarding *understandability*, none of the above methods showed significant difference between the two groups. Thus there is no evidence for supporting $H_1$ hypothesis. Regarding *maintainability*, the variables MEL, INCONSISTENT, COMPLETENESS, CONSISTENCY and CORRECTNESS were tested individually using the *t*-test and the M–W test. Differences were found by both tests for variable MEL ($p < 0.1$) and by M–W test for variables INCONSISTENT ($p < 0.05$) and CONSISTENCY ($p < 0.1$). At a significance $0.1 < p < 0.2$, there is some evidence of difference between the two groups for the variables

COMPLETENESS and CORRECTNESS by both tests and for CONSISTENCY by the *t*-test. The differences of all variables related to maintainability between the two groups are shown by boxplots in the following Figs. 4–8.

The discriminant analysis (DA) was used in our study in order to build a model of group membership based on the variables characterizing maintainability. A discriminant function (DF) was generated based on a linear combination of the five variables from the results of the experiment where the membership is already known.

The DA is used not only to test the equality of the two groups but also to search a way in which the two groups can be distinguished. It is important to keep in mind that DA considers all the variables simultaneously and not individually. So, it is a statistical procedure that describes and interprets the differences between the two

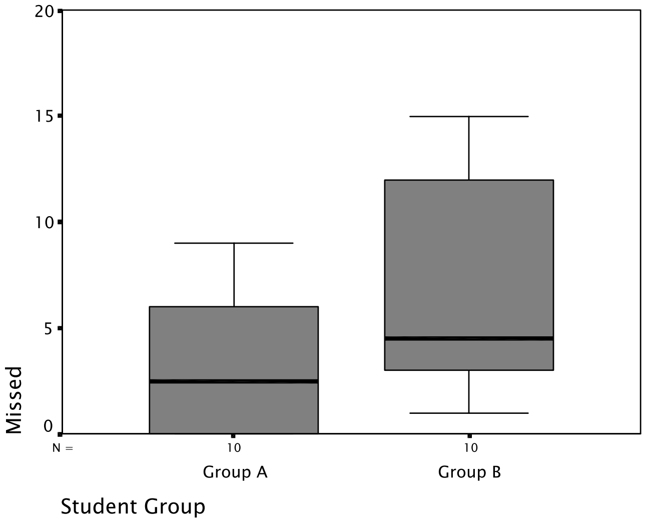

Fig. 4. Missed elements.

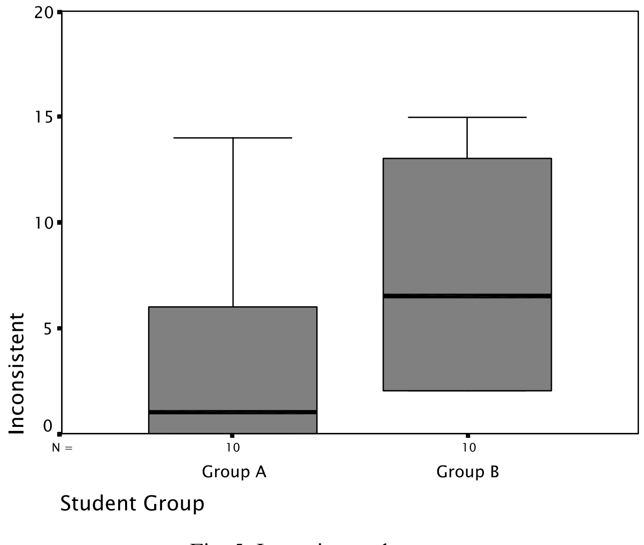

Fig. 5. Inconsistent elements.

