Difficulties experienced by students in maintaining object-oriented systems: an empirical study

Amela Karahasanović
Simula Research Laboratory
P.O. Box 134, NO-1325 Lysaker, Norway
amela@simula.no

Richard C. Thomas
Computer Science & Software Engineering,
M002, The University of Western Australia,
Crawley, Western Australia 6009, Australia
richard@csse.uwa.edu.au

Abstract

It is widely accepted that software maintenance absorbs a significant amount of the effort expended in software development. Proper training of both university students and professional developers is required in order to improve software maintenance. Understanding cognitive difficulties the students have while maintaining object-oriented systems is a prerequisite for improving their university education and preparing them for jobs in industry. The goal of the experiment reported in this paper is to explore the difficulties of students who maintain an unfamiliar object-oriented system. The subjects were 34 students in their third year of study in computer science. They used a professional Java tool to perform several maintenance tasks on a medium-size Java application system in a seven-hour long experiment. The major difficulties were related to understanding program logic, algorithms, finding change impacts, and inheritance of the functionality. Based on these results we suggest teaching the basics of impact analysis and introducing examples of modifying larger object-oriented programs in courses on object-oriented programming.

1 Introduction

Software maintenance has been widely recognised as a dominating cost factor in most software organisations. Although the reported figures differ, most researchers on software maintenance agree that more than 50% of programming effort is constituted by changes made to the system after the implementation (Coleman *et al.*, 1994; Holgeid *et al.*, 2000; Lehman and Belady, 1985; Lientz, 1983; Nosek and Prashant, 1990; Pfleeger, 1987; Zelkowitz, 1978).

Whereas difficulties the students have during the design of object-oriented systems are relatively well understood, rather less attention has been paid to their difficulties during maintenance. The study reported in this paper explores difficulties of programmers when conducting maintenance tasks.

The program used in this study was a 3600 lines of code (LOC) large library application system written in Java and can be considered to be a medium-size application according to the classification given by von Mayrhauser and Vans (1995). The participants were 34 students in their third year of study in computer science at the University of Western Australia (UWA). The experiment lasted seven hours and the participants conducted three maintenance tasks on the given application system, using JBuilder. The participants were provided with documentation that describes the application system and the JBuilder documentation. They had access to the Java online documentation.

1.1 Background

In spite of their complexity, maintenance tasks are often given to beginners and less experienced developers (Gunderman, 1988). To improve maintenance proper training of both university students and professional developers is required (Kajko-Mattsson *et al.*, 2002). Furthermore, it requires the provision of meaningful feedback to maintainers (Jørgensen and Sjøberg, 2002).

Object-oriented programming has become a *de facto* standard and therefore we need to understand the problems of maintaining such systems. Object-oriented programming is increasingly being taught in computer science courses. A survey conducted by Dale (2005a; 2005b) shows that 65% of the participating educational institutions teach object-oriented programming as a part of introductory courses in computer science education. Some of these students are going to have to maintain object-oriented systems when they start work.

It is therefore important to understand the cognitive difficulties the students have while maintaining object-oriented systems. It is a prerequisite for improving their university education and preparing them to enter the maintenance workforce.

Several studies have been conducted on the cognitive consequences of the object-oriented approach in the context of software design (Détienne, 1997). In her survey of empirical research on object-oriented design, Détienne (1997) gives a comprehensive list of difficulties experienced by individuals (novices and experts) and teams during the design of object-oriented systems. Novice designers have been found to have problems with class creation and with articulating the declarative and procedural aspects of the solution. They also had misconceptions about some fundamental object-oriented concepts.

In a previous study Karahasanovic *et al.* (2006), explored the strategies and difficulties of programmers when conducting maintenance tasks. The participants were 38 students in their third or fourth year of study in computer science at either the University of Oslo or Oslo University College, and can be considered as advanced beginners according to the classification of Dreyfus and Dreyfus (1986). They conducted three maintenance tasks on a Java library application system, using JBuilder. The results showed that two major groups of difficulties were the comprehension of the application structure (identifying GUI components affected by a change, identifying classes affected by a change, identifying impacts of a change within a class) and using the inheritance of functionality. Furthermore the subjects had difficulties with the GUI, understanding and using a given Java API class, and algorithms.

