

UNIVERSITY OF OSLO
Department of informatics

**Comprehension-Related
Activities during Maintenance
of Object-Oriented Systems: An
In-Depth Study**

Master thesis
60 credits

Gøril Tømmerberg

May, 1st 2006



Abstract

The research presented in this thesis identifies professional programmers' comprehension-related activities during the phases of maintenance tasks on a previously unknown Java application. Program comprehension is the process of understanding an unfamiliar program and is a vital task in software maintenance. The extended knowledge of professional programmers' activities is thus useful to improve the maintenance process related to education in successful techniques. Furthermore it is useful for other researchers conducting studies of program comprehension

A controlled experiment was conducted with 24 participants. The participants used a professional programming environment, JBuilder, to perform three maintenance tasks on a 3600 LOC Java application. The participants' actions were logged and written feedback was collected.

We have made a further development of GRUMPS which includes data cleaning and analytical preparation of low-level usage data. The extended functionality provides detailed information about each participant's use of compilation, execution and various documentations. The analysis tool gives an overview of how the participants solved the tasks. It shows the chronological actions and time spent on source code, compilation, execution, web pages and documentation throughout the experiment. This information is useful regarding program comprehension research because we can identify the participants' comprehension-related activities when familiarizing themselves with an unknown application and during the maintenance tasks.

The participants used source code, system documentation and execution to understand the application. The most important finding was that program execution proved to be important to get an understanding of the system's dynamic functions. The lack of facilities like compilation and execution in previous comprehension studies can thus cause threats to the validity of their findings.

The use of information sources changed during the phases of the experiment and it was a distinct difference between the initial phase and the rest of the tasks. As expected the source code was most used in all phases, but in the initial phase the system documentation was nearly as much used. Compilation and execution were often performed throughout the experiment. In addition to test the modifications, compilation was also used in the corrective task to locate source code that had to be altered. The participants' use of internet and web pages appeared to be on an "as-needed" basis. They used the Java API web pages when they needed more information of special Java classes. The participants work cycles consisted of program modifications, reading documentation, compilation and execution.

In addition the work practises of the two participants who performed best and the two who performed worst were identified. The results showed that the two best executed the application more, compiled less and accessed fewer classes. The two best had a more successful employment of the JBuilder programming environment.

Acknowledgements

First I would like to thank my supervisor Amela Karahasanović. I am very grateful for her valuable guidance, encouragement and support during the work with the thesis. I thank Richard Thomas for the useful documentation and SQL-code regarding the GRUMPS database and Gunnar Carelius for his technical assistance. I thank Johan Almqvist and Rolf Vassdokken for preparing and testing the experimental material and the participant in the experiment. I also thank the Simula Research Laboratory for supportive environment.

My special thanks to Kaja Kværn for being such an excellent co-worker during the master study and this research. In addition I thank Jørgen Busvold for the research cooperation and Line Joakimsen and Kristrún Arnarsdóttir for inspiring and great team-work. Many thanks also to David Skogan, Cecilie Mohr and Ingvild Tømmerberg who read the thesis and gave me valuable feedback.

Finally, I would like to thank my husband Kjell for encouraging me to start at the master study and all support during the study. Thanks also to my two boys, Kristian and Sigurd, who have been very patient.

Oslo, May 2006
Gøril Tømmerberg

Table of contents

1	Introduction	11
1.1	<i>Motivation.....</i>	11
1.2	<i>Objective</i>	12
1.3	<i>Research Method</i>	12
1.4	<i>Research Context</i>	12
1.5	<i>Contribution.....</i>	13
1.6	<i>Structure.....</i>	14
2	Related Work.....	17
2.1	<i>Identification of Related Work</i>	17
2.2	<i>Program Comprehension Strategies.....</i>	17
2.2.1	Direction of Comprehension	18
2.2.2	Breadth of Comprehension.....	18
2.3	<i>Empirical Studies of Comprehension Strategies.....</i>	19
2.4	<i>Work Practises.....</i>	23
2.4.1	Use of the Information Sources.....	23
2.4.2	Work Cycles.....	24
2.5	<i>Summary</i>	25
3	Methodology.....	27
3.1	<i>Participants.....</i>	27
3.2	<i>Experimental Procedure</i>	27
3.3	<i>Tools and Data Collection.....</i>	28
3.3.1	GRUMPS Description.....	28
3.4	<i>Tasks</i>	32
3.5	<i>The Application.....</i>	33
3.5.1	Architectural Overview	33
3.6	<i>Documentation.....</i>	34
4	Analysis.....	35
4.1	<i>Measures Used in Previous Studies</i>	35
4.2	<i>Analysis model.....</i>	36
4.2.1	Data Preparation for Analysis	36
4.2.2	Correctness.....	37
4.2.3	Solution Time.....	37
4.3	<i>Initial Strategy</i>	37
4.3.1	Data from GRUMPS.....	38
4.3.2	Data from Feedback-Collection	40
4.4	<i>Work Practises during the Tasks.....</i>	40
4.4.1	Direction of Comprehension	41

4.4.2	Detailed Use of Information Sources	41
4.4.3	Data from GRUMPS	42
4.4.4	Data from Feedback-Collection	45
5	Results	47
5.1	<i>Initial Strategy</i>	47
5.2	<i>Work Practises during All Tasks</i>	48
5.2.1	Work Cycles.....	50
5.3	<i>Detailed Use of the Information Sources</i>	52
5.3.1	Source Code	52
5.3.2	Documentation	54
5.3.3	Compilation.....	55
5.3.4	Execution	56
5.4	<i>Work Practises of the Best and Worst Participants</i>	57
5.4.1	Initial Strategy	57
5.4.2	Work Practise during the Tasks.....	58
5.4.3	Detailed Use of the Information Sources	59
5.5	<i>Summary and Discussion</i>	60
5.5.1	Initial Strategy	60
5.5.2	Work practises during the tasks.....	61
5.5.3	Work Practises of Best and Worst Participants	63
6	Validity	65
6.1	<i>Experimental Design</i>	65
6.2	<i>Programs and Tasks</i>	65
6.3	<i>Participants</i>	66
6.4	<i>Data Cleaning and Preparation</i>	66
6.5	<i>Measuring Time</i>	66
6.6	<i>Analysis</i>	67
7	Conclusions and Future Work	69
7.1	<i>GRUMPS</i>	69
7.2	<i>Initial Strategy</i>	69
7.3	<i>Work Practises during the Tasks</i>	70
7.4	<i>Best and Worst Participants</i>	71
7.5	<i>Future Work</i>	71
	Bibliography	73
	Appendix A - Tasks.....	77
	Appendix B - The Application	85

List of Tables

Table 1. Empirical studies of comprehension strategies in the object-oriented paradigm.....	19
Table 2. Empirical studies of comprehension strategies in the procedural paradigm.....	20
Table 3. Overview of new tables created	30
Table 4. Description of roles.....	31
Table 5. Measures used in empirical studies.....	35
Table 6. Description of queries to extract data from GRUMPS	36
Table 7. Statistics of use of information sources during initial phase of Task 2	47
Table 8. Amount of time spent on information sources during the tasks.....	49
Table 9. Statistics of time spent on source code during experiment	52
Table 10. Statistics of number of classes accessed during experiment.....	52
Table 11. Description of documentation	54
Table 12. Statistics of use of documentation.....	54
Table 13. Statistics of compilation during experiment	55
Table 14. Statistics of execution during the experiment.....	56
Table 15. Statistics of quality of solution.....	57
Table 16. Proportion of information sources used in the initial phase of Task 2 - best and worst	57
Table 17. Statistics of use of information sources during for each task – best and worst	59
Table 18. Number of classes visited - best and worst.....	60
Table 19. Necessary alterations for task 2.....	81
Table 20. Necessary alterations for task 3.....	82
Table 21. Necessary alterations for task 4.....	83
Table 22. Classes in the User Interface package.....	85
Table 23. Classes in the Business Object package.....	85
Table 24. Classes in the Database package.....	86
Table 25. Classes in the Database package.....	86

List of Figures

Figure 1. (a) The repository schema (b) XML for a change of window focus event.....	29
Figure 2. Description of the data cleaning process	30
Figure 3. Architectural overview of the application packages and their dependencies	34
Figure 4. Example of a participant using all information sources (participant 4).....	38
Figure 5. Example of a participant using mainly source code (participant 10)	39
Figure 6. Example of a participant using mainly execution (participant 8).....	40
Figure 7. Example of total profile Task 2 (participant 15).....	43
Figure 8. Example of documentation profile Task 2 (participant 15).....	43
Figure 9. Example of compilation and execution profile Task 2 (participant 15)	44
Figure 10. Example of class profile Task 2 (participant 15).....	44
Figure 11. Proportion of information sources used by the participants during each task.....	49
Figure 12. Example of work practise Task 2 (participant 13).....	50
Figure 13. Example of work practise Task 3 (participant 13).....	51
Figure 14. Example of work practise Task 4 (participant 13).....	51

Figure 15. Example of class profile - many classes visited (participant 3, Task 2).....53
Figure 16. Example of class profile - few classes visited (participant 24, Task 2).....53

1 Introduction

1.1 Motivation

Software maintenance is widely recognised to account for a major part of a programmer's work and the key to successful maintenance is an adequate understanding of the program to be maintained (Koenemann and Robertson 1991; von Mayrhauser and Vans 1995; Corritore and Wiedenbeck 2001). Program comprehension is the process of understanding an unfamiliar program and is thus an important task of any maintenance whether it is corrective, reusing code or making enhancements to a program. The knowledge of professional programmers' comprehension-related activities during maintenance is hence useful for education in successful techniques that can result in more efficient programming and less faults. Furthermore it is useful for other researchers conducting studies of program comprehension.

The programmers' use of information sources has been explored by researchers in comprehension studies to analyse the program understanding strategies employed. The information sources mostly examined are source code and documentation (Koenemann and Robertson 1991; Corritore and Wiedenbeck 2001; Burkhardt, Détienne et al. 2002; Parkin 2004). O'Brien and Buckley (2005) argue that even the source code is an obvious information source, much information can also be obtained from the documentation and program execution. Singer et al.'s study and Seaman's survey (2002) of software maintainers work practises confirm that programmers rely on several sources of information about a system they are trying maintain. Although much research have been done, Corritore and Wiedenbeck (2001) explain that our understanding of the topic of program comprehension is still incomplete and there is a need for more knowledge about how the use of information sources changes over time.

Different methodologies have been used in empirical studies of program comprehension. The data collection methods most used are verbal protocols (Koenemann and Robertson 1991; von Mayrhauser and Vans 1997; Burkhardt, Détienne et al. 2002) and screen capture software (Corritore and Wiedenbeck 2001; Ko, Aung et al. 2005). Logging is rarely used (Parkin 2004). The majority of previous studies only presented the application in hard-copy (Pennington 1987; O'Brien, Buckley et al. 2004) while some allowed programmers to use tools and execute program (Corritore and Wiedenbeck 2001; Burkhardt, Détienne et al. 2002), but the extent to which these facilities assisted the comprehension process has not been assessed.

Limitations of previous studies were small size applications, source code only in hard-copy format and the lack of focus on other information sources but source code and documentation. The program used in our experiment was much larger than those used in previous experiments (Corritore and Wiedenbeck 2001; Burkhardt, Détienne et al. 2002; Parkin 2004). The JBuilder programming environment provided compilation and execution facilities and the participants had the possibility to search for information on the internet. In addition, all available documentation and source code was presented electronically at the same time. The participants could choose whether they wanted to familiarize themselves with the application or proceed directly to task-solving. They were not explicitly asked to understand the program and it was up to them what

information sources they wanted to use during the experiment. All activities performed by the participants were logged. This gives detailed information of their work practises during the study; the order, frequency and time spent on activities such as documentation, compilation, execution and source code. Written feedback was collected to supply the logged data.

1.2 Objective

The goal of this experiment was to examine the following:

1. The effects of expertise and strategies
2. Difficulties in object-oriented programming
3. Comprehension-related activities

The first topic is addressed by Kaja Kværn in her thesis (2006). Jørgen Busvold has examined the second topic in his thesis (2006). More details regarding these two topics can be found in their theses. The third topic is the objective of this thesis. This research explores the comprehension-related activities of professional programmers conducting several maintenance tasks on a medium-sized Java application. The collected data were analyzed to address the following research questions:

- How do professional programmers familiarize themselves with an unknown application in order to conduct maintenance tasks? What information sources are used to gain the initial program comprehension?
- What are the programmers' work practises during the different phases of maintenance? How does the use of information sources change over time?
- What working practises characterize the participants who perform extremely good or bad? How do they utilize tools and documentation during the experiment?

1.3 Research Method

We have performed a controlled software experiment with 24 participants. The participants were professional programmers from five different software companies. The experiment lasted six hours and the participants conducted three maintenance tasks on a 3600 LOC Java application by means of JBuilder 9 IDE. The participants' actions were logged by the Generic Remote Usage Measurement Production System (GRUMPS) and written feedback collected by the feedback-collection method. Post-experimental group interviews were also performed.

1.4 Research Context

This master thesis is part of the Comprehensive Object-Oriented Learning (COOL) project. COOL is an ongoing 3-year research project launched in 2002 by a consortium of four Norwegian institutions: InterMedia, Norwegian Computing Center, Simula Research Laboratory and Department of Informatics at the University of Oslo.

COOL aims at gaining insights into the complex area of learning and teaching object-oriented concepts and raising awareness of the problem areas in the communities of Computer Science Education and Computer-Supported Collaborative Learning. This aim is approached through a variety of designed experiments and examples, and through studies of existing practice. More details about COOL can be found on the project's web site (COOL).

