ELSEVIER

The Journal of Systems and Software 65 (2003) 127–139

The Journal of Systems and Software

www.elsevier.com/locate/jss

An empirical investigation of an object-oriented design heuristic for maintainability

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

[a] Department of Informatics, Technological Education Institute of Thessaloniki, 54101 Thessaloniki, Greece
[b] Design, Engineering and Computing, Bournemouth University, BH1 3LT Bournemouth, UK
[c] Department of Applied Informatics, University of Macedonia, Thessaloniki, 54006 Thessaloniki, Greece
[d] Department of Informatics, Aristotle University of Thessaloniki, 54006 Thessaloniki, Greece

Received 21 July 2001; received in revised form 31 August 2001; accepted 14 November 2001

Abstract

This empirical study has two goals. First, to investigate the impact of a design heuristic on the maintainability of object-oriented designs, namely the 'god class' problem. In other words, we wish to better understand to what extent a specific design heuristic contributes to the quality of designs developed. The second goal is to investigate the relationship between that OO design heuristic and metrics. Namely, are we able to capture a specific design heuristic by applying a suitable subset of design metrics? The results of this study show that: (a) the investigated design heuristic significantly affects the performance of the participants; (b) it also 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.

Keywords: Object-orientation; Empirical study; Design; Heuristics; Metrics

1. Introduction

Software systems, after initial release, are subject to continuous evolution and modification in order to respond to changes in the real world. Thus, we want them to be easy to understand, enhance or correct; if they are, we say that our software is maintainable (Fenton and Pfleeger, 1997). To achieve those goals it is essential that software be based on robust design. A good design allows us to easily *plug-in* new functionality in terms of new classes and new methods without a need to re-implement the results of the previous iteration cycles. More specifically, it is crucial that we identify sound classes (Booch, 1995), as well as their dependencies, which reveal the implication of change—a change in one class will potentially change dependent classes. Clearly, we want dependencies that make the system easy to maintain and change (Jacobson and Christerson, 1995). The maintenance phase is considered the most costly part of the system lifecycle (Hatton, 1998; Meyer, 1997). In order to apply it successfully, two things are required: the ability to make enhancements and changes easily—*maintainability*—and in-depth understanding of the software's structure and behaviour—*understandability* (Wilde et al., 1993).

A wide variety of heuristics [1] have been proposed in the literature in order to improve the structural quality of OO designs (Booch, 1995; Jacobson and Christerson, 1995; Riel, 1996; Firesmith, 1995; Lorenz and Kidd, 1994). Such heuristics are aimed at providing ways and techniques for developers to make proper decisions. Design heuristics strive to enhance design quality, thus affecting the highly desirable quality factor in OO software engineering of maintainability. Maintainability is a high-level quality factor that is indirectly evaluated in this study by the following means: a debriefing questionnaire, the measured performance effort, i.e. the time

* Corresponding author. Tel./fax: +30-310-791295.
E-mail address: igndel@it.teithe.gr (I. Deligiannis).

[1] The computer science field has traditionally defined heuristics as rules of thumb. However, the dictionary does not support this definition (instead, it says heuristics are trial and error approaches). In this paper we use the term with the computer science meaning.

0164-1212/03/$ - see front matter © 2002 Elsevier Science Inc. All rights reserved.
doi:10.1016/S0164-1212(02)00054-7

spent to perform the modification task, and the delivered solutions.

Both before and during the maintenance phase, software measurement can be extremely valuable. During the maintenance process, measurements could be a guide, so that we can evaluate the impact of a change, or assess the relative merits of several proposed changes or approaches (Fenton and Pfleeger, 1997). One of the most influencing factors of software systems quality, in which metrics can play an important role, is the structure of software design (Abreu and Melo, 1996). Detecting potential problematic modules and dependencies at this early stage of system lifecycle, before the underlying design decisions are ''ossified'' into code, is one of the main objectives of design metrics (Abreu and Carapuca, 1994). The OO literature abounds with proposed metrics aimed at providing ways of assessing the quality of various design aspects. Such an assessment of design quality is objective, and the measurement can be automated. But how do we know which metrics actually capture important design aspects, as for example provided by heuristics? Despite numerous theories and claims on what constitutes good OO design and best solutions on specific situations, only empirical studies of actual systems structure and quality can provide tangible answers. So far (to the best of our knowledge), one empirical study directly related to object-oriented design heuristics has been performed (Kirsopp et al., 1999). The lack of empirical studies into this area was our major motivation for this study.

The work presented in this paper is an observational study, where the impact of a single design heuristic on quality of object-oriented designs, as well as its effect on maintainability, is empirically investigated. Furthermore, by applying a number of metrics proposed in the literature that are considered most related to structural design aspects, we investigate whether it is feasible to determine where that design heuristic is violated. In doing so, it will make it possible to identify a suitable subset of design metrics, which could detect cases where the specific design heuristic is breached at an early stage of the design phase.

The rest of the paper is structured as follows. In the next section a description of OO heuristics is given as well as their relationships to measurement. The following section describes the design and the performance of the observational study. An analysis of the results is then presented in Section 4. In Section 5 threats to validity are discussed. In the final section, a number of implications are drawn from this work and future work is discussed.

2. OO design heuristics

Since the topic of OO design heuristics is broad and there is lack of precise definition, we consider it useful to quote the most interesting statements found in the literature. Booch states (Booch, 1995): ''As any discipline matures, experience accumulates, drawn from both successful and unsuccessful ventures. Typically, this experience gets passed on through books, articles, classes and mentoring. However, as more and more details accumulate, or as these lessons become more complex, it becomes difficult for new folks to absorb this wisdom as they approach the discipline for the first time. Indeed, 'old hands' in any discipline, be in farming or painting or even programming, will collapse many of their lessons learned into simple rules of thumb that serve to guide their actions. A rule of thumb is intentionally imprecise, but it at least puts you close to what is exactly right''. Other authors describe them as: guidelines to help developers make proper decisions; identified and encapsulated experience; imprecise and informal guide to good and bad practices; provision of knowledge and judgement. However, it is widely admitted that they 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 in the literature (Booch, 1995; Firesmith, 1995; Jacobson and Christerson, 1995; Lorenz and Kidd, 1994; Riel, 1996) related to a multitude of software aspects. As mentioned above, they are aimed at enhancing software quality. However, when violating them there is a risk of leading to 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.