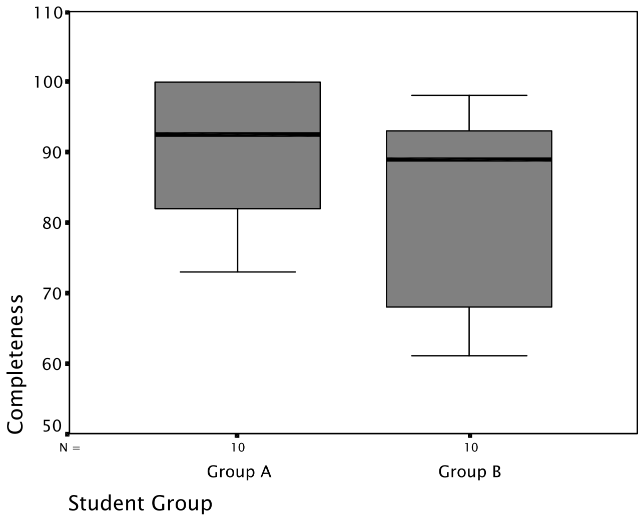

Fig. 6. Completeness.

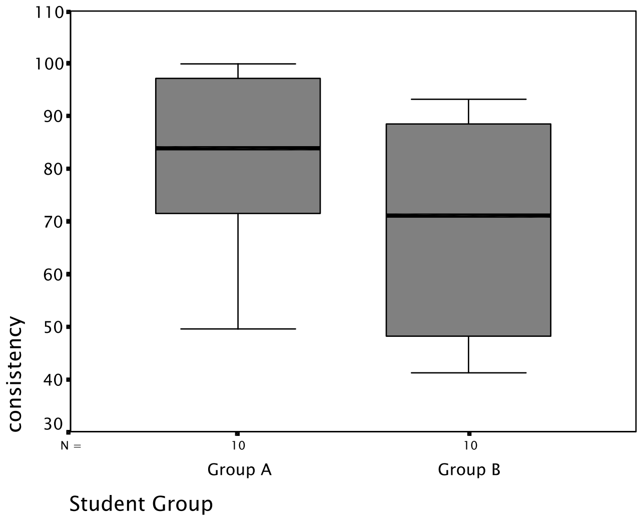

Fig. 7. Consistency.

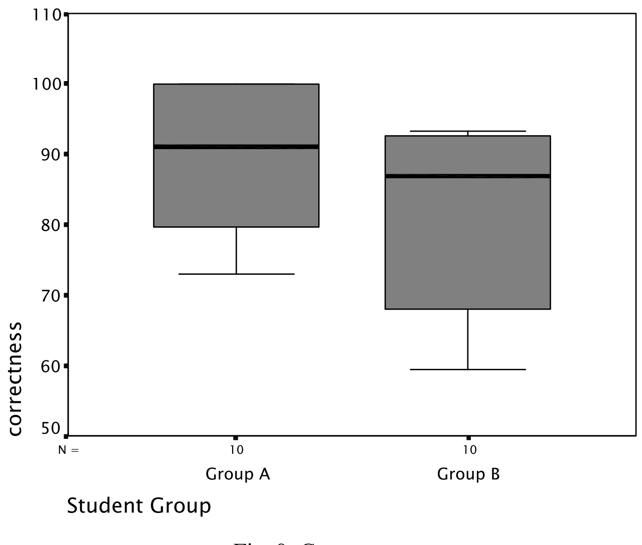

Fig. 8. Correctness.

groups in a much more positive approach than a test for a null hypothesis (Krzanowski, 1993).

In our case the procedure resulted in a model described by the following Fisher's linear DF:

$$z = 15.046 * MEL - 0.84 * INCONSISTENT + 6.881 * COMPLETENESS - 0.209 * CONSISTENCY - 1.454 * CORRECTNESS - 526.4$$

So for a participant student, if we know the values of the five variables, we can compute the value (score) of the classification function and classify him/her to Group A if the score is <0 and to Group B if the score is >0.

The performance of the above DF can be judged by the percentage of correct classifications of the students in the sample. However, this percentage is optimistic since the classified students are the same ones used to build the model. In this regard, the *cross-validation method* is used to decrease the bias. Each student is removed from the data and a classification model is computed from the rest. Then the removed student is classified according to the model. The percentage of the correct classifications by this method is reported too. In our case, all the cases were classified correctly by both methods (with and without cross-validation), i.e. the percentage of the correct classifications is 100%.