1.5 Contribution

The main contributions of this research are the following:

1. The further development of GRUMPS
2. Detailed knowledge about the comprehension-related activities during the experiment: (1) use of information sources in the initial comprehension, (2) work practises and use of information sources during the maintenance tasks and (3) work practises of the best and the worst participants

We have made a further development of GRUMPS which includes data cleaning and analytical preparation of low-level usage data. The functionality added provides detailed information about each participant's use of compilation, execution and various documentations. The analysis tool gives a detailed picture of how the participants solved the tasks. It shows the chronological actions and time spent on source code, compilation, execution, web pages and documentation throughout the experiment. The written documentation and SQL-code can be reused by researchers in similar studies. The data preparation for analysis is a difficult process and the reuse of our work can simplify this process and save time.

The results regarding the initial strategy showed that in addition to source code and system documentation, the participants used execution to get an understanding of the system's dynamic functions. Identifying execution as an information source is useful for researchers conducting studies of program comprehension and can contribute to improve experimental design. The lack of facilities like compilation and execution in previous comprehension studies can cause threats to the validity of their findings. Our results show that the findings of these studies can not be trusted without reservation.

This research has extended the knowledge of professional programmers' comprehension-related activities during the phases of maintenance tasks. It is useful to identify programmers work practises and how the use of information sources change during an experiment. This knowledge is valuable for the development of more effective working methods and can be used as guidelines for novice programmers. The identification of information sources used for initial understanding and during the tasks, provide knowledge about information sources which should be available for programmers in both controlled experiment and ordinary work. The findings showed that system documentation, compilation, execution and the Java API was important information sources in addition to the source code. Compilation and execution were often performed during the study and served other purposes in addition to testing the modifications. Compilation was used to locate source code which had to be altered and execution was used to understand the application.

In addition to program modification, compilation and execution the participants working cycles also consisted of reading documentation. The Java API webpage was used when the participants needed more information about Java classes such as Calendar. The following findings confirm and extend previous studies and today's knowledge of program comprehension:

- Top-down comprehension in the initial phase (Corritore and Wiedenbeck 2001)
- An increasingly bottom-up approach throughout the tasks (Corritore and Wiedenbeck 2001)
- Mixed comprehension during the tasks (von Mayrhauser and Vans 1996)
- The scope was influenced by the phases of the experiment (Corritore and Wiedenbeck 2001)
- Different length of work cycles among the participants (Nanja and Cook 1987)
- Object-oriented programmers little use of dynamic guidance (Burkhardt, Detienne et al. 1998)
- The importance of source code as an information source (Singer, Lethbridge et al. 1997; Singer 1998; Seaman 2002)
- The importance of system documentation when learning an unknown application (Lethbridge, Singer et al. 2003)

Identifying the work practices of best participants can be useful, particularly for training programmers in successful techniques. The results show that the two best had a more successful employment of the programming environment, JBuilder. The main differences among the work practises of the two participants who performed the best and the two who performed the worst were that the best executed the application more, compiled less and accessed fewer classes.

1.6 Structure

Parts of this thesis have been written in collaboration with two other MSc students, Jørgen Busvold and Kaja Kværn. These parts are thus common for all three theses. This experiment is a replication of the experiment described by Levine (2005) and thus some sections have been taken from her thesis. The remainder of the thesis is organized as follows:

Chapter 2

Related work

This chapter presents an overview of related work. The identification of related work has been performed in collaboration with Kværn and Busvold. Section 2.2 gives an overview of the existing comprehension models and section 2.3 describe relevant empirical studies of program comprehension. These sections are quite similar in all three theses. The studies of programmers work practises in section 2.4 are addressed in this thesis only.

Chapter 3

Methodology

This chapter gives a detailed description of the controlled software experiment. The experiment is designed by Karahasanović (2005).

The description of the experiment, tasks and application are mainly based on Levine (2005). The further development of GRUMPS is described in section 3.3.1 and in Kværn (2006). All sections except 3.3.1 are common for all three theses.

Chapter 4

Analysis

Describes the analysis of comprehension-related activities collected by GRUMPS and the feedback-collection, including data preparation. The data preparation for the analysis, correctness and solution time are common for Kværn (2006) and this thesis.

Chapter 5

Results

This chapter presents and analyses the data collected in the experiment.

Chapter 6

Validity

The most important threats to the validity of the experiment are discussed in this chapter. Some of the threats mentioned can also be found in Kværn (2006), but they are especially related to the respectively researches.

Chapter 7

Conclusions and
future work

This chapter presents the conclusions and suggests future work

Appendix A

Task description

The appendix contains the maintenance tasks texts the participants performed in the controlled experiment and describes how the tasks can be solved in detail. The descriptions are written by Levine (2005) and are common for all theses.

Appendix B

The application

The application is described in this appendix. The description is written by Levine (2005) and is common for all theses.

2 Related Work

This chapter presents related work in the field of program comprehension. Section 2.1 describes the identification of related work. Section 2.2 presents the key models of program comprehension strategies. Section 2.3 gives an overview of empirical research of program comprehension. Section 2.4 presents research related to comprehension-related activities. Section 2.1 and 2.3 can also be found in Busvold (2006) and Kværn (2006) and are fairly equal for all three theses. Section 2.4 is only presented in this thesis.

2.1 Identification of Related Work

The identification of related work on program comprehension strategies have been done in collaboration with Kværn and Busvold. We performed the searches individually and merged the results. In order to find relevant work we have searched digital libraries and reference databases. The libraries we have searched are the ACM Digital Library, INSPEC, ICI Web of Knowledge and IEEE Explore. We also used the search engines Google and Google Scholar. In addition I have performed searches especially related to my research. The following keywords were used:

- Program comprehension
- Software comprehension
- Program comprehension strategies
- Object-oriented program comprehension
- Software maintenance
- Program comprehension documentation
- Data collection methods
- Work practises
- Information sources

Considerable research has been conducted on program comprehension and identifying relevant articles was not easy. The initial search resulted in more than 600 articles. Narrowing the search using various keywords resulted in 200 titles. I thereby filtered out the irrelevant articles based on the titles. Thereafter, the articles I found relevant were selected by reading the abstract. Relevant articles should focus on program comprehension and comprehension strategies, primarily maintenance task in the object-oriented paradigm. I also used references in relevant articles to find related articles. The last search was performed in March 2006.

2.2 Program Comprehension Strategies

To date there have been a number of studies investigating various aspects of program comprehension. Research has been conducted on the topic of general strategies, but also on software maintenance and enhancement. The topics that have been studied mostly are the direction of comprehension and the breadth of comprehension.

2.2.1 Direction of Comprehension

The direction of comprehension concerns the programmer's strategic approach to program comprehension. Existing cognitive models in program comprehension are classified as top-down, bottom-up or a combination of these two.

In the top-down model the comprehension process starts with the programmer forming a general hypothesis of the overall purpose of the program. The initially hypothesis is refined and elaborated based on information extracted from the code and other information sources (Brooks 1983).

The bottom-up model of program understanding assumes that the programmer build up a general understanding of the program by first reading code statements and then mentally group this information until a high-level understanding is reached. Pennington (1987) describes two program abstractions formed by the programmer during comprehension. These abstractions are the *program model* which is a low level abstraction and the *domain model* which is high level abstraction. The program model is formed before the domain model and is strong when code is new to the programmer. Pennington found that programmers who attained a high level of comprehension had a more dominant domain model and that construction of a domain model is essential for good comprehension.

Recent studies have suggested integrated comprehension models. Von Mayrhauser and Vans (1995) have proposed the Integrated Metamodel which builds on the previous models. The model has four major components: program model, situation model, top-down model and knowledge base. The first three describe the comprehension processes used to create mental representations of a program and the fourth provides support to them. The integrated model considers programmers to behave opportunistically in program comprehension, switching from top-down to bottom-up comprehension depending on task and situation. Von Mayrhauser and Vans (1997) observed programmers using the top-down approach when working in a familiar domain, and a bottom-up strategy in unfamiliar domain. In large programs the switches between the different models occurred frequently because the programmer's knowledge about the domain varied in different parts of the program.

Studies examining the direction of comprehension have not fully supported one strategy over another. Support has been found for top-down, bottom-up or a combination of these two, but the integrated comprehension models are increasingly viewed as a more realistic description of how programmers understand programs (Corritore and Wiedenbeck 2000; Corritore and Wiedenbeck 2001; Torchiano 2004).

2.2.2 Breadth of Comprehension

The breadth of comprehension refers to the programmers approach regarding the scope of comprehension. Littman et al. (1986) suggested two strategies; systematic and as-needed. The systematic strategy traces data flow through the program in order to understand the overall program behaviour. The programmer using the as-needed strategy focuses on local program behaviour in order to localize relevant code. Littman et al. (1986) observed that programmers who used the systematic strategy performed better than those who used the as-needed approach.

Their explanation was that the programmers who used the systematic strategy gathered knowledge about the causal interactions (interactions between components in the program when it is executed) while those using the as-needed approach failed to detect these interactions.

Koenemann and Robertson (1991) argued that the systematic approach is not realistic in larger programs. They carried out an experiment on a 600 LOC program, while Littman used a program of only 200 LOC. In the Koenemann and Robertson's experiment none of the subjects followed a systematic strategy of comprehension and were only interested in the parts relevant to the modification task.

Von Mayrhauser and Vans (1996) (1997) also present this view and argue that the systematic strategy is unrealistic for large programs, even though it seems better or safer. However they observed an incident of systematic study of a large program (von Mayrhauser and Vans 1994). Both Littman et al. and Von Mayrhauser et al. agree that a disadvantage to the as-needed approach is that understanding is incomplete and code modifications based on this understanding may be error prone.

2.3 Empirical Studies of Comprehension Strategies

Several experiments have been conducted in order to determine programmers' comprehension strategies, both in the procedural and object-oriented paradigm. Table 1 and 2 summarize studies regarding participants, applications (language and size), tasks (type of task and duration), environments (e.g. hard-copy or computer), data collection method and purpose of the study. The experiments which have been conducted in both paradigms are only listed once under the object-oriented paradigm (Table 1).

Table 1. Empirical studies of comprehension strategies in the object-oriented paradigm

Study	Participants	Application Task Environment	Data Collection Method	Purpose
Burkhardt et al. (1998)	49 professionals (28 experts, 21 novices)	C++ 550 LOC Comprehension for later documentation or reuse. 35 min. program study	Verbal protocols	Analyze object-oriented program comprehension and examine the effects of expertise in three dimensions of strategies
Burkhardt et al. (2002)	51 subjects (30 experts, 21 novices)	C++ 550 LOC Comprehension for later documentation or reuse. 35 min. program study (phase 1) Task performance (phase 2)	Verbal protocols Questionnaire	Evaluate the effect on program comprehension of three factors: programmer expertise, programming task and the development of understanding over time
Corritore and Wiedenbeck (2000) (2001)	30 professionals (15 OO experts, 15 procedural experts)	C, C++ 783 and 822 LOC Maintenance tasks	Screen capture software	Program understanding strategies employed during comprehension

		Two 3-hour sessions (7 to 10 days apart).		and maintenance activities carried out over time
Karahasanović et al. (2005)	39 students	Java 3600 LOC in 27 classes JBuilder IDE 3 maintenance tasks 6 hours	Logging Verbal protocols Feedback-collection Interview	Explore interactions between programmers comprehension strategy and difficulties in maintenance tasks
Ko, Aung et al. (2005)	10 experts	Java 503 LOC in 9 classes Eclipse IDE 5 maintenance tasks 70 minutes	Screen capture videos	Discover fundamental activities in maintenance work and use this understanding to elicit design requirements for new tools to support maintenance tasks.
Torchiano (2004)	28 students (4th year)	Java 628 LOC Maintenance Code and documentation available on-line	User action capture software	Non-intrusive approach to study comprehension cognitive models

Table 2. Empirical studies of comprehension strategies in the procedural paradigm

Study	Participants	Application Task Environment	Data Collection Method	Purpose
Koenemann and Robertson (1991)	12 professionals	Pascal 636 LOC 4 maintenance tasks 15 – 44 min spent on modification Documentation available	Verbal protocols	Scope of comprehension: systematic and as-needed
Littman et al. 1986 (1986)	10 professionals	Fortran 250 LOC Maintenance	Videotapes	Scope of comprehension: systematic and as-needed.
Mayrhauser and Vans (1994)	11 professionals	Maintenance	Verbal protocols	Find a code comprehension process model
Mayrhauser and Vans (1996)	11 professionals (detailed description of 1 of the subjects)	Pascal Large scale production code. Modules from ≤ 200 to ≥ 9000 LOC Maintenance task Two hours session	Verbal protocols	Find a code comprehension process model using the Integrated Comprehension model as a guide for large-scale program understanding.
Mayrhauser and Vans (1997)	4 professionals	Pascal Min 40 000 LOC Corrective maintenance tasks Observational field study Two hours session	Verbal protocols	Program comprehension behaviour during the debugging tasks of large scale software

Parkin (2004)	29 students (3 or more years into Computer Science studies)	C 281 LOC Maintenance (enhancement and correction) 1.5 hour	Logging Verbal protocols	Program comprehension strategies employed during maintenance tasks
---------------	--	---	-----------------------------	--

Purpose of study

All studies were in the area of program comprehension and examined how various conditions affected the strategies employed. The studies have examined the effect of expertise in the object-oriented paradigm (Burkhardt, Detienne et al. 1998; Burkhardt, Détienne et al. 2002) and the procedural paradigm (Koenemann and Robertson 1991), software maintenance and enhancements in both paradigms (Corritore and Wiedenbeck 2000; Corritore and Wiedenbeck 2001), object-oriented paradigm (Karahasanović, Levine et al. 2005; Ko, Aung et al. 2005) and procedural paradigm (Littman, Pinto et al. 1986; von Mayrhauser and Vans 1994; von Mayrhauser and Vans 1996; von Mayrhauser and Vans 1997; Parkin 2004), effects of phase (Corritore and Wiedenbeck 2000; Corritore and Wiedenbeck 2001; Burkhardt, Détienne et al. 2002), use of documentation by procedural programmers (Parkin 2004) and object-oriented programmers (Torchiano 2004), effects of programming paradigm (Corritore and Wiedenbeck 2000; Corritore and Wiedenbeck 2001), large scale code (von Mayrhauser and Vans 1994; von Mayrhauser and Vans 1996; von Mayrhauser and Vans 1997), tasks (von Mayrhauser and Vans 1997; Burkhardt, Détienne et al. 2002; Parkin 2004) and difficulties (Karahasanović, Levine et al. 2005).