The design heuristic investigated in this study is proposed by Riel (1996) which we consider works 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'' (Riel, 1996).

Our motivation for the selection of the above 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. Therefore, the major objective of this study is to investigate to what extent such classes/objects affect the maintainability of design.

In order to assess the benefits provided by the investigated heuristic, as well as its impact on the quality factor—maintainability—in which this study is most interested in, quantitative analysis is also required. As mentioned above, metrics are the primary means of assessing specific design aspects; they are easy to automate; and they are specific and descriptive (Lorenz and Kidd, 1994). Whitmire argues that, objective criteria depend upon data gathered through measurement (Whitmire, 1997). He 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).

Into this measurement framework, this study is most closely related to a technical point of view, and to objects of processes and products. From processes the internal attributes are measured: time and number and duration of incidents of a specific type of activity. From the products the external attributes are measured: maintainability and understandability, as well as the internal attributes: size (number of new and modified classes, methods, attributes, and associations), coupling and cohesion.

Subsequently, it is expected that applying mainly product metrics as strongly related to design aspects, a suitable set of design metrics capturing a specific design heuristic could be determined. This is, however, a secondary objective of our study.

3. Design of the empirical study

3.1. The task

The application used for the study was the tariff selector system (TSS). This is a small but realistic application, implementing a mobile phone TSS to aid prospective buyers of telephones, in determining which tariff best matches their needs. We produced two identically performing versions of the system, called Design A and B. They were designed using the OMT method (Rumbaugh et al., 1991), an analysis/design method, and the Unified Modelling Language (UML) notation (Booch et al., 1999). They were implemented using the Java 1.2 programming language and have a GUI built using the standard AWT components. Although participants only worked with design documents, it was considered that the implementation of the two versions of the system was necessary for three reasons: (a) to help participants to thoroughly understand its functionality through executing the program, prior to any modification; (b) to make feasible measurement evaluation of the systems (the initial and the produced versions), apart from design metrics, even from metrics applied on code; and (c) to ensure that the two designs were indeed functionally equivalent.

Design A consists of 18 classes while Design B of 16. Design A is considered to be the 'heuristic compliant' one, due to its decentralized structure. It was designed and implemented according to the above-mentioned design heuristic. Design B, considered as the 'heuristic non-compliant' one, was designed and implemented violating this heuristic. The basic concept was to build a class playing a central role into the system, as described above. The two versions differed in that, in Design B one class captured the central functionality of the system, while 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); (ii) ''Avoid having a large, central object which controls most of the functionality'' (Riel, 1994). Extra care was taken for both systems, except the 'god' class in Design B, to be consistent with some other properties for well-designed classes, namely: coupling, cohesion, sufficiency, completeness, and primitiveness (Booch, 1994).

The participants first had a session running the application, intended to give them a thorough understanding of its functionality. Then they were given a set of documents (Table 1) describing the system. The required modification task (Appendix A) was to add new functionality mostly affecting the part of the system under investigation. Thus, a new modified class diagram was expected from each participant. The whole maintenance process was videotaped, intended to record every activity performed by each participant.

3.2. Study participants

Four participants were used to perform maintenance tasks on two system designs, A and B. In the rest of the paper we call them as A1, A2, B1 and B2 for

Table 1
Documents provided to the participants

(a) Specifications
(b) Use case diagram
(c) Event flow diagram
(d) Class diagram
(e) Sequence diagram
(f) Modification task

convenience, according to the system they worked with. Two were given the version compliant to design heuristic (Design A). The first subject 'A1' was a research fellow having a Ph.D. degree in OO Technology but with little practical experience on OO design documents. The other subject 'A2' was a Senior Lecturer, also having little practical experience on OO design documents. The other two were given the non-compliant version (Design B), which violates the design heuristic under investigation. Subject 'B1' was a Ph.D. candidate having sufficient practical experience. The other one 'B2' was a research fellow, teaching OO methodologies. Nevertheless, experience level is a subjective factor. It is also worthwhile to note that 'A2' and 'B1' are mobile phone users, thus more familiar with the whole functionality of the 'TSS', used for this study.

3.3. The method

The method used here is the observational study. Kirwan and Ainsworth (1992) state that, the objective of observational techniques is to obtain data by directly observing the activity or behaviour under study. They may be used to record the full overt sequence of actions. The experimenter may subsequently choose to extract whatever information is of interest, usually from an audio–visual recording. Thus, observation methods can be particularly useful for recording physical task sequences. Some advantages of such methods are the following:

- Detailed physical task performance data can be recorded.
- They are ideal for pilot studies, as they can reveal potential behaviour patterns and influences, which may not have been predicted, and can be subsequently analysed in more detail.
- They can be used to identify and develop explanations of individual differences in task performance.
- They provide objective information, which can be compared with information collected by another observer, or by another method.

These characteristics of the method, in particular the convenience to observe and document in detail by videotaping every activity of each participant, from the whole maintenance session, were the major motivators for using it. More specifically, by documenting the duration (action time) and visiting frequency on each available design document (Fig. 1a and b), it is hoped to better understand the participants' behaviour, namely to identify possible behaviour patterns, in addition to when and where they face difficulties working on the different designs.