Another measure of the model's efficiency is the canonical correlation which is 0.883 (very close to 1), and shows high association between the scores of the DF and the groups. Therefore, the analysis considering the combination of the variables related to maintainability, supports hypothesis $H_2$. Regarding now hypothesis $H_3$, the related variable METHODS DELETED was tested for differences between groups A and B by *t*-test and M–W test. Both tests showed statistically significant difference ($p < 0.01$ and <0.05), so $H_3$ is supported.

### 3.2. $H_1$—ease of understanding

This was captured by the debriefing questionnaire, which was designed to collect both information not covered by the reports, and information to add corroboration to the report data. It includes sections on attitudinal information regarding the background experience, understandability, and maintainability (Table 1). This information was based on the ordinal scale 1-low to 5 or 10-high, depending on the question, which is usually used in similar questionnaires (Briand et al., 2001). We did so, wishing a fine granularity level for some questions. Although the statistical tests do not show any significant difference for the variable understandability between the two groups, we consider some detailed discussion on these subjective data would highlight some aspects concerning the difficulty they faced and what caused it.

First, regarding the *allocated time*, whether it was at best sufficient for them to complete their tasks (question 9), two of the participants, one of each group, felt that they ran out of time. This is an indication that the allocated time was marginal. Second, regarding the participants' *experience* (question 2), we can imply from their low scores that OO concepts are particularly hard to learn and apply. Nevertheless it is clear that they feel more comfortable with programming (question 3). Third, regarding the *understandability*, it is distinguished in four questions aiming to capture different aspects of it. From question 4 "How well you understood what was required of you", we see that both groups felt some difficulty, although their scores are quite high. Regarding the quality of the designs they had (questions 5 and 6), although the difference between the two groups is relatively small (72% vs. 66%, and 66% vs. 57%) shows that Design A had a better quality than Design B in terms of understanding. Concerning what caused the most difficulty to understand the design documents (question 7) the answers are similarly split between the two groups stating both cohesion and coupling. However, considering a related question concerning what caused the most difficulty to perform the modification task (question 11), there is a significant difference between the two groups. Namely, Group A considered it was coupling between classes (8 answers vs. 4 of Group B), while Group B considered it was cohesion (5 vs. 1 of Group A). Notice that Group B answers highlight the cohesion factor, mainly responsible for the 'god class' problem.

However, although the above data cannot be regarded as significant evidence, they do support the direction of our hypotheses and can be considered further support for the results of the quantitative analysis. Nevertheless, we must conclude that $H_1$ is not supported.

### 3.3. $H_2$—ease of maintaining

The quality factor maintainability was captured in this study by the three basic criteria, as described above: *completeness, correctness*, and *consistency*.

Concerning completeness there is a considerable difference between the two groups. Namely, Group B manifests greater number of missed members than Group A. This mainly occurred in those cases inheritance was not applied, as a consequence a number of members that could be reused were not used at all.

Correctness does not indicate any significant difference between the two groups, although Group A manifests higher correctness. It is based on the completed members of the design, mainly on associations between the classes. An example could be the use of a simple association ("uses") where composition with cardinality would be more appropriate.

Consistency is easier to measure here, since it mainly counts contradictions among the design members (duplicated members, redundant attributes, empty methods, inconsistent associations), as described above, and it is an objective measure. Comparing the delivered solutions of the two groups, Group A indicates higher consistency than Group B. These results occur mainly due to duplication.

Analysis of the data collection from question 12, concerning the difficulty to accommodate the new functionality, shows a small difference between the two groups, i.e. Group B found it harder than Group A. However, analysis considering the combination of the variables related to maintainability, supports hypothesis $H_2$.

## 3.4. $H_4$—heuristic compliance

We consider of particular importance the additional evaluation of the delivered solutions by heuristics. This is not because design measures do not provide objective measurements, but considering from another perspective, as that of *subtyping*, we realize that there is a risk the reuse results to prove colorable.