The ability to generalize these results to the target population of advanced beginners, i.e., external validity of this study can be questioned. It is recommended that a way to identify potentially important factors that affect the process under investigation is to replicate the study with variations in the context variables (Basili *et al.*, 1999). Thus, our earlier findings need to be tested through replications with subjects from a slightly different educational background.

The present investigation is a replication of the study conducted with students in Norway by Karahasanovic *et al.* (2006) with a major difference that the subjects were from a university in another country and had a slightly different syllabus. It aims to identify the difficulties that students have while conducting maintenance tasks on a medium-size object-oriented application.

The remainder of this paper is organised as follows. Section 2 describes the methodology. Section 3 presents the results. Section 4 discusses the limitations of this research. Section 5 concludes and suggests avenues for further work.

2 Research Method

The main goals of the experiment were:

- To identify students' difficulties in conducting changes on a medium-size object-oriented application,
- To conduct an analysis of students' comprehension strategies on a medium-size object-oriented application, and
- To replicate and extend the earlier investigation by Thomas et al. (2005) into keystroke latency metrics as an indicator of programming performance.

This paper reports results regarding the first goal. The results regarding other goals are outside the scope of this paper.

The experimental material from the original experiment conducted by Karahasanovic *et al.* (2006) was translated from the Norwegian by the authors. This experiment had two treatments (Feedback Collection and Control Silent), whereas the original experiment had two more treatments (Concurrent Think-Aloud and Retrospective Think-Aloud) as it aimed to evaluate different think-aloud methods. A short copy typing test was introduced in this experiment to be used in the keystroke latency investigation. Otherwise, the experimental design and the material were the same in both experiments.

2.1 Experimental Design and Participants

A randomised design was employed: participants were randomly assigned into one of two treatment groups. There were 34 participants, half in each group.

All the participants were students at UWA. They were mainly in their third year of study in the School of Computer Science & Software Engineering and were asked to volunteer via an email sent to everyone taking a third or fourth level unit. The normal minimum attainment was to have passed two second level units: *Data Structures and Algorithms* and *Object Oriented Programming*. They will also have passed *Java Programming* from level one. About half the participants were on double degrees, such as Bachelor of Computer Science and Bachelor of Engineering; these have high cut off grades for entry. The remainder were taking single degrees, such as Bachelor of Computer Science. Everyone was male, aged 19-47, mean 22.

The protocols reported in this paper were approved by the university's Human Research Ethics Committee prior to the commencement of the recruitment of participants.

2.2 Treatment

The two treatments were Feedback Collection (FC) and Control Silent (CS). In Feedback Collection the participants were asked every 15 minutes "What are you thinking now?" This was delivered through the feedback collection screen that appeared for two minutes during the change tasks (Karahasanovic *et al.*, 2005). Participants could write whatever they wanted in that period and if they did not close the window it would automatically disappear after two minutes. In Control Silent, the tasks were the same but there was no feedback collection.

2.3 Procedure

Sessions were organised for 7 separate days, each testing 2-7 students, and held in the computer science laboratories. Sessions would start at 09:15 with an information meeting; continue through to lunch, provided around 12:30-13:00, and finish about 16:15. Participants were paid an honorarium of A$100. Two observers were present in the laboratory during the experiments to answer questions and to provide help if any technical problems arose.

The first task was a short copy typing test, followed by a background questionnaire administered over the web using the Simula Experiment Support Environment (SESE) (Arisholm *et al.*, 2002). Next, everyone solved a simple training task and then a calibration task. Following this, those in the FC group were trained on providing written feedback, while CS members started on the change tasks. Everyone attempted three change tasks and then an exit questionnaire. Lastly there were group interviews.

2.4 Tasks

The subjects were asked to conduct a small training task and a pre-test task. The purpose of the training task was to make subjects familiar with SESE and the experimental situation. The participants downloaded the task, created a Java program to write a string in reverse order and uploaded their solutions. The pre-task was to extend the functionality of a bank teller machine program. This application was a small Java program consisting of seven Java classes and about 400 LOC. The task was to extend the program to provide a printout of all successfully performed transactions (deposits and withdrawals) for a given bank account. The purpose of the pre-test task was to provide a basis for comparing the programming skill level of the subjects. These tasks were taken from (Arisholm *et al.*, 2001).