3.4. Data collection

Four different elements made up the data collected for this study. The description and assessment of each one is as following:

1. *Performance effort:* This mainly refers to time spent and how it was spent, on each design document, and refers to two kinds of data:

(i) *Time*, distinguished in *total time*, *activity time*, and *action time*. Action time refers to the time spent in a single document visit (an *action*) and is shown in Fig. 1a and b. Activity time is the cumulative time spent in visiting a specific document and is shown in the relative columns of Table 2 and in Fig. 2. Total time is the time spent by each participant during the study and is reported in Table 2.
(ii) *Design document analysis*, which concerns the manner time was spent. It captures the number of visits (actions) the participants performed to each available design document, their order, their frequency, and their pattern. It is distinguished in *action frequency*, *activity sequence*, *and activity interrelation* (Table 2, Fig. 1a and b).

All data was extracted from the video recording and is objective in nature.

2. *Delivered class diagrams:* These refer to four reports with the delivered solutions, each produced by one participant, as required by the modification task. They were evaluated for their *completeness*, *correctness*, and *consistency* (Table 3).

3. *Metrics and measurement values:* In order to perform a thorough assessment of all six designs (two original A and B, and four produced by the participants), a number of OO metrics were needed along with a tool to extract measurement values according to these metrics. For this purpose, three well-known suites of metrics were used, proposed in the literature (Abreu and Melo, 1996; Chidamber and Kemerer, 1994; Lorenz and Kidd, 1994). Measurements were collected using a measurement tool named MOT, developed at Bournemouth University. Thus, six sets of metrics values were extracted, one from each design. These values provide a means to evaluating the quality of the total six designs, since they capture specific quality properties, as well as to make comparisons between them in order to draw some conclusions.

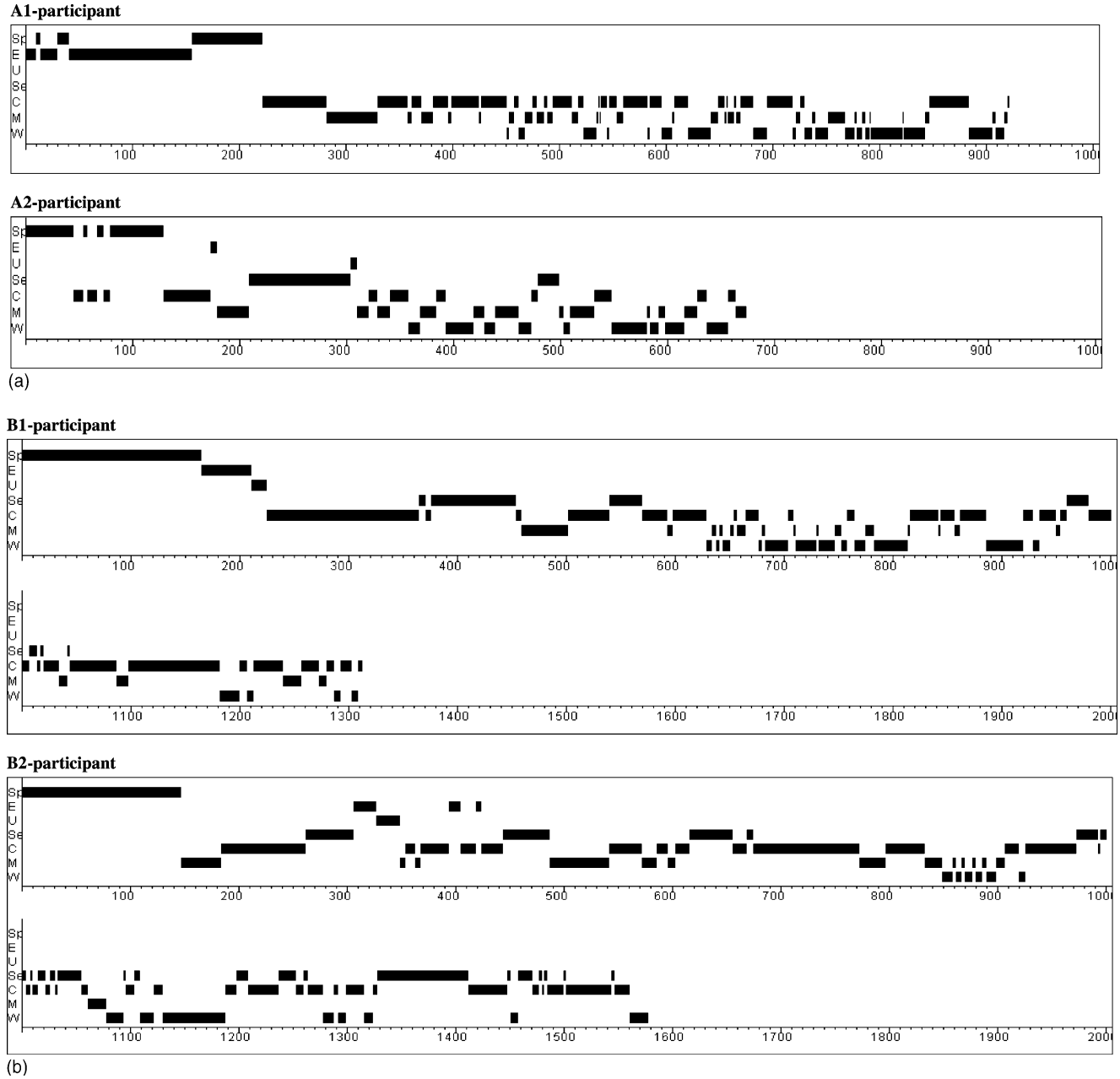

Fig. 1. (a) Detailed document visiting diagram of group A participants (action duration, sequence and interrelation of activities); times in seconds. Activity list—Sp: specifications, E: event flow diagram, U: use case diagram, Se: sequene diagram, C: class diagram, M: modification task document, W: write on class diagram. (b) Detailed document visiting diagram of group B participants (action duration, sequence and interrelation of activities); times in seconds.