Data Collection Method

The data collection methods used were verbal protocols, which were mostly used (Koenemann and Robertson 1991; von Mayrhauser and Vans 1994; von Mayrhauser and Vans 1996; von Mayrhauser and Vans 1997; Burkhardt, Detienne et al. 1998; Burkhardt, Détienne et al. 2002; Parkin 2004; Karahasanović, Levine et al. 2005), screen capture software (Littman, Pinto et al. 1986; Corritore and Wiedenbeck 2000; Corritore and Wiedenbeck 2001; Ko, Aung et al. 2005), logging (Parkin 2004; Karahasanović, Levine et al. 2005), questionnaire (Burkhardt, Détienne et al. 2002), interview and feedback-collection (Karahasanović, Levine et al. 2005).

Identification of Strategy

The comprehension strategies were identified by the proportion of documentation and code files accessed (Burkhardt, Detienne et al. 1998; Corritore and Wiedenbeck 2000; Corritore and Wiedenbeck 2001), percentage of program functions and lines of code accessed (Koenemann and Robertson 1991), times and proportion of files accessed (Parkin 2004), number of web pages visited and the time spent on each page (Torchiano 2004), verbal statements, the time spent on classes and the number of classes accessed (Karahasanović, Levine et al. 2005), verbal statements and the components studied (Littman, Pinto et al. 1986), correctness of responses to questions (Burkhardt, Détienne et al. 2002) and classification of verbal statements related to the integrated cognition model (von Mayrhauser and Vans 1994; von Mayrhauser and Vans 1996; von Mayrhauser and Vans 1997).

The direction of comprehension has been categorized in different ways. The top-down approach has been equated with the programmers accessing files at the most abstract level (documentation

and program header information) and bottom-up processing with the low-level implementation files (Burkhardt, Detienne et al. 1998; Corritore and Wiedenbeck 2000; Corritore and Wiedenbeck 2001; Parkin 2004). The top-down process has also been identified by assessment of global description (Koenemann and Robertson 1991) or by a small number of web pages visited and a long time spent on each page (Torchiano 2004). Torchiano equated the bottom-up approach by a large number of pages visited for a short time. Von Mayrhauser and Vans (1994; 1996; 1997) identified the direction of comprehension by classifying the actions types related to the domain (top-down), situation or program model.

Littman et al. (1986) categorized the systematic strategy as reading the code to understand how the program behaves before attempting to modify it and the as-needed strategy as minimizing studying the program and quickly localizing local parts of the program to modify. The systematic scope of comprehension has also been indicated by a broad study of all material while the as-needed was categorized by limitations of the study of program materials to only a small part of those available (Koenemann and Robertson 1991; Corritore and Wiedenbeck 2000; Corritore and Wiedenbeck 2001). In Karahasanović et al.'s categorization (2005) the systematic strategy was identified by the participants trying to get an overview of the application by use of provided documentation or executing the application before performing the tasks. The as-needed approach was defined by the participants going straight to the code to perform the tasks without trying to understand the system.

Results

Von Mayrhauser and Vans (1994; 1996) reported that code size affected the level of abstraction. Large size code increased the participants' work with higher level program details. They identified that programmers used a multilevel approach to understanding, frequently switching between program, situation and domain (top-down) models. This is supported by the results of Corritore and Wiedenbeck (2000) (2001) who found that the object-oriented participants initially used a top-down approach by focusing strongly on documentation, later their strategy shifted to a more bottom-up orientation. The procedural participants employed a more bottom-up strategy during the study. In Torchiano's study (2004) the result showed no clear separation among the top-down and the bottom-up behaviours. Torchiano argued that this agrees with the integrated comprehension model that predicts a switching between models.

Expertise in domain and programming language allowed more top-down comprehension, while little experience meant that comprehension was more bottom-up (von Mayrhauser and Vans 1997). Burkhardt et al. (1998) also found evidence of top-down behaviour in expert comprehension, but in the beginning the participants read files at both the top and bottom of the hierarchy showing both top-down and bottom-up direction.

The type of task also affected the comprehension. In adaptive maintenance the participants used more top-down comprehension than in corrective (von Mayrhauser and Vans 1997). The results from Parkins' study (2004) showed that programmers doing corrective tasks utilized documentation and header information and thus a more top-down approach than programmers undertaking an enhancement. Enhancers made more specific use of task documentation than corrective programmers and switched from initial top-down processing to bottom-up earlier.

In Littman et al.'s study (1986) all programmers acquired the static knowledge about the program, but only the programmers who used the systematic strategy acquired the necessary causal knowledge. The programmers using the systematic approach constructed more successful modifications than those using the as-needed approach. Karahasanović et al. (2005) also found that those who used the systematic strategy performed better. Koenemann and Robertson (1991) reported a very restricted scope of comprehension. In this study none of the participants used a systematic strategy and the authors argued that the systematic approach was unrealistic in large program. The results by Karahasanović et al. (2005) contradict this. They found that 58% of the object-oriented programmers applied the systematic strategy while maintaining an application of 3600 LOC. Corritore and Wiedenbeck (2000) (2001) reported that the procedural programmers employed a wider scope of comprehension than the object-oriented throughout the experiment. The scope of both groups was wider in the beginning and then narrowed.

2.4 Work Practises

Research into professional programmers' work practises includes both empirical studies and surveys. Programmers work practise include the activities they perform and the use of information sources during work.

2.4.1 Use of the Information Sources

Source code as programmers' main source of information is confirmed by several studies. Singer et al. (1997) explored the activities of a single engineer and a group of engineers maintaining a large system. In addition they considered company-wide tool statistics. Their data included the frequency of choosing particular tools and focused on what activities were the most common. Their shadowing studies showed that programmers spent a significant amount of time in just looking at the code. Source code was also reported as the most used information source in Seaman's survey (2002) of 45 software professionals use of information gathering strategies. In Singer's survey of software maintenance practises (1998) seven out of the ten companies claimed that the source code is the primary source of information used by programmers when carrying out enhancements to software systems. However, when solving a problem the programmers would only consult the source code in the incidents there the source code was known.

System documentation, if correct, complete and consistent, substantially facilitates the understanding and learning process of an unknown application (Kajko-Mattsson 2005). The importance of documentation is confirmed in a survey about how programmers use documentation (Lethbridge, Singer et al. 2003). More than 50% of the respondents found software documentation effective when learning or working with a new system. 50% found it effective when looking for an overview of the system and 35% found it useful when maintaining a system. The general attitudes to documentation were that architecture and other abstract documentation information were useful, but a considerable fraction of documentation was untrustworthy because it lacked maintenance. The importance of system documentation was also noticed by DeLine et al. (2005) in their study of professional programmers making changes to unfamiliar code. One of the biggest complaints noticed concerned the inadequate overview documentation about the system. This is in contrast to Koenemann et al.'s study (1991) which

reported that documentation was mainly seen as a last resort and only consulted, with the exception of flow-charts, when other methods of comprehension failed due to the participants' bad experiences with useless documentation. In Singer et al.'s study of work practises (1997) the authors discovered that there was a disagreement in the programmers' self reported activities and the activities carried out. 66% stated they spent most time reading documentation but the results showed that this was only true for 12 out of 356 events.

O'Brien and Buckley (2005) argue that in addition to source code much information can also be attained from other programmers, the documentation and program execution. Seaman's survey (2002) reported that human sources (programmers) were useful, especially when the code was unknown. Other information sources used were maintenance control systems, Integrated Development Environments (IDE), Computer-Aided Software Engineering (CASE) tools and test code. In addition she found that maintenance engineers frequently used execution traces. Those who regularly used execution traces claimed they were a most accurate and efficient way to understand a problem, especially when used in conjunction with the source code. The most reported activities in Singer et al.'s study (1998) in addition to source code and search, were execution trace and compilation.

2.4.2 Work Cycles

Ko et al. (2005) studied object-oriented experts programmers' maintenance work on a 501 LOC application in detail by recording the programmers work in full screen capture videos. They found that the programmers spent an average of 35% of their time navigating between dependencies, and an average of 46% inspecting task-irrelevant code. The authors suggested that the fundamental activities of maintenance work consisted of three activities; (1) forming a working set of task-relevant code fragments; (2) navigating the dependencies within this working set; and (3) repairing or creating the necessary code.

Nanja and Cook (1987) observed that the on-line debugging process performed by experts, intermediates and novices were different when debugging a small procedural application. Most participants started the session by studying the application, but the experts spent more time in their initial reading. Half of the novices and intermediates immediately ran the application without doing any initial reading. The programmers used the same series of steps repeatedly; (1) form bug hypothesis, (2) modify application to test the hypothesis and (3) execute the application to see the effects. The experts corrected groups of errors before verifying the corrections while the novices and intermediate corrected and verified single error. Novices and intermediates ran the application three and two times more than the experts. The authors suggested that expert's comprehension strategies resulted in more effective debugging performance than novices. In contrast, Ko and Uttil (2003) found that no comprehensions strategy was particularly successful, but individuals with stronger domain knowledge for specific bugs tended to succeed.

Tadaka et al. (1994) studied the debugging process in procedural applications (49 – 419 LOC). They reported that although the programmers' activity sequences appeared to vary considerably, the cycles of compilation, program execution and program modification were repeated in strict order. The frequencies of transitions were quite different from programmer to programmer. Some repeated the cycles very quickly while others spent longer time modifying. The increment of

program execution length also varied. Some participants recompiled programs without completely and correctly fixing faults. The faults appeared again when testing the program and such incomplete debugging resulted in an irregularly short cycle in the activity sequence.

2.5 Summary

A number of studies have been performed in the context of program comprehension, but according to the best of my knowledge, no studies have been conducted within program comprehension research to examine the extent to which the program execution assists the comprehension process. Furthermore, most experiments have only been conducted on small applications relative to industrial software. Programmers' maintenance of large application has merely been examined in observational field studies and surveys. Programmers' activities over the phases of the experiment has only been studied by Burkhardt et al. (2002) and Corritore and Wiedenbeck (2001), but they used different data collection methods.

In this study I intend to extend previous research by examining the participants work practises in detail; the order, frequency and time spent on documentation, source code, compilation and execution. Collection of feedback will be used together with the data from the log files.

3 Methodology

This chapter describes the research methods of the software engineering experiment. The main goal of this research was to identify the comprehension-related activities performed by professional programmers while understanding and performing maintenance tasks on a medium sized object-oriented program.

All sections except 3.3.1 are common for all three theses. The detailed description of GRUMPS is only presented in section 3.3.1 and in Kværn (2006). The description of the tasks is taken from Levine (2005).

3.1 Participants

The participants in this experiment were employees from five different software companies. The size of the companies ranged from small (5 employees) to medium size (over 100 employees). All companies developed software products for both the private as well as the governmental sector. All of the firms offered consulting services though some of them reported that their main focus was product development. The application background specialties ranged widely from Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), Systems Analysis and Program Development (SAP), advisory service, sale, and mobile technology to special development. The companies seem to be separated between focus on concrete methods like Rational Unified Process (RUP) on the one hand and more “loose” methods like Extreme Programming (XP) on the other. But there was also one company that claimed the method depended on their custom relationship.

All of the 24 participants were male, and the mean age was 31.9 range 25 to 50. They had taken between 0 to 80 credits (a full school year is 20 credits) in programming courses, the median being 30. Their programming experience ranged from 0 to 25 years and the median Java work experience was 2.35 years range 0 to 8. The participants self-reported they had produced between 1000 to 1000000 lines of Java code (median 40000). Almost all of the participants self-reported their skills in Java-programming as medium to good (median 4 on a five-point scale: 1 equals no experience with Java, 5 means expert). Their knowledge of JBuilder was self-reported as below medium (median 2 on a five-point scale: 1 equals no experience with JBuilder, 5 equals expert).

All of the participants were paid for attendance in the experiment.

3.2 Experimental Procedure

A pre-test with 2 MSc students were undertaken to ensure that the tasks, documentation and tools were appropriate.

The main study consisted of three phases. In the first phase, a week before the experiment, the participants filled in the background questionnaire answering questions on their education,

programming experience and experience with tools via a web-based tool. Usernames and password needed to answer questionnaires were distributed via e-mail. The second phase was the experiment. First, participants were welcomed, informed about the experiment's goals and procedure. They were also informed that all their actions were logged by a logging tool. Second, the participants were asked to sign a consent form. They agreed that they would not share information about the experiment with their colleagues and teachers, either during or after the experiment. The researchers guaranteed confidentiality and anonymity. After the introduction, the participants were given a unique username and password to be used during the day. Then, all participants were asked to solve a small training task and a pre-test calibration task. After that the participants were asked to conduct three maintenance tasks on the library application while writing feedback every 15 minutes on a screen. There was a 30 minute lunch break during the experiment and the participants were allowed to take shorter (few minutes) breaks when needed.