The tasks of the experiment were to modify a library application system given in Eriksson and Penker, (1998). A library lends books and magazines. The books and the magazines are registered in the system. A library handles the purchase of new titles for the library. Popular titles are bought in multiple copies. Old books and magazines are removed when they are out of date or in a poor condition. The librarians can easily create, update, delete and browse information about the titles in the system. The borrowers can browse information about the titles. They can reserve a title if it is not available. The application consists of four packages with a total of 3600 LOC in 26 Java classes. This application system was used because we assumed that the application domain is very familiar to students. The subjects were asked to conduct the following changes on the library application system:

Task1	Delete functionality related to ISBN number
Task2	Extend the system to handle customer e-mail address
Task3	Introduce the functionality to inform a person when a loan is due

The tasks were ordered by complexity. The subjects were provided with documentation describing the library application system in addition to the normal JBuilder documentation. They also had access to the Java online documentation. We emphasized that subjects should give higher priority to the quality of solutions rather than to shorter development time.

2.5 Data Collection and Supporting Tools

A Web-based tool, the Simula Experiment Support Environment (SESE) (Arisholm *et al.*, 2002) was used for logistics support. The subjects used this tool to answer the background questionnaire, to download the documents and code, to upload their solutions and to provide feedback (FCM group only). The tool recorded start-time and end-time for each task. The typing test was distributed by the observers. Keystrokes, mouse-clicks and window focus events were logged with timestamps in milliseconds by the GRUMPS-Lite software (Thomas *et al.*, 2003). The programming environment was Borland JBuilder, chosen as it was used in prior experiments at Simula. The pretest questionnaire confirmed that most students had little or no prior experience with JBuilder. Observers made notes during the experiment.

2.6 Data Preparation and Analysis

Cognitive difficulties the participants had while solving the given tasks were identified from the collected feedback and from their solutions.

2.6.1 Collected Feedback

Information collected by the feedback-collection tool was first categorised in four broad categories: experimental context, subjects' perception, experimental conduct and interaction. The feedback-collection data was then analysed and the information about the comprehension problems and background knowledge was used to make a list of difficulties for each participant. The coding took about 16 working hours.

2.6.2 Participants' Solutions

The assessment of the participants' solutions (correctness and problems) was done by a PhD student who was not involved in this research. She was provided with task specifications, correct solutions and guidelines for giving scores. All solutions were compiled, executed and thoroughly tested for functionality. The source code was also manually inspected. Questionable cases were resolved through discussion with the researchers. Based on this analysis we made a list of problems for each participant. The assessment of the solutions took about 80 working hours.

3 Results

The participants experienced different difficulties while conducting change tasks. We first give an overview of the difficulties identified as a result of examining the participants' solutions and collected feedback. We then describe the major difficulties in greater detail.

To identify the difficulties participants had, we first analysed the assessor's report and made a detailed list of errors for each participant. We then extended this list with the difficulties that we found in the collected feedback.

As in the original experiment conducted by Karahasanovic *et al.* (2006), the difficulties were then categorised within a refinement of the model of von Mayrhauser and Vans (1995). Von Mayrhauser and Vans describe two types of knowledge: (i) general knowledge, which is independent of the specific software application that the programmers are trying to understand, and (ii) software-specific knowledge, which represents their level of understanding of the software application. They suggest that difficulties and errors in comprehension arise from a lack of either general, or specific knowledge, or both. Among the difficulties caused by the lack of general knowledge, we identified the following sub-categories: difficulties concerning program logic, graphical user interface (GUI), object-oriented programming, algorithms and programming environment. Among the difficulties caused by the lack of the specific knowledge, we identified the following three sub-categories: difficulties concerning the GUI, object-oriented programming and testing procedure.

Table 1 gives an overview of the difficulties. A difficulty is presented only once per participant per task. As described in Section 3, Task 3 was given as an extra change task and the majority of participants did not complete it. Consequently, the numbers of difficulties per category reported for Task 3 will not give us a complete picture. However, we present them here because they provide additional information about the difficulties that the participants experienced.