In our case the reuse measurements do not adequately capture the reuse situation. Namely, Group A participants performing on a decentralized design inherited a smaller number of members than Group B, which performing on the 'god-class' version had a larger number of members in one class to inherit. This is not to say that inheriting more design members lead to better design. Group-B, by inheriting from the 'god class' was forced to inherit a few class members not required (redundant) in the new subclass. This suggests a poor design practice that cannot be captured solely by strict measurements. Such design choices, we consider can effectively, as well as objectively, be captured by applying a number of proper design heuristics, such as the heuristic about deleted methods, mentioned previously in Section 3.4 and

Inheritance should only be used for subtyping (Bar-David, 1992).

Both heuristics are of the stand-alone type.

Two kinds of violations were found while applying heuristic assessment: the first one was the 'deleted feature occurrence', which was not violated at all by Group A, while violated by Group B in five cases. The second one was the 'subtyping' heuristic, which was violated by Group A in one case, and by Group B in three cases. However, it is worthwhile to notice that both violations can also be captured objectively by the *substitutability* principle, suggested by Amstrong and Mitchell (1994). The data of the quantitative analysis provide significant evidence toward Group A supporting the direction of our hypotheses. Therefore, $H_3$ is supported.

## 3.5. Analysis summary

The results show that design strategies play important role in the design quality affecting, in different extent, both participants' understandability and maintainability. In this study, Design A complying to the design heuristic under investigation, hence providing a decentralized structure, is slightly easier to understand than Design B, where structure is more concentrated. Nevertheless, it is considerably easier to maintain. Although, decentralization affects the maintainability, due to 'delocalised plans' (Wilde et al., 1993) it is shown here that when compared to centralized design structures, it is more effective in exploiting also inheritance due to cohesive classes, therefore more maintainable.

Analyzing furthermore the data collection from the debriefing questionnaire a number of additional implications could be drawn. Thus comparing the two designs (Design A vs. B), the following issues are stemming:

Higher design quality (question 5: 72% vs. 66%), provides higher understandability (question 6: 66% vs. 57%), and lower difficulty to accommodate the new functionality i.e. maintainability (question 12: 48% vs. 52%). Furthermore examining what factor caused the most difficulty to perform the modification, Group A alleged it was coupling (question 11: 8 vs. 4), apparently due to its design's decentralized structure, while Group B alleged it was cohesion, mostly responsible to its design's concentrated non-cohesive structure (question 11: 5 vs. 1). Notice that questions 7 and 11 included an additional point to note if they considered any other aspect, however, they indicated none.

Comparing the debriefing questionnaire data, to the objective data drawn from participants' delivered designs, further determination of the collected objective data could be provided, as follows:

Design A offered more inheritance reuse convenience (70% uniformly vs. 50% non-uniformly). This indicates encouragement for inheritance for Group A, but discouragement for Group B. Group A also inherited fewer members and in a uniform way, while Group B inherited more members in different ways. A possible explanation is that Design A, with its decentralized functionality and its fewer members, offered more convenience to apply inheritance than Design B, which presumably discouraged the participants, due to greater number of members to inherit and to its centralized functionality. Notice, that if one did not inherit the existent functionality, he should then re-implement it causing a more intense duplication problem in Design B, while if he inherited the whole 'god class', he should introduce a few redundant members.

Comparing the completeness achieved, it is supported with significant evidence that Group A achieved significantly higher completeness (Fig. 6) than Group B. Significant evidence is also provided by the consistency results (Fig. 7). However, no difference is shown from correctness, since it is mainly limited to associations, as described previously. Finally, we expected that mobile phone users could perform better than the others, however there was not any clear indication of it.

## 4. Threats to validity

This section discusses various threats to validity and the way we attempted to alleviate them.

### 4.1. Construct validity

Considering the *construct validity*, i.e. the degree to which various factors accurately measure the concepts they purport to measure, the following possible threats have been identified.