The experiment was conducted in five separate sessions on five separate days. The participants could choose the day they wanted to participate in the experiment. The number of participants per session was five on four days and four on one day. The participants were accommodated in a laboratory with one observer and one for technical assistance. Post-experimental group interviews were conducted by one interviewer and one observer.

3.3 Tools and Data Collection

A Web-based tool, the Simula Experiment Support Environment (SESE) (Arisholm, Sjøberg et al. 2002) was used for logistics support. The participants used SESE throughout the experiment to answer the background questionnaire, download documents and source code, upload their solutions and provide feedback. Start-time and end-time for each task were also recorded by SESE.

Keystrokes, mouse-clicks and window focus events were logged with timestamps in milliseconds by the GRUMPS-Lite software (Thomas, Kennedy et al. 2003). JBuilder 9 was used as the programming environment for the experiment.

3.3.1 GRUMPS Description

The Generic Remote Usage Measurement Production System (GRUMPS) was developed at Glasgow University (Evans, Atkinson et al. 2003). The goal of GRUMPS is to provide general purpose mechanisms for the capture of user actions for subsequent analysis and mining. In this experiment we used a reliable, low complexity version called GRUMPS-Lite. It includes a User Action Recorder (UAR) that runs under Windows. It can monitor all window activation, mouse and keyboard events, but can be restricted by the user or investigator when required.

There is a transport mechanism to harvest collected data and store it in the repository, a SQL Server database. The repository is designed for flexibility and is extremely simple. The database consists of three main tables as shown in Figure 1(a). Figure 1 is taken from Thomas and Mancy (2004). These tables are the basis of the data cleaning process. The XML fields store tagged data appropriate to the circumstances, such as shown in the Figure 1(b) for a Change Window Focus

event.

The repository design has proved to be very robust, and well adapted for rapid collection of large volumes of data (Thomas, Kennedy et al. 2003). Thus, collecting large volumes of low level data has become technically feasible.

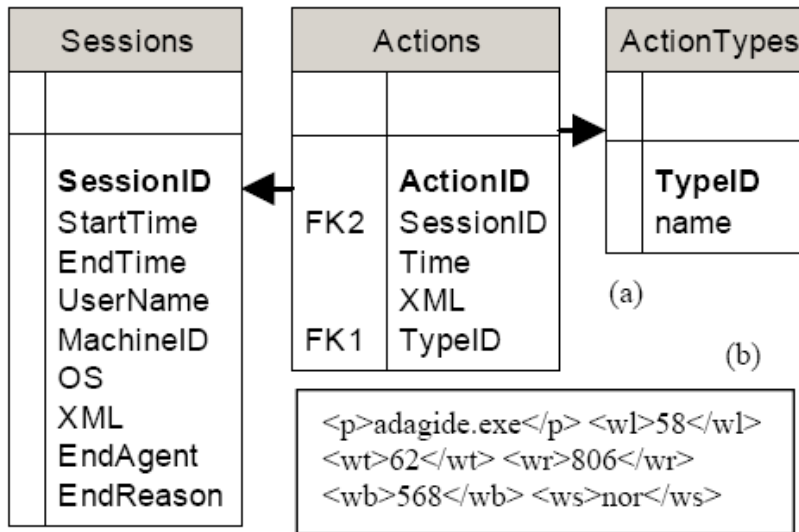


Figure 1. (a) The repository schema (b) XML for a change of window focus event

Difficulties experienced in previous studies

Several studies report that the data preparation for analysis of low-level data is an extremely difficult process. A number of researchers have noted that generic data collection often generates unwieldy and unmanageable data from which it is difficult to extract meaningful information (Misanchuk and Schwier 1992; Reeves and Hedberg 2003).

Renaud and Gray (2004) have analyzed data collected by GRUMPS. They report on a method of data cleaning and analytical preparation of low-level usage data which can be used in similar studies. In particular they discuss the special challenges confronted during a study based on low-level keystroke data. They point out that cleaning and meaningfully interpretation of low level data is not an easy task.

Thomas, Kenney et al. (2003) report from a study that investigated the use and usefulness of the GRUMPS software. They claim that the data preparation phase of the investigation is a large bottleneck and experienced that the data preparation took about a full person-month. This was due to the poor reusability of previous written queries

Our work with the database

The creation of the full database from the raw GRUMPS logs (Session, Actions, Actiontypes) was done by Richard Thomas, and the steps performed are described in a working draft written by him. We have further developed this database based on Thomas' documentation and previous SQL code written. Our goal was to get further knowledge of the participants' comprehension

related activities during the study. We especially wanted to examine their use of documentation and program execution behaviour. The database already contained two tables that held information on the participants' use of Java classes, Class and ClassActivity. We used a similar structure as in these tables. Our new tables are described in Table 3.

Table 3. Overview of new tables created

Table	Content
Doc	Documents and web pages.
DocumentActivity	Activities related to documents and web pages.
Category	Categorization of the activities
Library	Window titles related to compilation and execution of the application.
LibraryActivity	Activities related to compilation and execution of the application.

In order to create the tables described in Table 3 and extract the data for analysis, we went through a process that is illustrated in Figure 2. The steps performed by us are marked in grey.

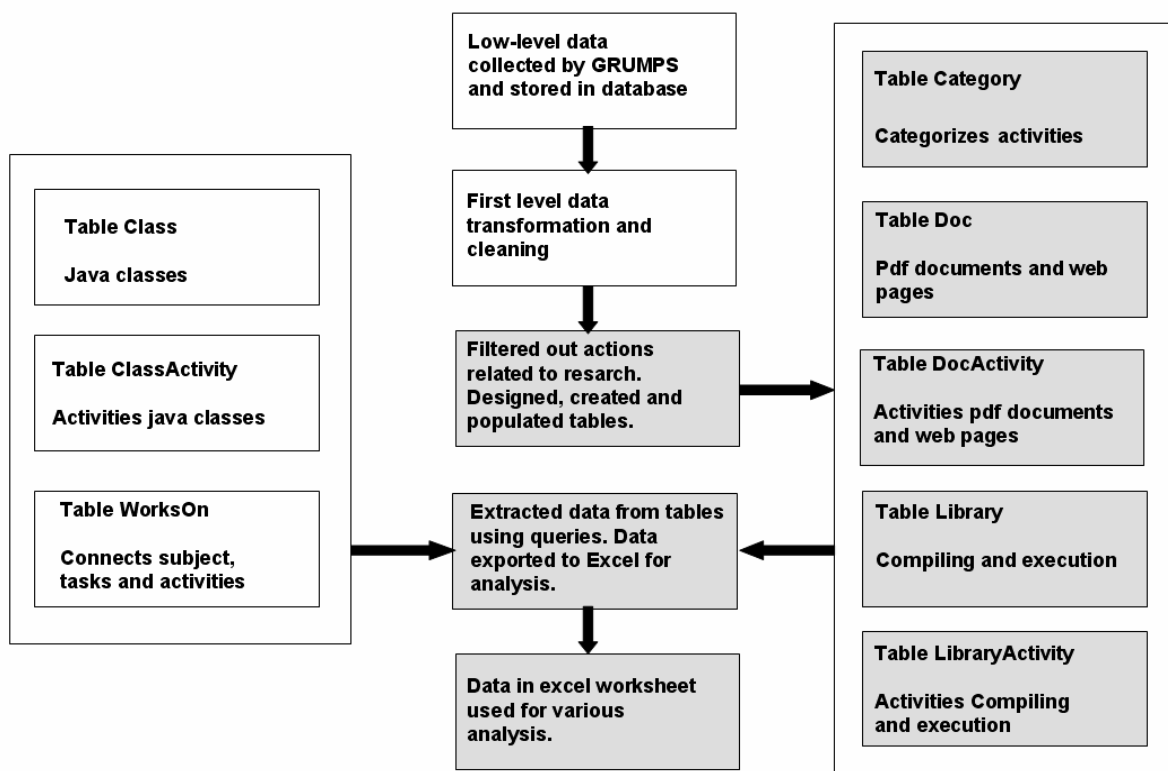


Figure 2. Description of the data cleaning process

We have written documentation that contains the SQL-code and the steps that must be performed to create and populate the tables. The following example describes the creation of documentation activities:

1. Create table Doc
2. Create table DocActivity
3. Insert values into Doc
4. Update fields in Doc
5. Create procedure DocActivitySituation(@sess numeric(18,0))
6. Execute the procedure for every valid session
7. Execute query for update workson_id in DocActivity
8. Execute query for update starttime and duration in DocActivity
9. Give permissions to the roles (grant select on 'tablename' to 'role')

By combining the data from the new tables with the data from the existing tables we achieved a detailed chronological overview of each participant’s use of source code, compilation, execution, web pages and documents during the experiment.

The database is used by researchers with different purposes. To protect the data we found it useful to create different roles which regulated the access to the database. The roles we created are Datacleaner, ResearcherComprehension and ResearcherGuest, see Table 4. The use of roles simplifies the job of giving new users access to the data.

Table 4. Description of roles

Role	Permissions
Datacleaner	Select: All Tables, Views Create: Tables, Views, Functions, Procedures Execute: Functions, Procedures
ResearcherComprehension	Select: All Tables, Views Create: Tables Execute: Functions, Procedures
ReseracherGuest	Select: Tables ActionsClean02, Application, ClassActivity, DocActivity, LibraryActivity, Subject, Task and WorksOn

The extraction of the data was not a trivial task. By modifying previous queries we managed to extract the data we needed for the analysis. The process of data retrieval and preparation for analysis was as follows:

1. Execute queries for ClassActivity, DocActivity and LibraryActivity for each participant and each task
2. The results were pasted into Microsoft Excel. One spreadsheet was made for each participant.
3. Excel’s Pivot function was used to transform the data into a more comprehensible format
4. Four graphs per task were made for each participant.

Thomas, Kennedy et al. (2003) experienced in their study that the data preparation took about a full person-month. It took us one full semester each for the total work. We spent about three

weeks just to comprehend the content and structure of the database. In addition we spent an amount of time understanding the procedures and queries that were crucial to our work. Much time was also spent validating the results. These challenges are described further in the next section. The tedious Excel work described above took several weeks to complete.

Lessons learned

There are several things we have learned. Firstly, we got a hands-on experience in the process of comprehending an unknown system written by others in order to perform maintenance tasks which is exactly what we have studied in our theses. We experienced the importance of having good documentation and source code available to understand the system before we performed the work. Secondly, we have learned the importance of documenting the work, decisions and difficulties that arose throughout the process. Thirdly, it is extremely important to verify the data. In fact, one of the greatest challenges was to get the data correct. We made several attempts before we finally succeeded. The challenges were especially related to:

- Catch all the actions when we grouped continuous actions. In addition we discovered that the summarising of duration was wrong when continuous actions were grouped. Finally we decided to get all single actions instead of grouping those which were in order. Then we got all the actions with correct durations.
- Getting the correct window titles. When we tried to use the table ActionsClean02 to get the window title, we discovered that the table contained wrong window titles for some actions. It was better to use the Actions table which contained correct titles.

During the work we experienced problems we had to take care of. We discovered that the documents had different titles depending on whether the participant had them maximized or not. This affected for instance the titles of the PDF-documents, and we had to find a solution to this problem. Another issue was that we found it necessary to categorize the activities in order to simplify the data preparation in Excel. We decided to create another table and alter the other tables by adding a new column. Thus several updates had to be executed. If we had realized this need at a previous stage, it would have been included in the creation of the tables and perhaps saved us some work. The lesson learned is that unpredicted events will occur and must be dealt with.

Generalization and Limitations

Our contribution can be applied to similar databases and can contribute to improve the understanding of developers' program comprehension strategies. However, researches must be aware of that the SQL code written is adjusted to our experiment. If other experiments use different tasks, IDE's or applications, the code has to be modified. The data preparation and meaningful interpretation of low-level data is a difficult process and we thus point out that our contribution is of great value because of the amount of time saved by reusing our code.

3.4 Tasks

The experiment consisted of totally four different tasks. The full task texts and descriptions of how the tasks can be solved can be found in Appendix A. First the participants went through a

training task to get familiar with the SESE-tool and experimental situation. The training task (Task 1) was a 400 LOC program (seven Java classes) which reproduced the functionality of an automated teller machine. The training task was to add a logging function to this application and was 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 and reserve a title if it is not available. The participants were asked to conduct the following changes to the library application system:

Task 2	Delete functionality related to ISBN number
Task 3	Extend the system to handle customers e-mail address
Task 4	Introduce the functionality to inform a person when a loan is due

The tasks were ordered by complexity, but Task 2 and Task 3 were about the same level. Task 2 was a corrective task while Task 3 and Task 4 were enhancement tasks.

3.5 The Application

The application consisted of four packages with totally 3600 LOC in 27 Java classes. The main class, StartClass.java, was not included in any package. The application is described in detail in Appendix B.

3.5.1 Architectural Overview

The architecture is a traditional three-tier architecture where user interface classes access data and behaviour from the persistent business objects, which store and retrieve themselves using services from the database layer. Figure 3 shows an architectural overview of the application.