Two major general difficulties that prevented the participants for completing the task or caused a significant delay were related to program logic (23 occurrences) and algorithms (10 occurrences). The participants had two types of problems related to program logic (category 1.1). The first one was related to an if-then-else statement. This statement implements a book search on title, author or ISBN. The participants removed either too much or too little from this statement and, as a result, the library application did not work properly. It might be that the participants interpreted this task as a pure text editing task (finding all ISBN occurrences and deleting them) and did not try to understand the logic of the program. However, it is also possible that the participants felt time pressure and were not sufficiently careful. Another was related to finding a given title. Two participants explicitly reported that they had problems understanding the logic of this part of the application.

Another difficulty that was reported relatively frequently concerned knowledge of algorithms or the application domain (category 1.4). Ten participants failed to calculate the expiry date correctly, or did not try to calculate it at all. They also reported in the collected feedback that this is difficult. One usually uses library classes for different conversions and calculations. Therefore, it might be that making their own calculations was difficult for these students. Furthermore, the array index started from zero in this library class, which might be unusual for the students, who mostly programmed in the programming language Java.

Two specific difficulties that frequently occurred were related to adding or removing GUI components (category 2.1.2, 17 occurrences) and removing label declarations (category 2.1.3, 19 occurrences). The participants either forgot to add or remove different components of the GUI-like radio buttons and text fields or failed to remove all label declarations of ISBN in Task 1. It was clearly stated in the task description that all references to ISBN should be removed. However, leaving some of these ISBN declarations had no effect on the functionality of the application.

The participants in the present study appeared to have particular difficulty with the GUI components (category 2.1.2). It should be noted that many of the present people would not have studied the design and implementation of GUIs by the time they participated in this experiment; this is covered towards the end of the degree. The given tasks required no special proficiency in GUI programming, but they required basic understanding of impact analysis. Familiarity with a domain (GUI in this case) could make impact analysis easier for the students.

Two major specific difficulties that prevented the participants from completing the task or caused a significant delay were related to finding impacts on classes (category 2.2.2, 14 occurrences) and inherited functionality (category 2.2.4, 12 occurrences). To conduct change tasks, the participants had to comprehend the structure of a medium-sized application object-oriented application. They had to comprehend relationships between the classes and to find the classes affected by a change. It seems that this was difficult for them.

Furthermore, the participants needed to understand inheritance of the functionality to solve the tasks. Classes in the library application that needed to be persistent had to inherit an abstract class called Persistent. The subclasses of the Persistent class had to implement the methods *read()* and *write()*, which reads/writes from/to a file. The failure to make data persistent occurred relatively often thus indicating that this was difficult for the participants.

Compared with the previous experiment, these students were more challenged to understand impacts on classes but performed better on inherited functionality. The latter is a specific topic in the Object Oriented Programming unit considered as a core attainment (section 2.1). In contrast comprehending the structure of a medium-sized object oriented system may not have been studied by some students and this may have reduced their comprehension of impacts.

The present cohort has relatively few problems with attributes and methods in the wrong class (2.2.2.2-3) or their removal (2.2.3.1). This may have been because of the importance placed on understanding the fundamentals of classes, objects and methods in the Java Programming foundation unit. The BlueJ environment [1], used for this unit, is especially good for illustrating these concepts, perhaps at the expense of practice with larger programs.

	Task1	Task2	Task3
1 General			
1.1 Program logic	20	3	
1.2 GUI			
1.2.1 Forgot to expand the window		6	
1.2.2 Little experience with GUI programming	2	1	
1.3 Object-oriented programming			
1.3.1 Initialise objects			2
1.3.2 Instantiate a class			1
1.3.3 Understand and use a Java API class			2
1.3.4 Reuse of methods		2	
1.4. Algorithms			10
1.5 Programming environment	1		1
2 Specific			
2.1 GUI			
2.1.1 Changing interface		1	
2.1.2 Adding or removing GUI components	10	4	3
2.1.3 Removing label declarations	19		
2.2 OO comprehension and programming			
2.2.1 Overall program structure	2	2	
2.2.2 Impacts on classes	1	10	3
2.2.2.1 Self-reported problems			
2.2.2.2 Attributes in the wrong class			2
2.2.2.3 Methods in the wrong class			
2.2.3 Impacts within class	1		
2.2.3.1 Removing variables and methods	3		
2.2.4 Inherited functionality	2	6	4
2.3 Testing procedure	1	1	1