Comparisons were performed in three directions: (i) between the two original designs (A against B); (ii) between each original design and the two new ones produced from that (e.g. A against A1 and A2; B against B1 and B2), and (iii) between the four produced designs (e.g. A1 and A2 against B1 and B2). Also, so as to further evaluate the designs, the six designs were implemented as code.

From the three suites of metrics applied, it was decided to present (Tables 4 and 5) only those metrics whose values differ between the two original designs (A and B), while the others were discarded as not indicating any difference. This was done because it was considered that their differences are due to their sensitivity to the design strategies that were used in this study. Note, that some of the metrics are applicable to code and not on raw designs (Lack of cohesion in methods (LCOM) and Response set for a class (RFC) in Table 4).

Three kinds of assessment were provided by the metrics values. The first two refer to the original

Table 2
Frequency and performance times per activity (document)

Document	Version compliant to design heuristics (group A)				Version non-compliant to design heuristics (group B)			
	Participant A1		Participant A2		Participant B1		Participant B2	
	Freq.	Activity time	Freq.	Activity time	Freq.	Activity time	Freq.	Activity time
Specifications	3	01:21	4	1:45	1	02:45	1	02:27
Use case diagram	–	–	1	0:06	1	00:14	1	00:22
Event flow diagram	3	02:21	1	0:06	1	00:46	4	00:37
Class diagram	25	05:44	11	2:20	30	10:04	34	10:45
Modification task	28	02:48	12	2:38	19	02:23	15	03:21
Sequence diagram	–	–	2	1:55	7	02:26	22	05:47
Writing on Class D	18	03:08	9	2:24	16	03:15	15	02:59
Totals	77	15:22	40	11:14	75	21:53	92	26:18

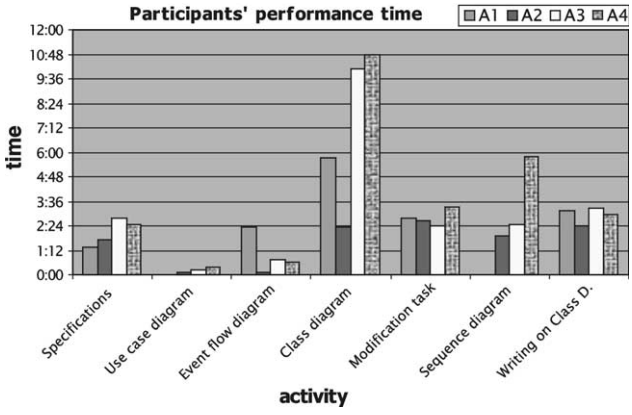

Fig. 2. Total performance time per participant.

versions (Design A and B) and the third refers to the produced versions:

(i) *Thresholds:* Since Design A was constructed under the guidance of heuristics, its metrics values were considered as thresholds so that necessary comparisons with the other versions could be performed.

(ii) *Heuristic violation:* To justify and specify in which part of Design B the heuristic violations took place, it was first needed to specify a set of metrics. Such metrics were chosen from the three metrics suites discussed above. Only those metrics with values considerably different from their thresholds have been considered (Table 5).

(iii) *Maintainability:* This quality characteristic can be assessed by comparing each measurement to its corresponding threshold, providing two pieces of information: (a) evaluation of any improvement or deterioration of maintainability; (b) justification of additional effort for the modification task.

4. *Debriefing questionnaire:* This captures the participants' personal opinions regarding architectural aspects of the systems, as well as the modification task (Appendix B and Table 6) and is subjective in nature. The debriefing questionnaire 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 maintain-

Table 3
Evaluation data from the modified class diagrams according to Laitenberger's checklists

	Criteria	How to detect	A1	A2	B1	B2
1.	Cons.	Is each name unique?	Y	Y	Y	Y
2.	Cons.	Are all names consistent?	Y	Y	Y	Y
3.	Corr.	Are all names correct?	Y	Y	Y	Y
4.	Compl.	Have all classes been adequately defined and described?	Y	Y	N One attribute and one method have not been defined at all	Y Attributes and methods that should normally belong to a new class have been defined into an existing class
5.	Compl.	Does each class attribute have an associated type?	Y	Y	Y	Y
6.	Cons.	Are all data types primitives?	Y	Y	Y	Y
7.	Corr.	Is the cardinality and arity of each association correct?	N	Missed	Y	Y
8.	Cons.	Are the design class diagrams consistent?	Y	Y	Y	Y

Cons.—consistency, corr.—correctness, compl.—completeness.

Table 4
Metrics distinguishing heuristic violation

Name	Description	Reference
The next three metrics are all described in Abreu et al. (1995). They are used as Intermediate calculations for producing Abrue's main 6 metrics.		
Number of attributes defined	The number of attributes defined within a class. This does not include inherited attributes.	Abreu et al. (1995)
Number of methods available	The number of methods that can be called on a class. This includes inherited methods.	Abreu et al. (1995)
Number of visible methods	A count of all public methods in a class. Public methods are the methods that can be executed in response to a message received by an object of that class.	Abreu et al. (1995)
CBO (coupling between object classes)	A class is considered coupled to another, if methods of one class use methods or attributes of the other, or vice versa. CBO for class is then defined as the number of other classes to which it is coupled. This includes inheritance-based coupling (coupling between classes related via inheritance).	Chidamber and Kemerer (1994)
COF	A ratio of actual coupling between system classes and maximum potential coupling.	Abreu and Melo (1996)
LCOM	The number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do.	Chidamber and Kemerer (1994)
RFC	Response set of class is considered the set of methods M of the class, and the set of methods directly or indirectly invoked by methods in M. RFC is the number of methods in the response set of the class.	Chidamber and Kemerer (1994)
WMC (weighted methods per class)	WMC is a 'size' metric which weights a method count with its complexity. Complexity is deliberately not defined more specifically here in order to allow for the most general application of this metric. Therefore, some traditional static complexity metric may be appropriate.	Chidamber and Kemerer (1994)
Class Coupling	Coupling between classes relates to the interrelationships that bind the two together. Is it measured by: • The number of other classes collaborated with, and • The amount of collaboration with other classes.	Lorenz and Kidd (1994)
NOIM (number of instance methods)	Number of non-static methods defined in a class. It counts all the public, protected, and private methods defined for a class' instances.	Lorenz and Kidd (1994)
NOPIM (number of public instance methods)	Same as NOIM but only counts public methods.	Lorenz and Kidd (1994)