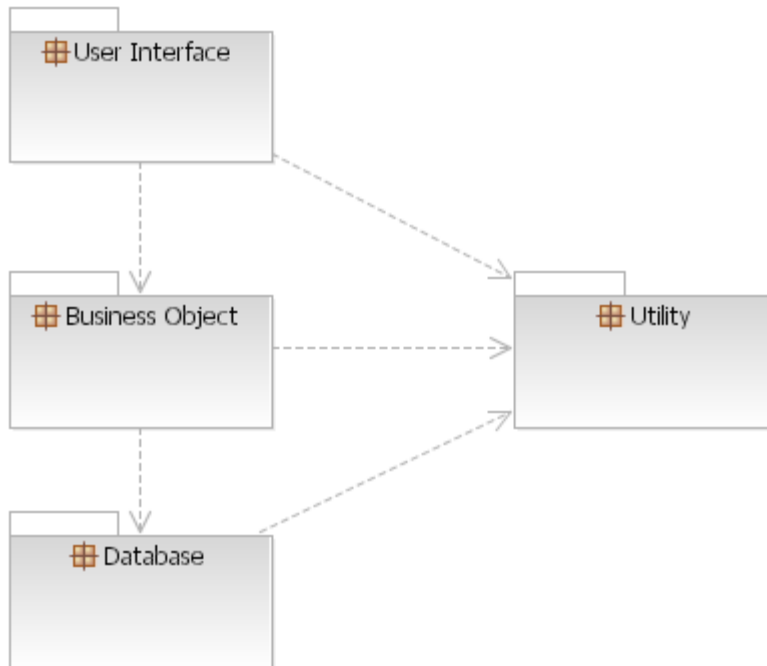


Figure 3. Architectural overview of the application packages and their dependencies

3.6 Documentation

The participants were provided with a document describing the library application system (Library.pdf) and a short introduction to the Java programming language (JavaDoc.pdf). They also had access to the Java online documentation (Java API) and the possibility to search for information on the internet. The system documentation contained a short introduction to the application, detailed UML-diagrams with description of the objects and interactions and a user manual including screen dumps of the system.

The task documentations given to the participants included test examples and screen dumps of the correct solution. The documentation of Task 4 also contained additional description of Java's Calendar class which was relevant for solving the task.

4 Analysis

This chapter describes the analysis of data collected by GRUMPS and the feedback-collection. Related work concerning measurement and identification of comprehension-related activities is described in section 4.1. Section 4.2 presents the analysis model. The data preparation for analysis, correctness and solution time are common for this and Kværn's thesis (2006). The identification of initial strategy is described in section 4.3. Section 4.4 describes the identification of work practices during the tasks.

4.1 Measures Used in Previous Studies

Table 5 summarises measures used in previous studies regarding what they studied as well as, what and how it was measured. The table includes studies of program-comprehension, programmers work practises and the debugging process.

Table 5. Measures used in empirical studies

Study	Purpose	What measured	Data Collection Method
Burkhardt et al. (1998)	Breadth and direction of comprehension	Proportion of files accessed	Verbal protocols
Corritore and Wiedenbeck (2001)	Breadth and direction of comprehension	Proportion of files accessed	Screen capture software
Parkin (2004)	Breadth and direction of comprehension	Proportion and duration of files accessed	Verbal protocols Logging
Singer et al. (1997)	Work practises	Number of activities	Observation Shadowing Interviews Tool usage statistics
Ko et al.(2005)	Activities in maintenance work	Number and duration of events	Screen capture videos
Nanja and Cook (1987)	Debugging process	Duration and number of activities	Recording Verbal protocols
Takada et al (1994)	Debugging process	Duration, number and order of activities	Monitoring

The studies have analysed the participants' use of different information sources. Studies of program comprehension used proportion of files accessed to analyse the breadth and direction of comprehension et al (Burkhardt, Detienne et al. 1998; Corritore and Wiedenbeck 2001; Parkin 2004). The files analysed were documentation and source files. Studies examining work practises analysed the number of activities performed by the programmers (Singer, Lethbridge et al. 1997; Ko, Aung et al. 2005). In addition to source code and documentation, they also examined activities like search, compilation and execution. In the studies of the debugging process they analysed the participants' activities regarding program modification, compilation and execution (Nanja and Cook 1987; Takada, Matsumoto et al. 1994).

4.2 Analysis model

This section describes the different elements used in the analysis.

4.2.1 Data Preparation for Analysis

In this study, the data preparation for analysis was as follows: First the data regarding each participant were extracted from the GRUMPS database and exported to Excel. The queries and data are listed in Table 6. Then the data were checked against user actions logs produced by the GRUMPS. We identified and removed all lunch breaks or breaks longer than 10 minutes from the extracted data. After the data cleaning Excel's Pivot-function was used to transform the data into a comprehensible format. Examples of the graphs are given in section 4.3.1 and 4.4.3.

Table 6. Description of queries to extract data from GRUMPS

Query	Data
queryClassAnalyzeMinutes	Classes visited and time spent in each class, ordered by minutes into the tasks.
queryDocAnalyzeMinutes	Documents and web pages visited and time spent in each document, ordered by minutes into the tasks.
queryLibraryAnalyzeMinutes	Start time and duration of compilation and execution of the library application, ordered by minutes into the tasks.

Total profile

The total profile is a graph which gives a high-level picture of the participants' actions during each task. The actions are categorized into "Source code", "Documentation", "Compilation" and "Execution". Each of the categories is detailed further in the class profile, documentation profile and the compilation and execution profile

Class profile

The class profile is a graph that presents an overview of the classes visited by a participant and time spent in each class, ordered chronologically.

Documentation profile

This is a graph that shows what kind of documentation a participant has accessed during the task. The graph also shows how long the participant has been reading each document

Compilation and execution profile

This profile is a graph that gives a detailed description of how a participant has compiled and executed the library application. It shows which windows in the application the participants have visited at and for how long.

4.2.2 Correctness

This is the assessment of the quality of the participant's solution. Marking was on the basis of correctness and the solutions were marked as correct (1) for fully functional or not correct (0) for those with major defects. In addition the solutions were ranged from score 5 for a perfect solution to score 0 for a non-attempt. The solutions with errors were given from score 4 to score 1, dependent on the type of error.

The assessment of correctness was given by an independent consultant from another research institute and was not involved in the experiment. The consultant was provided with task specifications, correct solutions and guidelines for giving scores.

4.2.3 Solution Time

This is the time taken, in minutes, to complete (understand, code and test) the tasks. It was calculated as $\text{end_time} - \text{start_time}$, where start_time is the time when a participant downloaded the task description and end_time is the time when the participant uploaded his solution, as recorded by the SESE tool.

Lunch breaks or other breaks longer than 10 minutes were subtracted from task times. Thus, the variable time is the time participants spent on the maintenance tasks excluding major non-productive breaks.

4.3 Initial Strategy

For the purpose of this analysis I have defined the participants' actions during the first third of Task 2 to represent their initial strategy for familiarizing themselves with the unknown application. The participants were introduced to the application at the same time as the task and the documentation. It was up to them what information source they wanted to use. When examining the participants' information preferences I found that they used source code, execution, compilation, task documentation and system documentation in the initial phase. In this analysis the information sources are thus divided into the following categories:

- Execution
- Compilation
- Task documentation (Task 2)
- Source code (classes)
- System documentation

To identify the information sources used in the initial phase I first analysed the data collected by the GRUMPS. I looked at both the time spent on the different information sources and the information preferences of each participant. I also looked at the feedback-collection written by the participants to enhance and detail the data collected by GRUMPS.

4.3.1 Data from GRUMPS

The data collected by the GRUMPS were transformed into three levels in Excel:

1. The raw data collected by the queries
2. Pivot tables
3. Graphs

I extracted the raw data from the initial phase for each participant and transformed the data into pivot tables and total-activity graphs. The pivot tables and total-activity graphs show the use of information categories as described above. The classes defined as “not of interest” (classes used in the training task) and the .jpx file (the project file containing the source code) were removed from the raw data because I wanted to examine what information the participants used to understand the application.

The total activity profiles for the initial phase were used to identify the participants’ use of information sources. In addition to the graphs I used the data in the pivot tables to get the exact time the participants spent on the different activities and to identify the program compilation. The graphs were too coarse for these purposes.

Examples of initial strategies identified by GRUMPS

Figure 4 to 6 give examples of total activity graphs for the initial phase. The first figure shows a participant who used all information sources. He mainly read the task description and system documentation only interrupted by short periods of execution. The source code was first visited after the fourteen minutes.

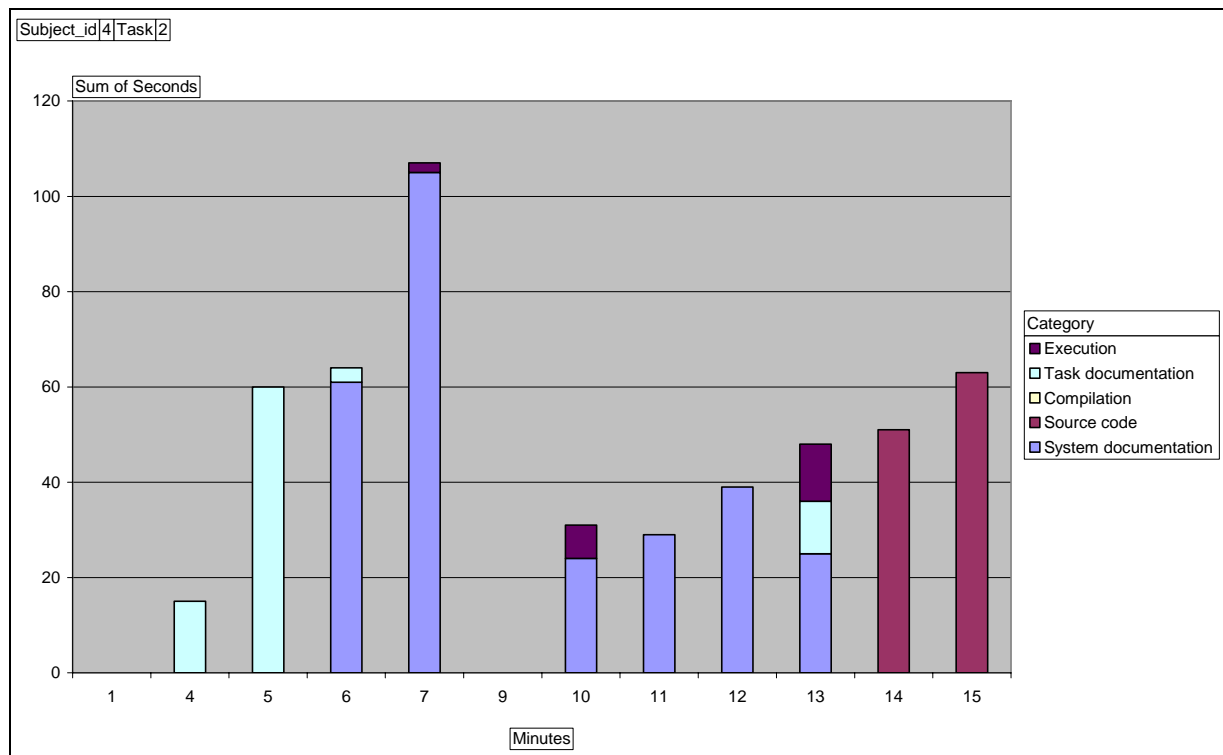


Figure 4. Example of a participant using all information sources (participant 4)

In Figure 5 we see a participant who spent most of his time accessing the source code. He read the task documentation in fourth minute and used mainly source code after that. He executed the application for 4 seconds in the seventh minute and did not read any system documentation.

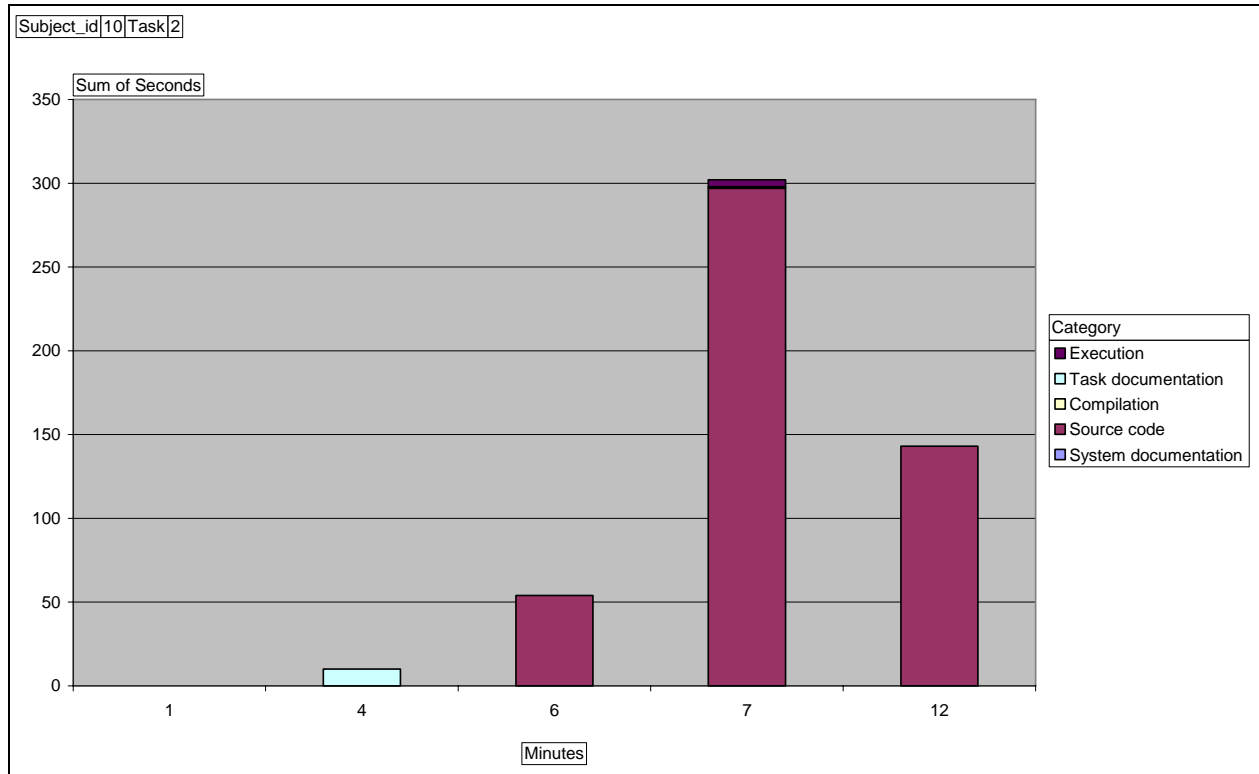


Figure 5. Example of a participant using mainly source code (participant 10)

Figure 6 shows a participant who spent most time executing the library application. First he visited source code for 2 seconds before he compiled and executed the application. Then he alternated between accessing the source code and executing the application until the sixteenth minute. The system documentation was accessed for 5 seconds in the tenth minute and the task documentation was first read in the seventeenth minute.