Table 1: Number and type of difficulties per tasks

[1] www.bluej.org

4 Limitations of this Study

The data on difficulties experienced by the participants are partly qualitative and subjective. A detailed list of errors for each participant made by the independent assessor (difficulties that the participants did not overcome) was combined with the difficulties identified in the collected feedback. However, one should be aware that there might be differences among participants. Some might have forgotten to report their difficulties. Hence, some of the difficulties the participants had during the experiment that they managed to overcome might be missing from our list.

One should also be aware that the majority of the participants did not finish Task 3. Hence, the list of difficulties for this task is not complete.

The experiment lasted only seven hours, which might be too short a time to become familiar with the application. Future work should therefore include case studies that last longer.

5 Conclusions and future work

This paper provides further empirical evidence with respect to the difficulties students had while conducting maintenance tasks on a medium-sized Java application. It allows generalisation of the results of the previous study by Karahasanovic *et al.* (2006) to the population of advanced beginners. The results revealed the difficulties the participants had due to a lack of knowledge that is independent of the specific application (general knowledge) and a lack of knowledge of the specific application (specific knowledge). The major general difficulties that prevented the participants from completing the tasks were related to program logic and algorithms. These difficulties were also identified in the previous experiment. These findings can be used for improving courses on data algorithms.

The major specific difficulties that prevented the participants from completing the tasks were related to finding impacts of changes (removing label declarations and impacts on classes) and inheritance of functionality. The same difficulties were identified in the previous experiment. However, there were some differences. UWA students were more challenged to understand impacts on classes than students in Norway. On the other hand, UWA students performed better on inherited functionality. This can be explained by a different syllabus within a broadly similar degree. Findings on differences between universities in different countries can be used for further improvement of their syllabuses.

Based on these results we recommend introducing examples of modifying larger object-oriented programs in courses of object-orientation. Students should learn the basics of impact analysis earlier in their computer science education. However, this does not mean that the training in understanding the fundamentals of classes, objects and methods as provided at UWA should be reduced. The BlueJ environment can be recommended for illustrating the fundamentals of object-orientation.

What needs to be taught to improve effectiveness at maintenance is a complex question. Efforts have been made by the community to introduce the theory and practice of maintenance in computer science education (Austin and Samadzadeh, 2005; Postema *et al.*, 2001). Nevertheless, large number of students would leave their universities without any maintenance experience. Furthermore, their understanding of object-oriented concepts gained through the introductory programming courses affects their ability to maintain such systems later on. We thus believe that students should obtain some experience in understanding and modifying larger programs earlier in their education. Identifying difficulties the students had while conducting maintenance tasks is only a first step towards improving their education. We intend to further explore interactions between programming and general problem-solving knowledge, and the effects of these interactions on the students' ability to maintain larger object-oriented applications.

Acknowledgements

Amela Karahasanovic acknowledges support from the University of Western Australia for the Gledden Visiting Senior Fellowship.
The authors are grateful to Kesaraporn Techapichetvanich and Junisilver Taij for valuable contributions to this paper. We thank Gunnar Carelius, Ashley Chew, Ryan McConigley and Laurie McKeaig for their considerable technical help, and Louise Bolitho and Jeff Pollard for administrative help. We are grateful to Kaja Kværn and Gøril Tømmerberg for preparing experimental materials. We thank the students of UWA for participating in the experiment.

References

Arisholm, E., Sjøberg, D.I.K., Carelius, G. and Lindsjørn, Y. 2002. A Web-based Support Environment for Software Engineering Experiments. *Nordic Journal of Computing*, 9, No. 4, 231–247.

Arisholm, E., Sjøberg, D.I.K. and Jørgensen, M. 2001. Assessing the Changeability of two Object-Oriented Design Alternatives – a Controlled Experiment. *Empirical Software Engineering*, 6, No. 3, 231–277.

Austin, M.A. and Samadzadeh, M.H. 2005. Software comprehension/maintenance: an introductory cours. *18th Int. Conf. on Systems Engineering (ISCEng'05),* IEEE, 414–419.