*Understandability* and *maintainability* are difficult concepts to measure, because the former is based on the subjective estimation and experience of the participants, while the latter one, although based on objective measures does not capture all the dimensions of each concept. In a single controlled experiment, however, it is unlikely that all the different aspects of a concept can be captured; hence the researcher must focus on representative as well as realistic conditions. In this study, understandability is examined by four questions in the debriefing questionnaire, while maintainability is examined by three criteria (completeness, correctness, and consistency).

The *compliance* and *violation* of the design heuristic under investigation are also difficult to measure. However, in a previous observational study (Deligiannis et al., 2003), it was shown that different design strategies, as in this case, could be measured by a number of appropriate metrics proposed in the literature. Those metrics may be used to identify the conformity of a design aspect to a design heuristic. In this study, redundant members can be easily measured on a design by a simple metric, while in the case of deleted methods they can be additionally detected by some automated tool, applied on code. However, our approach has been based on work in the literature on OO design and measurement (Bieman and Kang, 1995; Riel, 1996; Whitmire, 1997).

### 4.2. Internal validity

Threats to internal validity are influences that can affect the independent variable with respect to causality. Thus they threat the conclusion about a possible causal relationship between treatment and outcome. The following possible treats are identified.

*Instrumentation*. This is the effect caused by the differences in the artifacts used for the experiment. The threat to this study was the possibility of differences between the two design structures of the system and the modification task causing performance differences. There is no concern for the modification task, since both groups worked on the same task. Considering the different designs applied, as discussed previously, both designs were performing identically. As for their structure, they were also identical, apart from a small part they differed due to the conformance or not to the design heuristic investigated. This is also confirmed from the analysis of the debriefing questionnaire data. Specifically, no significant difference was found on the participants' opinion between the two groups, concerning the quality of the design documents, as well as the difficulty they faced with them. Also, two independent professionals who examined both designs asserted their functionality.

*Selection*. This is the effect of natural variation in human performance. Volunteer students were used, considered more motivated and suited for the task than the whole population. Hence the selected group is not representative for the whole population. Our concern was to select the most capable of the students from the course, offering them a degree bonus for their participation. An additional reason for this kind of selection was that we considered to exclude those not really willing to participate, because they might not perform properly.

### 4.3. External validity

Regarding the *external validity*, i.e. the degree to which the results of the research can be generalized to the population under study and other research settings, the following possible threats have been identified:

This study took place in a university *environment* and not in the work place.

The *size of the system* used in this study is relatively small. Although the project was a real project, its size, 18 and 17 classes may not present an industrial scale piece of software. However, the task was essentially a design extension, i.e. to add a new functionality to a system that occurs in practice in large systems too, and according to OO theory, maintenance should require the modification of small parts of an OO design.

The students used as *participants* may not be representative of OO software professionals. On the other hand, the OO paradigm offers several mechanisms and tradeoffs where decisions on best alternatives are usually fuzzy and mostly based on expert judgement. In other words, cumulative knowledge is likely to play a very important role in the design phase. Thus, it is questionable whether novice designers, performing under such circumstances and with limited time, are the most appropriate subjects. However, Briand et al. (2001) argue that student based experiments can provide useful results for several reasons. First, they can provide confirmatory evidence for results from case or other studies. Second, they can identify interesting hypotheses that are worthy of further investigation in more realistic settings.

In general terms, it would be difficult to try to predict whether, and in what way, these threats may have affected the results. While the external threats limit generalisation of this research they do not limit the results being used as the basis of future studies. Also, this is not to say that the results cannot be useful in an industrial context. Empirical studies as this one allow the investigation of a larger number of hypotheses at a lower cost than field studies, which can then be tested in more realistic industrial settings with a better chance of discovering important and interesting findings.

## 5. Conclusions

This study has investigated the effects of a single design heuristic on system design documents, with respect to understandability and maintainability, two essential components of software quality. The study has compared two designs, Design A and B, which were developed according to design heuristics in general, apart from a small but functionally important part of Design B violating the 'god class' heuristic. Since the difference between them was restricted to a specific part of the design, it is believed that any difference of the results might be due to different design strategies applied.