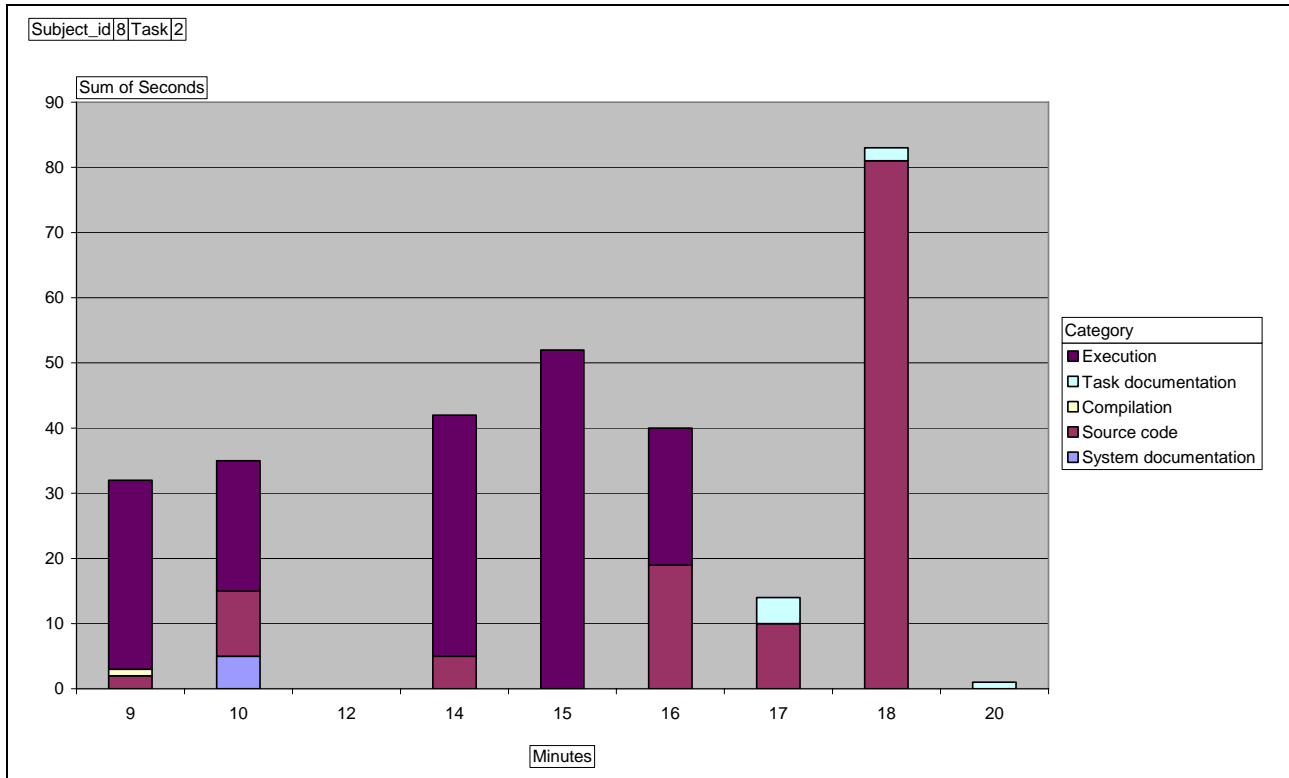


Figure 6. Example of a participant using mainly execution (participant 8)

4.3.2 Data from Feedback-Collection

The data from the feedback-collection was used to get a fuller picture of the participants' actions during the initial phase. While the data from the GRUMPS can tell us what the participants did, it says little of why the participants chose to use the different information sources. I therefore examined the feedback-collections for each participant to explain and validate the data obtained by GRUMPS. I was especially interested in the feedback related to the experimental conduct which gave information about the way in which the participants carried out the initial phase.

4.4 Work Practises during the Tasks

The participants work practises are the activities performed during the study. I have analysed the participants work practises in order to determine the comprehension strategies and to identify the use of information sources during the maintenance tasks. In our experiment all available material and documentation were presented electronically and the participants' actions were logged by GRUMPS. Detailed analysis of the information sources used during the experiment and the order in which the sources were accessed can thus be conducted. I used the data collected by GRUMPS and the feedback-collection to analyse the work practises.

4.4.1 Direction of Comprehension

The term “direction of comprehension” indicates whether the approach strategy is top-down, bottom-up or a mixture of both. In studies of object-oriented programmers using C++ (Burkhardt, Detienne et al. 1998; Corritore and Wiedenbeck 2001) the direction has been determined by the programmers accessing three levels of abstraction corresponding to three levels of file types; the abstract documentation files, the intermediate abstract class header files and the low-level implementation files. C++ has two types of files, .h files containing the declarations of data elements and functions and .cc files containing the implementation of the functions. In contrast to C++, Java has only one file type and has no division of header files and implementation files. All Java files are therefore characterized as low-level files. It can be discussed whether the packages containing the source code are at different levels of abstraction, and especially the business objects package may be characterized as more abstract than the other packages. The user interface classes access data and behaviour *from* the persistent business objects, which store and retrieve themselves using services from the database layer. When solving the tasks, changes are required to classes in the business package as well as the user interface package. That means that the business package has to be accessed in any case and I have therefore only considered the Java classes in this thesis. The definition of the directions of comprehension is somewhat different from other definitions because it has only two levels of abstraction:

- The abstract level (top-down) is equated with information sources which give an overview of the application; the system documentation and executing the library application in the initial phase of Task 2.
- The low-level (bottom-up) is represented by the Java classes.

To analyse the direction of comprehension I have analysed the data collected by GRUMPS. I have examined the duration, order and proportion of information sources categorized as documentation, source code, program compilation and execution during the tasks.

4.4.2 Detailed Use of Information Sources

Source code

The scope of comprehension refers to the extent of which the developers become familiar with most parts of the system during comprehension. In studies of object-oriented programmers using C++ (Burkhardt, Detienne et al. 1998; Corritore and Wiedenbeck 2001) the scope of comprehension were operationalized as the proportion of files accessed. In this study the evolution of scope is examined by the proportion of source code accessed throughout the tasks. I also looked at the time spent on the source code.

Documentation

The participants decided themselves what documents they wanted to use when familiarizing themselves with the new program and solving the tasks. I have analysed the documentation preferences during the experiment and examined what documentation the participants considered useful in the different tasks. I examined the time spent on the different documentation during the tasks and the number of participants who used them.

Compilation and execution

Program compilation and execution are facilities provided by the programming environment, JBuilder. To examine the necessity of these facilities I analysed the participants' use of compilation and execution. I looked at the time spent on execution and how many times they compiled during the tasks. The written feedback-collections were used to identify other purposes in addition to testing.

4.4.3 Data from GRUMPS

As described in section 4.3.1 the data collected by GRUMPS were transformed into three levels in Excel. The raw data were used to identify the number of compilations done by the participants. I used the pivot tables to identify and collect the time spent on the different information sources. The pivot tables were also used to identify the times spent on the different documentation.

The total profile graphs were used to identify the participants' overall use of information sources during the study. When I needed a more detailed picture of the participants' use of each information source I used the graphs showing the class profile, the documentation profile and the compilation and execution profile.

Examples of graphs

Examples of each of the graphs are given in Figure 7 to 10. The examples illustrate how the total profile is detailed by using the graphs for each information source. Figure 7 shows an overview of a participant's actions during Task 2. We see that he first looked at documentation and then accessed source code before he compiled and executed the application. Then he alternated between executing the application and documentation the first six minutes. From the seventh minute and through the rest of the task he mainly spent time on source code interrupted by short sequences of execution. In Figure 8 we see that the documentation he read during the first three minutes was the system documentation. Then he read the task documentation from the fourth minute and during the rest of the task. We see exactly when and for how long he read the task documentation. Figure 9 shows how he executed the application during the task. We see what windows he opened, the order in which they were opened and the time spent on each window. The detailed use of the source code is illustrated in Figure 10. The first class accessed was MiniBank.java which is a part of the training task. Then he opened the project file (.jpx) and started to access the classes in execution order; first the main class, StartClass.java, and then the MainWindow.java. In addition to the MiniBank.java and the project file, the participant accessed 8 classes during the task. He visited all the classes which needed changes in Task 2 and spent most time on them and the class MainWindow.java.

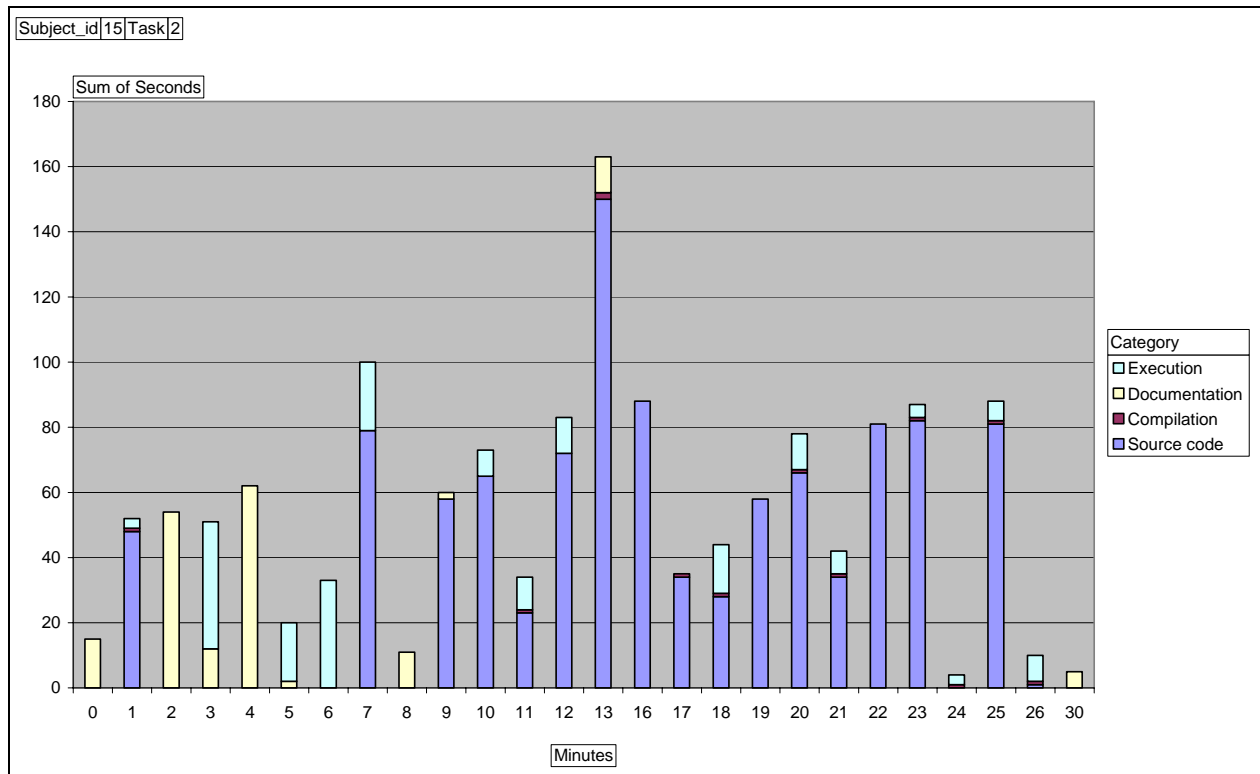


Figure 7. Example of total profile Task 2 (participant 15)