Basili, V.R., Shull, F. and Lanubile, F. 1999. Building Knowledge through Families of Experiments. *IEEE Transactions on Software Engineering*, 25, No. 4 (July–Aug.), 456–73.

Coleman, D., Ash, D., Lowther, B. and Oman, P. 1994. Using Metrics to Evaluate Software System Maintainability. *IEEE Computer, August 1994*, 44–49.

Dale, N. 2005a. SIGCSE members survey. http://www.cs.utexas.edu/users/ndale/ContentResults.html

Dale, N. 2005b. Non SIGCSE members survey. http://www.cs.utexas.edu/users/ndale/ContentResults2.html

Détienne, F. 1997. Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams. *Interacting with Computers*, 9, 47–72.

Dreyfus, H.L. and Dreyfus, S.E. 1986. *Mind over Machine*. New York: The Free Press.

Eriksson, H.E. and Penker, M. (1998). "Case Study". In: *UML Toolkit*. (editors), New York, John Wiley & Sons, Inc.,

Gunderman, R.E. (1988). "A glimpse into program maintenance". In: *Techniques of program and system maintenance*. Parikh, G. (editors), Wellesley, MA, QED Information Sciences Inc., 55–59.

Holgeid, K.K., Krogstie, J. and Sjøberg, D.I.K. 2000. A Study of Development and Maintenance in Norway: Assessing the Efficiency of Information Systems Support Using Functional Maintenance. *Information and Software Technology*, 42, No. 10, 687–700.

Jørgensen, M. and Sjøberg, D.I.K. 2002. Impact of experience on maintenance skills. *Journal of Software Maintenance and Evolution: Research and Practice*, 14, 123–146.

Kajko-Mattsson, M., Forssander, S. and Andersson, G. 2002. Developing CM3: Maintainers' Education and Training at ABB. *Computer Science Education*, 12, No.1–2, 57–89.

Karahasanovic, A., Anda, B., Arisholm, E., Hove, S.E., Jørgensen, M., Sjøberg, D.I.K. and Welland, R. 2005. Collecting Feedback during Software Engineering Experiments. *Empirical Software Engineering*, 10, No. 2, 113–147.

Karahasanovic, A., Levine, A.K. and Thomas, R. 2006. Comprehension strategies and difficulties in maintaining object-oriented systems: an explorative study, *Journal of Systems and Software*. Forthcoming.

Lehman, M. and Belady, L.A. 1985. *Program Evolution – Processes of Software Change*. Academic Press, London.

Lientz, B.P. 1983. Issues in Software Maintenance. *Computing Surveys*, 15 (3), 271–278.

Nosek, J.T. and Prashant, P. 1990. Software Maintenance Management: Change in the Last Decade. *Journal of*

Software Maintenance Research and Practice, 2, No. 3, 157–174.

Pfleeger, S.L. 1987. *Software Engineering – The Production of Quality Software*. Macmillan.

Postema, M., Miller, J. and Dick, M. 2001. Including practical software evolution in software engineering education. *14th Conference on Software Engineering Education and Training,* IEEE, 127–135.

Thomas, R., A. Karahasanovic, and Kennedy, G. 2005. An Investigation into Keystroke Metrics as an Indicator of Programming Performance. Proc. of the 7th Australasian Computing Education Conference 2005 (ACE 2005), Conference in Research and Practice in Information Technology, Newcastle, Australia, Australian Computer Society, Inc., 42, 127–134.

Thomas, R., Kennedy, G., Draper, S., Mancy, R., Crease, M., Evans, H. and Gray, P. 2003. Generic Usage Monitoring of Programming Students. *ASCILITE 2003 Conference, University of Adelaide, Australia,* Adelaide, Australia, ASCILITE, 715–719.

Von Mayerhauser, A. and Vans, A.M. 1995. Industrial Experience with an Integrated Comprehension Model. *Software Engineering Journal*, 171–182.

Von Mayrhauser, A. and Vans, A.M. 1995. Program Comprehension During Software Maintenance and Evolution. *Computer*, 28(8), 44–55.

Zelkowitz, M.V. 1978. Perspectives on Software Engineering. *ACM Computing Surveys*, 10, No. 2, 197–216.