Table 5
Metrics sets extracted from the 'control class' of each version (original and delivered by the participants)

Metric	Version					
	Design A			Design B		
	Original	A1	A2	Original	B1	B2
Number of attributes defined	5	5	5	8	8	8
Number of methods available	11	11	11	17	17	18
Number of visible methods (Mv)	9	9	9	15	15	16
CBO	14	14	14	16	16	18
COF	0.1287	0.1199	0.117	0.1048	0.1042	0.0956
LCOM	0.0	0.0	0.0	54.0	54.0	71.0
RFC	21	21	21	53	53	60
WMC	11	11	11	17	17	18
Class Coupling	48	48	48	63	63	69
NOIM	11	11	11	17	17	18
NOPIM	9	9	9	15	15	16

ability (Table 6). 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.

4. Results

The results presented in this section are grouped into *four* sub-sections. Section 4.1 is an analysis and discussion of how the participants performed the task. Section

Table 6
Data from the questionnaire, modified class diagrams and performance effort

Pa/nt	Version	Question														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A1	H-Compl.	2	2	2	3	7	Syntax/naming conventions	5	Declarations & dataflow	Coupling	41–60	5	3	2	–	No
A2	H-Compl.	2	3	3	4	6	The notation used	6	–	Notation	41–60	6	3	3	Read about the notation	Yes
B1	H-non-Compl.	3	2	3	4	6	Coupling	8	The distribution of data across objects	Coupling	0–20	3	8	2	Re-design	Yes
B2	H-non-Compl.	3	4	4	3	4	Method implementations	4	Use of central 'God' class	Coupling	21–40	5	6	2	Request longer to familiarize with the structure	No

		Metrics for maintainability			Change	Effort
		New classes added	Original classes affected	New attributes	New visible methods	New associations	Type	Time in min
A1	H-Compl.	2	1	5	6	3	Decentralized	15:22
A2	H-Compl.	2	1	5	3	3	Decentralized	11:14
B1	H-non-Compl.	1	1	4	4	2	Centralized	21:53
B2	H-nonCompl.	1	3	5	6	2	Centralized	26:18

H-Compl—Heuristic compliant, H-non-Compl.—Heuristic non-Compliant; Questions 6, 9, and 14: A summary of participants' answers is only cited on the table.

4.2 covers an analysis of their delivered solutions to the task. In Section 4.3 an analysis is performed to highlight any relationship between the investigated design heuristic and metrics. Finally, data concerning the participants' attitudinal opinion as well as their background experience is presented.

4.1. Analysis of performance effort

For convenience, the participants were divided into two groups A and B performing on the corresponding design. In Fig. 1a and b, the data clearly shows how the participants behave, trying first to understand the functionality of the whole system and then, to accommodate the required new functionality. Here, the duration, the sequence, and the frequency of visiting each document are of most importance.

Concerning the *total time* spent, performing the task, there is a considerable difference between the two groups (Table 2 and Fig. 2). Namely, group B spent considerably more time in total than group A. Examining the *activity time* spent focusing on documents, the 'class diagram' and 'sequence diagram' show the greatest difference, about 100% for the heuristic non-compliant versions. These times were significantly different between the two groups. Thus, it is believed that this is due to extra difficulty the group B faced performing on the heuristic non-compliant version.

The *frequency of document visiting* (*Freq*. columns in Table 2) seems of particular interest as it does show which documents are most visited (studied), as well as the way they are visited (Fig. 1a and b). The 'class diagram', 'modification task', 'sequence diagram' and the activity 'writing on class diagram' show the highest scores. Although the numbers do not differ significantly, the 'sequence diagram' is the most studied by group B. We might infer from this that group B, due to facing extra difficulty, attempted to gain extra information from the 'sequence diagram' where the interaction between objects was precisely recorded. This can also be confirmed by their answers on question 9 of the questionnaire, where they clearly state that coupling between objects caused the most difficulty to them.

From the *detailed document visiting* diagrams (Fig. 1a and b), two additional activities could be distinguished, namely *ordered document visiting* and *alternate document visiting*. Concerning ordered document visiting, it refers to the sequence of the alternate visits on the available design documents, the participants performed for a relatively long time (1–2 min per each document), as shown from the leftmost (initial) parts of the diagrams (e.g. see B1 participant's activity from 0 to 600). Striving to form a complete mental picture of the system's functionality, in order to accommodate the new functionality, they picked up related concepts illustrated on the various available design documents. From the sequence each participant

visited the design documents, it was assessed that some knowledge about possible interrelationship between the design documents, due to the pertinent information they provide, was gained.

Regarding the second activity (alternate document visiting), which refers to the frequency a few specific pairs of design documents were visited alternately, as shown from the rightmost (final) parts of the diagrams (e.g. see B1 participant's activity from 600 to 1000), the participants visited the documents alternately for a relatively short time (from some seconds to half a minute per each document).