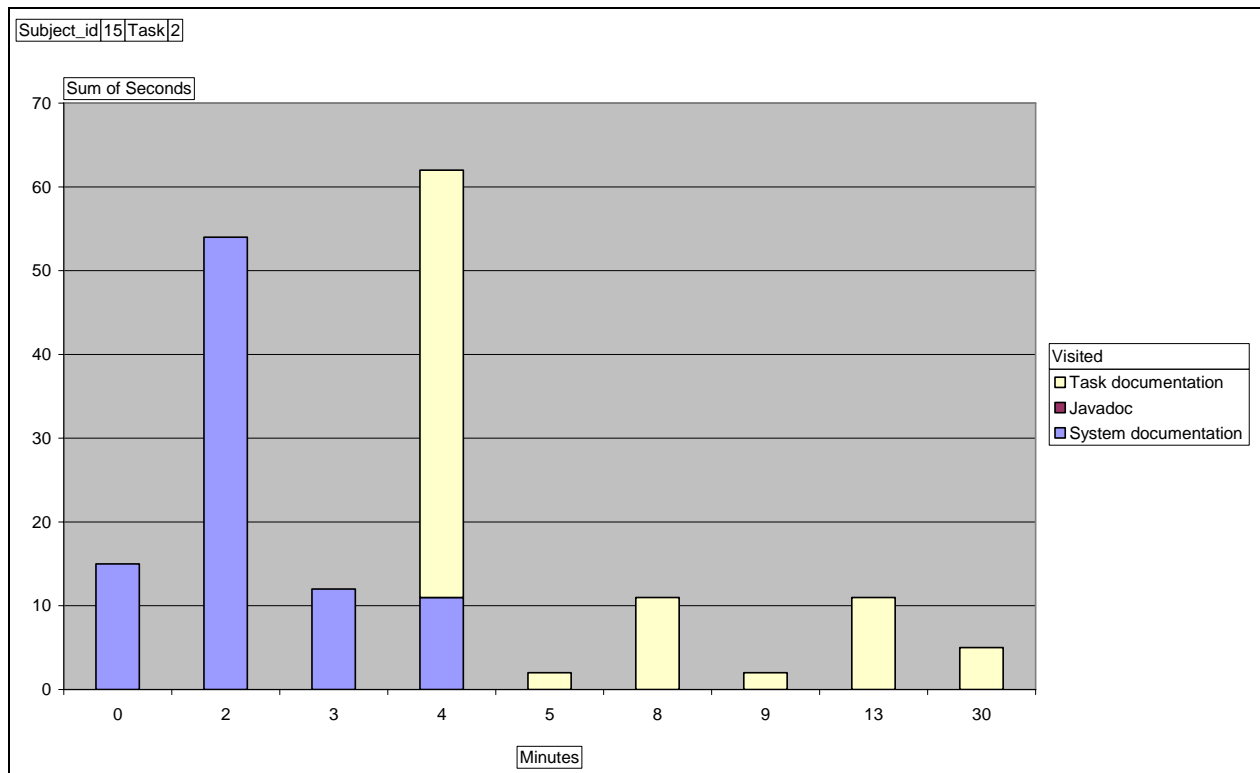


Figure 8. Example of documentation profile Task 2 (participant 15)

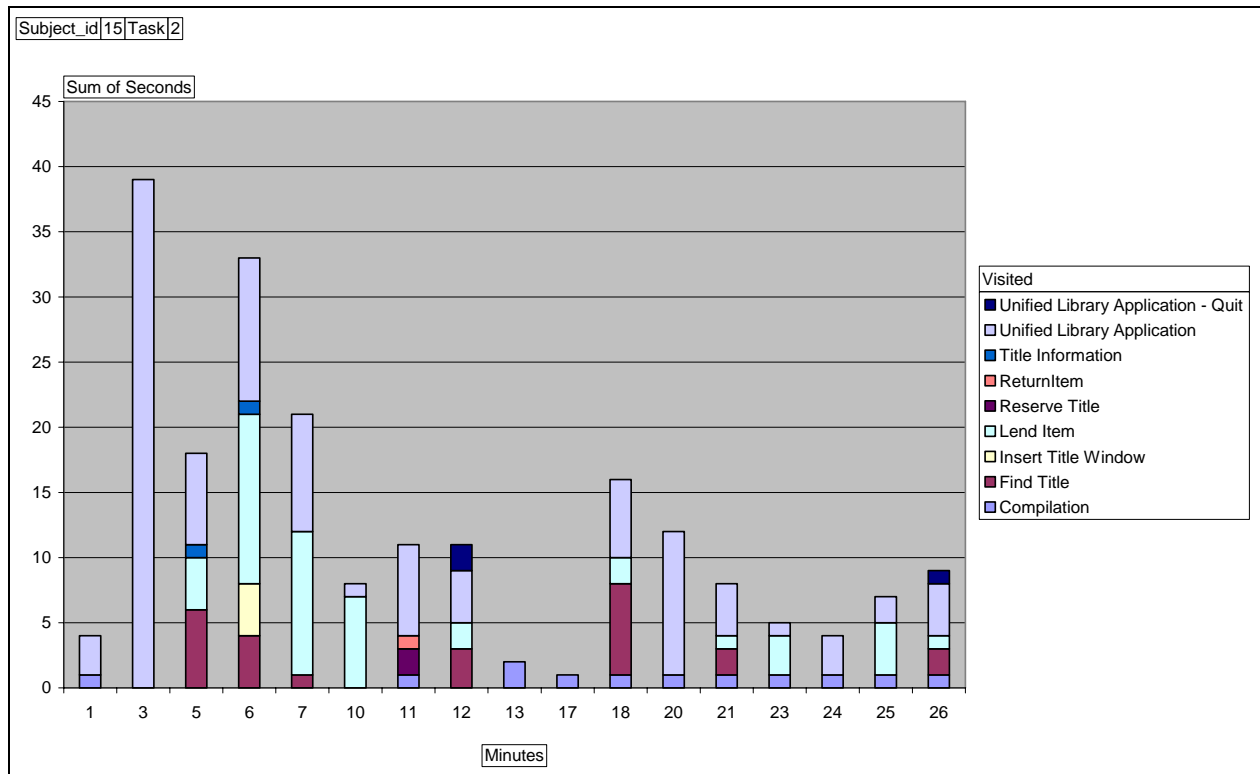


Figure 9. Example of compilation and execution profile Task 2 (participant 15)

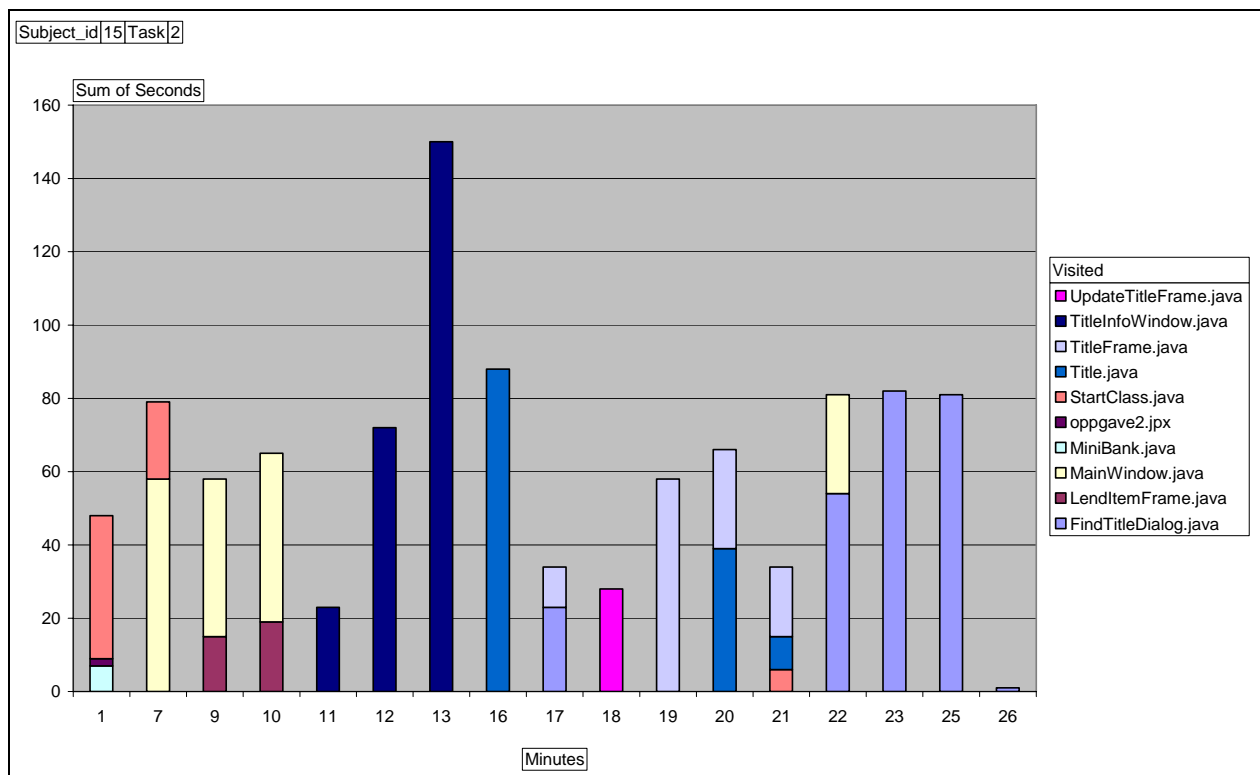


Figure 10. Example of class profile Task 2 (participant 15)

4.4.4 Data from Feedback-Collection

The data from the feedback-collection were used to enhance the data collected by GRUMPS. I examined the feedback-collections for each participant to explain and validate the data obtained by GRUMPS. I especially searched for statements related to the use of program compilation and execution.

5 Results

This chapter presents and analyses the data collected in the experiment. Section 5.1 – 5.4 presents the results. Section 5.5 summarizes and discusses the findings. Some sections present examples written on the feedback-collection screen. The comments were originally written in Norwegian and have been translated by myself for this thesis.

5.1 Initial Strategy

The participants' actions during the first third of Task 2 describe their initial strategy when they came to learn the unknown application. The solution times differed from 15 to 60 minutes and the time spent on the initial phase is thus different for each participant. Statistics of the participants' use of information sources are presented in Table 7. The proportion out of total time shows each information source's share of the total time used.

Table 7. Statistics of use of information sources during initial phase of Task 2

	Time (seconds)				Proportion out of total time
	Mean	Median	Min	Max	
System documentation	110	79	0	499	25.6%
Source code	196	157	0	614	46.2%
Task documentation	60	62	0	137	14%
Execution	58	35	0	159	13.6%

Source code was most used (46.2%) and ranged from 0 – 614 seconds (median 157). The system documentation was the second most used information source (25.6%) and ranged from 0 – 499 seconds (median 79). 58.3% of the participants spent more than 1 minute on the system documentation. The time spent on the task documentation and execution was quite equal, the task documentation ranged from 0 – 137 (median 62) and execution ranged from 0 – 159 (median 35). In addition one participant spent 111 seconds on the Java documentation and one spent 2 seconds on the Java API.

The number of participants who used the different information sources is listed here. The proportions of participants are in parentheses.

System documentation	20 (83%)
Source code	22 (92%)
Compilation	24 (100%)
Task documentation	23 (96%)
Execution	23 (96%)

There were only five participants who didn't access the system documentation. Four of them spent most time at source code and one at the task documentation. One of the two who didn't access the source code spent most time at the system documentation while the other executed the application. The participant who didn't read the task description spent most time executing the

application. The one who didn't execute the application spent most time on classes.

The results showed that the participants used source code, system documentation and execution to understand the unfamiliar application. There proved to be a huge difference in what information source the participants emphasised during their initial comprehension of the application. 50% of the participants spent most time on source code, 25% used mainly the system documentation and 12.5% spent most time executing the application. The participants used a combination of minimum 3 to 5 information sources and 66.7% used all of them.

Excerpts from the feedback-collection confirm that they used different information sources to familiarize themselves with the program. The participants expressed the view that executing the application and reading the system documentation together gave an overview of the system. The total profiles for the initial phase showed that 66.67% of the participants executed the application *before* they spent a major time on the source code.

"I'm reading the task documentation. And I want to execute the application and read the documentation to get an overview. Read the source code to get an overview"
[Participant 10]

"I'm trying to understand the application. I have registered some books and a borrower. Now I'm reading the description of the system, but not as thorough as I first planned. I'm mainly getting an overview. I want to start at the task pretty fast, but want to get an overview first." [Participant 12]

"I have started to read the documentation and the task documentation. I am not very familiar with class diagrams, so I must look at them. Now I'm thinking at executing the application to get a better foundation for further reading in the documentation"
[Participant 22]

5.2 Work Practises during All Tasks

In order to make the overview of the participants' activities more comprehensible, they were categorized into the following:

Documentation	All documentation used; task documentation, system documentation , Java documentation and web-pages visited
Source code	All classes
Execution	All compilation and execution

Table 8 summarises the total amount of time spent on source code, documentation and execution during the three tasks.

Table 8. Amount of time spent on information sources during the tasks

	Time (in minutes)		
	Task 2	Task 3	Task 4
Source code	464	791	862
Documentation	96	81	118
Execution	80	122	76
Total time for each task	640	994	1056

The participants spent distinctly less time on Task 2 than on the two other tasks. The total time used ranged from 640 minutes in Task 2 to 994 in Task 3 and 1056 minutes in Task 4. To compare the use of information sources during each task I have thus calculated the proportion of time spent on source code, documentation and execution out of the total time spent on information sources for each task. The proportion of information sources accessed is shown in Figure 11. The initial phase of Task 2 is presented separately because this phase shows the participants use of information sources when the application was unfamiliar.