The results provide sufficient evidence that the specific design heuristic can affect the two quality factors. This is supported by qualitative evidence by two means: (a) in the form of information from the debriefing questionnaire; there, participants expressed the views that Design A was slightly easier to understand and modify, and (b) from heuristic evaluation applied on the delivered solutions. It is also supported by quantitative evidence provided by: (a) simple measurement values concerning design elements as well as the inheritance use; and (b) quality criteria concerning completeness, correctness and consistency of the delivered solutions. All but understandability and correctness show significant differences toward Design A.

Our findings indicate that, concerning maintainability, OO design structures are sensitive to bad or good design practices. It seems that the continuous evolution of a design structure depends on whether certain design heuristics and principles are followed by the developers. That is, a design initially structured under the guidance of heuristics has a greater probability of continuing to evolve in a similar resilient and flexible manner, thus rendering it maintainable and reusable. If design heuristics are violated even once, there is an increased probability of maintenance changes leading to poorer designs, rendering it harder to maintain as design evolves over time, which leads to increasing entropy (Jacobson, 1996). Such a design also minimises opportunities for reuse, an important feature in OO technology.

Further research can investigate: (a) the impact of design heuristics on reusability; (b) the design quality evaluation based on the wealth of existing number of heuristics; (c) the automated detection of design heuristic violations, thus guiding a designers' choices on critical design strategies; (c) the relationships between design heuristics and metrics, since metrics could more precisely and quickly lead the automated process.

## Acknowledgements

We would like to thank the subjects for participating in this experimental study and the reviewers for their helpful comments on an earlier version of this paper.

## Appendix A. PBR technique elements used

*The designer scenario* (correctness and completeness)
*Classes*
Are the classes defined into the design class diagram (DCD) also specified into the requirements?
Are the class names correct?
Are the attributes names correct?
Are the method names correct?

*Attributes*
Is the number of the attributes correct?
Are the types of the attributes correct?

*Methods*
Is there correspondence of the methods between requirements and DCD?

*Associations*
Are the binary associations correct?
Is the cardinality of the associations correct?
Is the arity of the associations correct?

*Constraints*
Are the constraints between requirements and DCD correct?

*Abstract concepts*
Are the abstract concepts properly mapped?

*Questions to answer*
Is there everything you realized in the analysis document that is reflected in the design document?

Are the initial conditions for starting up a system operation clear and correct?
For every message that is defined in the requirements, is there a corresponding message in the DCD?
Is the sequence of messages correct?
*The implementor scenario* (consistency and completeness)
For each class, are the attributes types specified? (consistency)
For each class, are the methods defined? (completeness)
*Questions to answer*
Is there anything that prevents you from implementing the system design?

## References

Abreu, F., Melo, W., 1996. Evaluating the impact of object-oriented design on software quality. In: Proceedings of the 3rd ISMS (Metrics'96), March.

Allen, E., Khoshgoftaar, T., 1999. Measuring coupling and cohesion: an information theory approach. In: Sixth International Symposium on Software Metrics. IEEE Computer Society, Silver Spring, MD.

Amstrong, J., Mitchell, R., 1994. Uses and abuses of inheritance. Softw. Eng. J. (January), 19–26.

Bar-David, T., 1992. Practical consequences of formal definitions of inheritance. J. Object Orient. Program. (July/August), 43–49.

Basili, V., Burgess, A., 1995. Finding and experimental basis for software engineering. IEEE Softw. 12, 92–93.

Bieman, J., Kang, B.-K., 1995. Cohesion and reuse in an object-oriented system. In: ACM Symposium Software Responsibility.

Bieman, J., Xia Zhao, J., 1995. Reuse Through Inheritance: A Quantitative Study of C++ Software.

Booch, G., 1994. Object-Oriented Analysis and Design with Applications. Addison-Wesley, Reading, MA.

Booch, G., 1995. Rules of Thumb. ROAD 2 (4), 2–3.