From the above observations, it was considered that a number of sub-phases might be distinguished during the maintenance session. The possibility to perform such type of analysis demonstrates the advantages of the observational method. The identified maintenance phases are the following:

(a) *Knowledge of the system:* Trying to gain a thorough understanding of the system, they mostly visited the documents of specifications, event flow diagram and class diagram. However, they approached them differently, as it can be seen in Fig. 1a and 1b. Namely, 'A1' mostly studied the event flow diagram interchangeably with the specifications. 'A2' mostly studied the class diagram interchangeably with the specifications. 'B1' mostly studied the class diagram, and 'B2' studied all the diagrams. This sub-phase was considered to be finished at the first visit to the modification document (e.g. time 460 in B1-participant's diagram).
(b) *Exploratory:* The exact location of placing the new functionality could be the main characteristic in this case. The participants mostly focused on the class diagram and the modification task documents and less on the sequence diagram. This sub-phase was considered to be finished at the first 'write' action on the class diagram (e.g. time 630 in B1-participant's diagram).
(c) *Accommodating:* Having consolidated their knowledge of the system and identified where to accommodate the new functionality, the participants mostly focused on the class diagram and the modification task. The dominant characteristic here is the long write actions that took place (e.g. approximately up to time 1250 in B1-participant's diagram).
(d) *Checking:* The final sub-phase was to confirm that the modification was completed. However, a few sparse and short write actions also took place (e.g. from time 1250 to the end in B1-participant's diagram).

4.2. Analysis of delivered solutions

Examining each participant's delivered class diagram, it was considered they should be approached from two different points of view, namely the way the class diagrams evolved and the quality they provided.

Concerning diagram evolution, it was noted that the class diagrams evolved and developed in a similar way to the base designs. Namely, each one extended the functionality in the way the original design was built. Those performing on the heuristic compliant version produced a decentralized solution consisting of two classes, while the other two produced a centralized solution in one class (Table 6). A likely explanation for this is that the participants were affected by the structure of the design they were given, so that they were forced to continue in that style. Our subjective opinion stemming from the implementation we afterwards performed was that the heuristic non-compliant versions were also harder to implement than the heuristic compliant ones. We can therefore infer that, while the heuristic compliant version evolves smoothly, the heuristic non-compliant one manifests a more rapid deterioration.

According to Laitenberger et al. (2000) there are three basic criteria for evaluating the quality of UML-design diagrams: *Correctness*, *completeness* and *consistency*. Each one of these criteria can be operationalized as follows:

1. *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.
2. *Completeness:* A diagram or a set of diagrams is complete if no required elements are missing.
3. *Consistency:* A diagram or a set of diagrams is consistent if there are no contradictions among its elements.

Based on these criteria, Laitenberger developed a checklist providing a means for systematic design evaluation. From their checklist, a number of elements was used that best conform to the delivered solutions, in our case. The results of this analysis are presented in Table 3. Considering the *completeness* of the delivered solutions, there is a difference between the two versions towards the heuristic compliant version. Namely, 'B1' missed an attribute and a method, while 'B2' put a number of attributes and methods into an existing class instead of creating a new, resulting in a non-intuitive implementation. Considering the *correctness*, we observe that 'A2' missed the cardinality of the new relationships.

4.3. Analysis of metrics

To make meaningful metrics-based comparisons, both similarities and dissimilarities of the product being compared must be known. In this study, two classes were actually distinguished and compared, the *controller classes*, as described above, because of two reasons: (a)

they capture the central functionality of the system, and (b) their measurement values differ considerably due to the design strategies followed (Table 5).

Observing each version separately, it was noted that the vast majority of the measurement values of the designs generated from Design A do not show any change. Only the coupling factor (COF) metric shows some deviation for the produced designs from their original. As mentioned above, the original values of Design A were also considered as thresholds. Since the rest of the values do not differ it was concluded that the design quality has not been affected. Examining Design B, it can be seen that its original version values differ significantly from its counterpart Design A. This is due to the violations taken place, as discussed above. The modified design values also differ with each other. This indicates that there is a lack of *homogeneity* between their solutions. Although participant 'B1' design shows similar values with its original, 'B2' design differ significantly. Such a deviation suggests a clear deterioration of the design quality. At this point a special note is made that if such deterioration occurs when every design change takes place, the resulting design will become increasingly unmanageable after a sequence of modifications.

Another interesting finding is that some metrics do not always adequately capture the context they are intended for. In our case, the LCOM metric related to cohesion of a class, shows a clear difference between the two classes. Nevertheless, it does give high values to another trivial class dealing with 'get' and 'set' functionality. It is our belief that it might be useful in combination with other metrics. This is in agreement with Berard's argument (Berard, 1996), that a single software engineering metric in isolation is seldom useful. Instead, the most useful set of metrics for some particular product may be gathered from multiple metrics, known ahead of time. Our findings therefore, are in agreement to this statement. They also support our investigation question that design heuristics might be captured by a predefined set of metrics. However, those metrics should provide well-established and empirically evaluated thresholds.

4.4. Analysis of debriefing questionnaire

The questionnaire (Appendix B) provided the opportunity for the participants to express their subjective views about the structure and quality of the system and the ease of performing the required modification. The results are illustrated in Table 6. The quality factor *understandability* (questions 4–8), was captured and evaluated by the following points:

(a) *Understanding what was required (question 4)*. This is considered as a precondition to perform well and is related to their experience in practice. However, from their answers it was inferred that familiarity with the application domain plays an important role. The participants using mobile phones (question 15), understood slightly better what was required than the others.
(b) *Estimating, in terms of understandability, the quality of the design documents (question 5)*. This was intended to capture to what extent the quality of the design documents, of each original design, affected their understandability. There is a difference between the groups in that the A1 and A2 found it more understandable.
(c) *Understanding the overall design documents (question 7)*. There is no clear indication from this point. Group A's opinions are in agreement, while group B's considerably differ indicating some confusion. It is worthwhile to mention that B1, although he found it very understandable, did not perform adequately since the modification was incomplete (refer to the completeness row in Table 3).
(d) *What was least understandable about the design documents (question 8)*. This was intended to specify what kind of system information caused most difficulty. From both groups, group B gives a quite clear indication. Participant 'B1' considered that there was not a good distribution of data across objects, which identifies a cohesion problem, while the other, 'B2' precisely identified the problem.

The other important factor *maintainability (questions 9–12)*, was captured by the following points:

(e) *What caused the most difficulty to perform the modification task (question 9)*. From their responses there was agreement between them in that coupling caused the most difficulty. Coupling is a factor closely related to the design structure.
(f) To what extent *the quality of the design documents* affected their ability to make modifications to the system *(question 11)*. From their responses we see that group B was affected slightly more.
(g) *The overall difficulty of the task (question 12)*. Comparing the two groups it is apparent that there is a considerable difference. Namely, group A found the task quite easy, while group B found it quite difficult.

From the last four points (point *d* to *g*), it can be inferred that design quality is closely related to the difficulty of modification. Examining what factor causes most difficulty, both in understanding and performing modifications to the system, most participants agree that it is coupling between classes (question 9). Coupling relations increase complexity, reduce encapsulation and potential reuse, and limit understandability and maintainability (Abreu et al., 1995).

5. Threats to validity

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

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: (a) Understandability and maintainability are difficult concepts to measure, because they are based on the subjective estimation and experience of the participants. (b) The design heuristic under investigation, and the violation in Design B, are also difficult to measure. However, it is believed that the metrics selected capture a wide range of design aspects and the values between Design A and B show a considerable difference, indicating different design strategies. These values may be used to identify the conformity of a design aspect to a design heuristic. (c) Concerning the metric of 'public methods', ('Number of visible methods' in Table 5), which in the heuristic compliant version (Design A) shows 9 methods, there is a disagreement between two authors each proposing a different threshold for this metric. Namely, Coad (1991) is suggesting, ''Each class typically has no more than seven public services'', while Lorenz and Kidd (1994) are suggesting, ''The average number of public methods (services) for a class should be lower than 12 methods''. Although our metric value is consistent with the second threshold, slightly exceeding the first one, this case stresses the necessity for a more systematic empirical investigation of design heuristics.

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: (a) this study took place in a university environment and not in the work place. (b) the size of the system used in this study is relatively small. Although the project was a real project, its size, 16 and 18 classes is not 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 small systems too, and according to OO theory, maintenance should require the modification of small parts of an OO design. Possibly, the size of the system rendered some metrics less useful than might have been the case in a larger system. (c) The participants may not be representative of OO software professionals. Their background and experience in this area also differ. However, they are software engineers teaching and researching at a University. Particularly, two of them, 'A1' and 'B2', specialised in OO technology. (d) We also only studied four individuals. A few limitations to the usage of a small number of participants are that: the method is time and resource consuming, and in studying a great number of participants there is a risk of dissemination of the study taking place.

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. Observational 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.

6. Conclusions and future work

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, which was developed according to design heuristics in general, and Design B, violating the 'god class', in a particular but functionally important part of it. 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 show that design heuristics can affect maintainability. This is supported by qualitative evidence by two means provided by: (a) the participants in the form of information from their debriefing questionnaires; they expressed the views that Design A was easier to understand and modify, and (b) the delivered solutions evaluated by the authors. It is also supported by quantitative evidence provided by: (a) measurement values, and (b) the performance effort, measured by the time spent, that were significantly positive toward Design A.

Focusing on what factors have mostly affected the performance of the participants, we identified the following three. First the coupling between classes, considered to be most significant, since it was mentioned by most of the participants. Tight coupling leads to monolithic systems that are hard to learn, maintain and reuse. On the other hand, loose coupling increases the probability that a class can be reused, learned, modified and extended more easily. Second was the cohesion, mentioned as distribution of data across objects and declarations. Third, mentioned only by one participant, was syntax and naming conventions. In our case, Design B suffers from high coupling and low cohesion as measurement values show too.

Our findings provide a number of indications that OO design structures are sensitive to bad or good design techniques. It would seem 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. If design heuristics are violated even once, there is an increased probability of maintenance changes leading to poorer designs, thus rendering it harder to maintain as it evolves over time. Such a design also minimises opportunities for reuse, an important feature in OO technology. Regarding the relationships between heuristics and metrics, there is also a strong indication that the specific design heuristic we investigated could be captured by a suitable set of metrics as that shown in Table 5.

6.1. Future work

We would emphasize the point that this research is regarded as exploratory and we are in the process of building upon it. Further research, planned as a result of this observational study, includes a further investigation into the impact of design heuristics on maintainability as well their relationship to metrics on an experimental basis. Our review of experimental research into OO technology (Deligiannis et al., 2002) highlights the need for more empirical work to evaluate and refine the burgeoning number of heuristics as well as other factors. Thus, our next goal is to design and conduct a formal experiment where these research questions could be more accurately addressed.

Acknowledgements

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

Appendix A. Modification task

A.1. Asking for a tariff order

In order to tempt the user to buy a tariff, after the system has displayed the best option and asking for 'Print' or 'Cancel', a window will be displayed if the 'Print' option is selected, offering the customer a 10% reduction if an order is placed immediately.

First, the user is asked to whether he/she wishes to take advantage of the special offer.

If he/she clicks the 'Yes' button then will be prompted to enter address, telephone number, and credit card account holder, number of expiry date details. The only validation will be for non-empty fields.

This data, as well as the tariff's name, will be logged into the log file.