Briand, L., Bunse, C., Daly, J., 2001. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. IEEE Trans. Softw. Eng. 27 (6), 513–530.

Chaumun, M.A., Kabaili, H., Keller, R.K., Lustman, F., 2002. A change impact model for changeability assessment in object-oriented software systems. Sci. Comput. Program. 45, 15–174.

Chidamber, S., Kemerer, C., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20 (6), 476–493.

Coad, P., 1991. OOD criteria, part 1–3. J. Object Orient. Programm. (Jun-Sep), 67–70.

Coad, P., Yourdon, E., 1991. Object-Oriented Analysis. Prentice-Hall, Englewood Cliffs, NJ.

Coplien, J., 1992. Advanced C++. Addison-Wesley, Reading, MA.

Deligiannis, I., Shepperd, M., Webster, S., Roumeliotis, M., 2002. A review of experimental investigations into object-oriented technology. Empiric. Softw. Eng. J. 7 (3), 193–231.

Deligiannis, I., Shepperd, M., Stamelos, I., Roumeriotis, M., 2003. An empirical investigation of object-oriented design heuristics for maintainability. J. Syst. Softw. 65 (2), 127–139.

Fenton, N., Pfleeger, S.L., 1997. Software Metrics, A rigorous & Practical Approach, second ed. International Thompson Computer Press.

Firesmith, D., 1995. Inheritance guidelines. JOOP (May), 67–72.

Glass, R., 2002. In search of meaning (a tale of two words). IEEE Softw. 19 (4), 136, 134–135.

Goodley, S., 1999. Java on course to dominate by 2002, Available from <http://www.vnunet.com/News/87054>.

Hatton, L., 1998. Does OO sync with how we think? IEEE Softw. (May/June), 46–54.

Hillegersberg, J., Kuman, K., Kuman, K.,Welke, R., 1995. An empirical analysis of the performance and strategies of programmers new to object-oriented techniques. In: Psychology of Programming Interest Group: 7th Workshop.

Jacobson, I., 1996. Object-Oriented Software Engineering: A use case driven approach. Addison-Wesley, Englewood Cliffs, NJ.

Jacobson, I., Christerson, M., 1995. A confused world of OOA and OOD. JOOP (Sep.), 15–20.

Jones, G., 1994. Gaps in the object-oriented paradigm. IEEE Comput. (Jun.), 90–91.

Kirsopp, C., Shepperd, M., 2001. Using heuristics to assess object-oriented design quality. In: 5th International Conference on Empirical Assessment & Evaluation in Software Engineering, Keele University, Staffordshire, UK.

Krzanowski, W.J., 1993. Principles of Multivariate Analysis. Oxford University Press.

Laitenberger, O., Atkinson, C., Schlich, M., Emam, K., 2000. An experimental comparison of reading techniques for defect detection in UML design documents. J. Syst. Softw. 53, 183–204.

Lorenz, M., Kidd, J., 1994. Object-Oriented Software Metrics. Prentice-Hall, Englewood Cliffs, NJ.

Meyer, B., 1997. Object-Oriented Software Construction. Prentice-Hall PTR.

Riel, A., 1994. Introducing to object-oriented design heuristics. In: OOPSLA'94, Portland, OR, USA.

Riel, A., 1996. Object-Oriented Design Heuristics.

Rumbaugh, M., Blaha, M., Premerhani, W., Eddy, F., Lorensen, W., 1991. Object-Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs, NJ.

Sharble, R., Cohen, S., 1993. The object-oriented brewery: a comparison of two object-oriented development methods. ACM Sigsoft.—Softw. Eng. Notes 18 (2), 60–73.

Whitmire, S., 1997. Object Oriented Design Measurement. John Wiley & Sons, New York.

Wilde, N., Mathews, P., Ross, H., 1993. Maintaining object-oriented software. IEEE Softw. (January), 75–80.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., Wesslen, A., 2000. Experimentation in Software Engineering—An introduction. Kluwer Academic Publishers, Dordrecht.