Appendix B. Debriefing questionnaire

Nr	Factor	Question
Experience		
1	none—1, little—2, average—3, substantial—4, professional—5	In software engineering practice
2		In design documents in general
3		In object-oriented design documents
Understanding		
4	1—not, 2—poorly, 3—fairly, 4—well, 5—highly	Estimate how well you understood what was required of you
5	1—barely understandable, 10—easily understandable.	Estimate, in terms of understandability, the quality of the design documents you had
6		In your opinion, what caused you the most difficulty to understand the design documents?
7	1—very little, 10—complete.	Estimate your overall understanding of the design documents
8		What did you understand least about the design documents and why?
Maintaining		
9		In your opinion, what caused you the most difficulty to perform the modification task on the design documents?
10	In %	Estimate the correctness of your solutions to the modification tasks
11	1—barely modifiable, 10—easily modifiable	Estimate, in terms of modifiability, the quality of the design documents you had

Nr	Factor	Question
12	1—very easy, 10—very difficult	Estimate the overall difficulty of the tasks you have been asked to perform
Miscellaneous		
13	1—too small, 5—too large	How do you judge the size of the design documents you had?
14		Having performed the tasks, would you do anything different next time around?
15	Yes/no	Have ever owned a mobile phone?

References

Abreu, F., Carapuca, R., 1994. Candidate metrics for object-oriented software within a taxonomy framework. Journal of Systems and Software 26, 87–96.

Abreu, F., Goulao, M., Esteves, R., 1995. Toward the design quality evaluation of object-oriented software systems. In: Proc. 5th International Conference on Software Quality, Austin, TX, USA.

Abreu, F., Melo, W., 1996. Evaluating the impact of object-oriented design on software quality. In: Proc. IEEE 3rd Intl. Metrics Symp Mar.

Berard, E., Metrics for object-oriented engineering. Available from <http://www.toa.com/pub/moose.htm>.

Booch, G., 1994. Object-Oriented Analysis and Design with Applications. Addison-Wesley Publishing Company.

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

Booch, G., Rumbaugh, J., Jacobson, I., 1999. The Unified Modeling Language User Guide. ACM Press, Reading, MA: Addison-Wesley.

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

Chidamber, S.R., Kemerer, C.F., 1994. A metrics suit for object oriented design. IEEE Transactions on Software Engineering 20 (6), 476–493.

Coad, P., 1991. OOD Criteria, Part1–3. Journal of Objet-Oriented Programming 4 (2–4).

Coplien, J.O., 1992. Advanced C++: Programming Styles and Idioms. Addison-Wesley.

Deligiannis, I., Shepperd, M., Webster, S., Roumeliotis, M., 2002. A review of experimental investigations into object-oriented technology. Empirical Software Engineering, 7 (3), 193–231.

Fenton, N., Pfleeger, S.L., 1997. Software Metrics—A Rigorous and Practical Approach, second ed. International Thompson Computer Press.

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

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

Hatton, L., 1998. Does OO Sync with how we think? IEEE Software May/June, 46–54.

Kirsopp, C., Shepperd, M., Webster, S., 1999. An empirical study into the use of measurement to support OO design evaluation. In: IEEE 6th Intl. Metrics Symp. Boca Raton, FL.

Kirwan, B., Ainsworth, L.K., 1992. In: A Guide to Task Analysis. Tailor & Fransis.

Laitenberger, O., Atkinson, C., Schlich, M., El Emam, K., 2000. An experimental comparison of reading techniques for defect detection in UML design documents. Journal of Systems and Software 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, Upper Saddle River, NJ.

Riel, A., 1994. Tutorial 38 handout: Object-Oriented design through heuristics. OOPSLA.'94, Portland, Oregon.

Riel, A., 1996. Object-Oriented Design Heuristics. Addison-Wesley Publishing Company Inc.

Rumbaugh, M., Blaha, M., Premerhani, W., Eddy, F., Lorensen, W., 1991. Object-oriented modeling and design. Prentice Hall, Englewood Cliffs, NJ.

Whitmire, S., 1997. Object oriented design measurement. John Wiley & sons.

Wilde, N., Mathews, P., Ross, H., 1993. Maintaining Object-Oriented Software. IEEE Software January, 75–80.

Ignatios Deligiannis is a Lecturer at Technological Education Institute of Thessaloniki, Greece, since 1990, and a research associate at the University of Macedonia and the Aristotle University, Greece. He is also member of ESERG (Empirical Software Engineering Research Group at Bournemouth University, UK). His main interests are object-oriented software assessment, and in particular design heuristics and measurement. He received his B.Sc. in computer science from the University of Lund, Sweden, in 1979, and then worked for several years in software development at Siemens Telecommunications industry.

Martin Shepperd is a Professor of Software Engineering at Bournemouth University. He received a Ph.D. in computer science form the Open University in 1991. He has published more then 60 referred papers and three books in the field of empirical software engineering. He is also an editor of the journal Information and Software Technology. His research interests include software metrics and empirical software engineering.

Manos Roumeliotis received the Diploma in electrical engineering from the Aristotle University of Thessaloniki, Greece in 1981, and the MS and Ph.D. degrees in computer engineering from Virginia Polytechnic Institute and State University, Virginia, USA, in 1983 and 1986, respectively. At VPI & SU he taught as a visiting Assistant Professor in 1986. From 1986 to 1989 he was an Assistant Professor in the Department of Electrical and Computer Engineering at West Virginia University. Currently he is an Assistant Professor in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. His research interests include digital logic simulation and testing, computer architecture and parallel processing, and computer network optimization. He is a member of the IEEE Computer Society's Technical Committee on Computer Architecture.

Ioannis Stamelos is a Lecturer of computer science at the Aristotle University of Thessaloniki, Department of Informatics, since 1997. He received a degree in Electrical Engineering from the Polytechnic School of Thessaloniki (1983) and the Ph.D. degree in computer science from the Aristotle University of Thessaloniki (1988). He has worked for 10 years as a researcher in the telecommunications software industry in Italy and Greece. His research interests include software evaluation and management, software cost estimation and software measurement. He is author of 30 scientific papers. He is a member of the IEEE Computer Society.