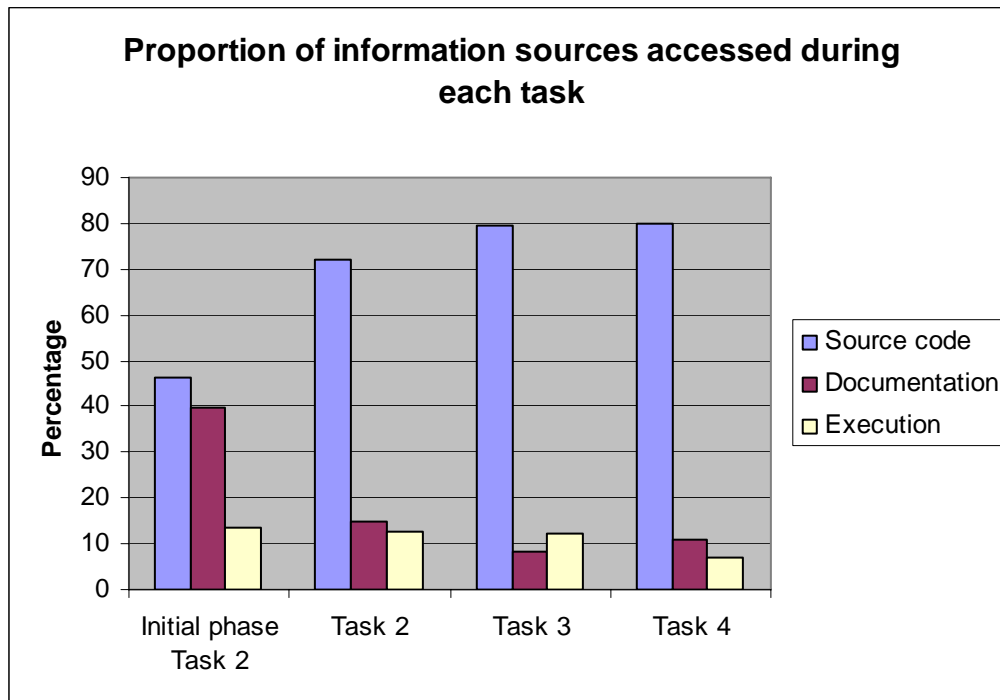


Figure 11. Proportion of information sources used by the participants during each task

Figure 11 shows that the source code was the participants' main source of information during all tasks. They spent distinctly more time on source code than reading documentation or executing the application. This is of course due to the maintenance tasks which required modification of the source code. Documentation was slightly more used than execution.

The use of information sources varied during the tasks and the difference between the initial phase of Task 2 and the rest of the tasks are considerable. The proportion of documentation use was much higher during this phase and the use of source code was lower. The use of

documentation was 40% in the initial phase versus 15% for the total Task 2, 8% in Task 3 and 11% in Task 4. The proportions of classes were 46% in the initial phase versus 72% for the total Task 2, 79.5% in Task 3 and 79.8% in Task 4. The use of execution ranged from 13.6% in the initial phase of Task 2 to 7% in Task 4. It was about the same for the total of Task 2 and Task 3 (12.5% and 12.3%).

5.2.1 Work Cycles

By examining the total profile graphs, I saw that most participants used the initial phase of Task 2 to understand the system through reading documentation, source code and executing the application. Thereafter they alternated mainly between reading/editing source code and execution. In the end of Task 2 they alternated between documentation and execution. In Task 3 they spent less time on documentation in the beginning, but alternated frequently between source code, documentation and execution. This may be due to that Task 3 consisted of three subtasks. The work cycles of source code, documentation and execution were the same in Task 4 even though it consisted of only one task. Figure 12 to 14 present a typical example of a participant's work practise during the tasks.

Counting the alternations between documentation, source code and execution showed that the participants' work cycles were different. An alternation was counted every time a switch to a new activity occurred. The alternations ranged from 14 to 55 times (median 24.5) in Task 2, from 13 to 88 times (median 52.5) in Task 3 and from 1 to 84 times (median 50.5) in Task 4.

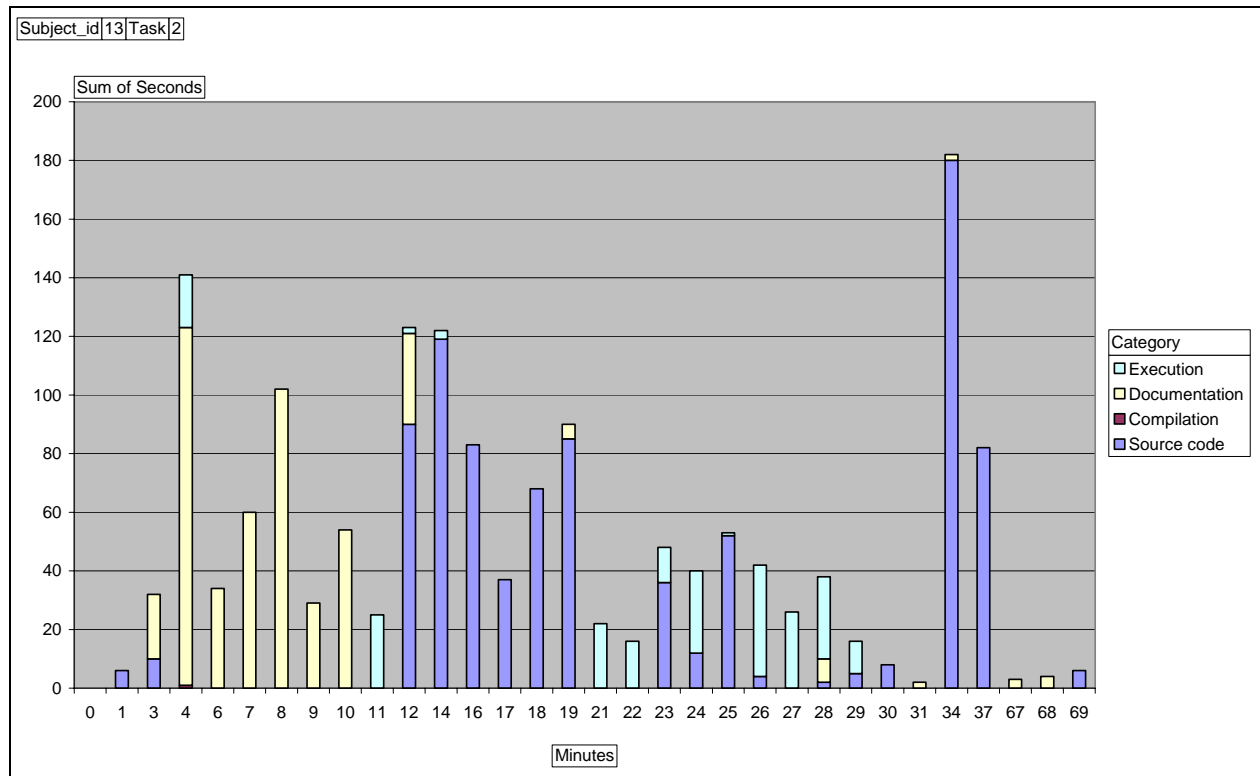


Figure 12. Example of work practise Task 2 (participant 13)

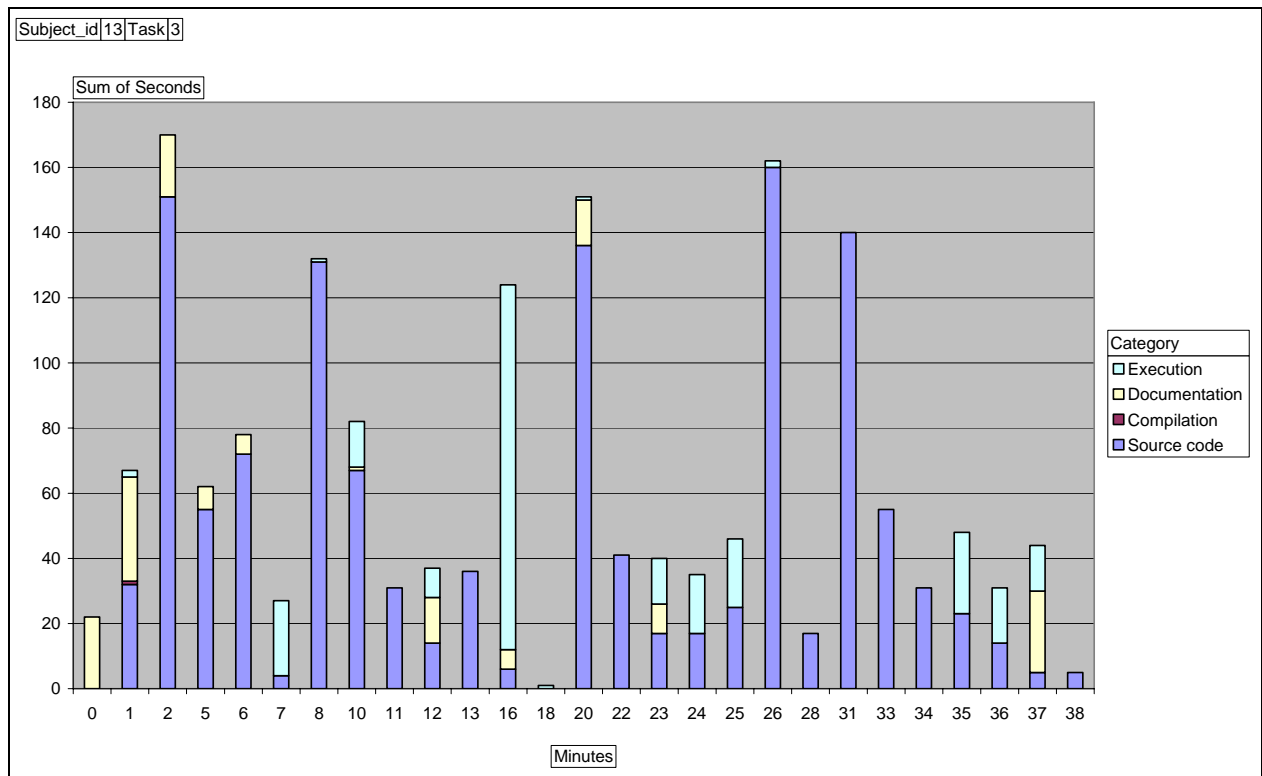


Figure 13. Example of work practise Task 3 (participant 13)

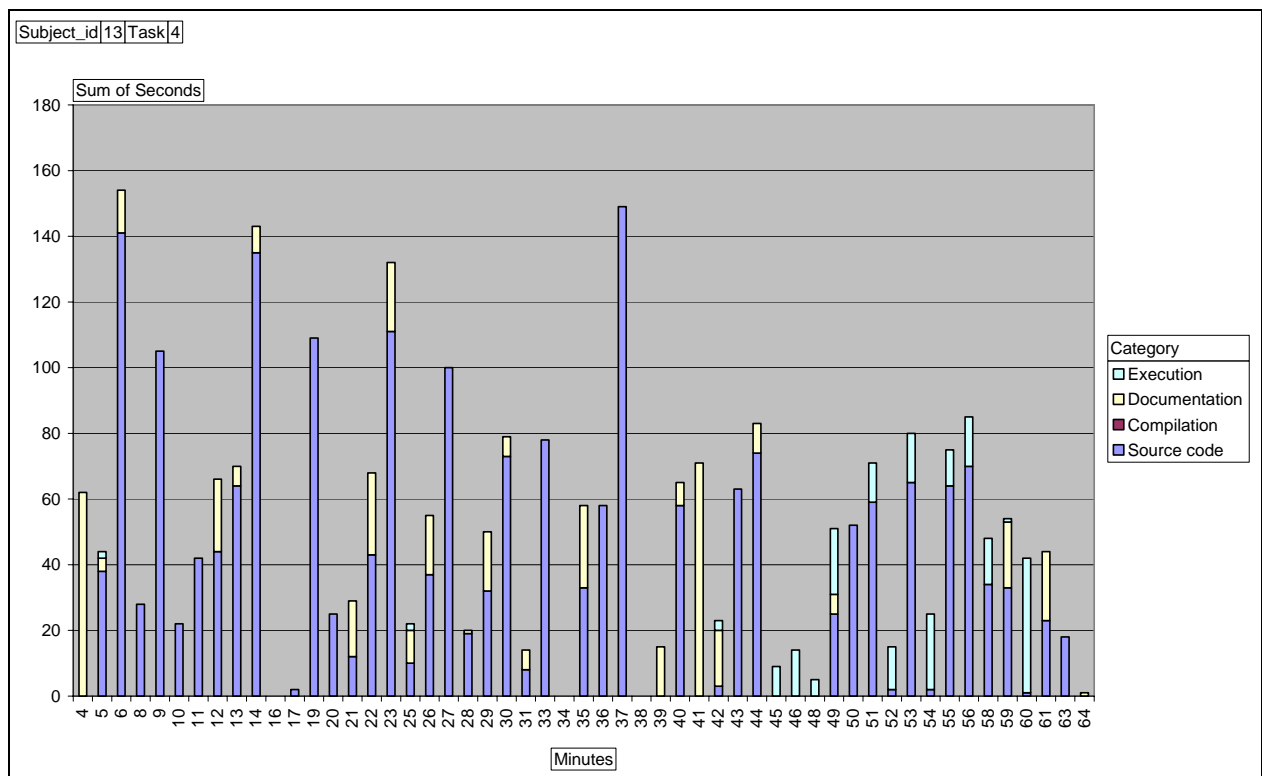


Figure 14. Example of work practise Task 4 (participant 13)

5.3 Detailed Use of the Information Sources

5.3.1 Source Code

Table 9 presents statistics of the time spent on source code in all three tasks.

Table 9. Statistics of time spent on source code during experiment

	Time (minutes)			
	Mean	Median	Min	Max
Task 2	17	18	8	40
Task 3	33	33	9	56
Task 4	36	35	0	73

The participants' use of source code varied in each task and ranged from 8 to 40 minutes (median 18) in Task 2, from 9 to 56 minutes (median 33) in Task 3 and from 0 to 73 minutes (median 35) in Task 4. The participant with 0 seconds in Task 4 did not complete the task.

Table 10 shows the number of classes visited during the experiment. The proportion of classes accessed is calculated as the median from the total number of classes.

Table 10. Statistics of number of classes accessed during experiment

	Number				Proportion of classes accessed (n = 27)
	Mean	Median	Min	Max	
Task 2	11.7	9.5	5	25	35.2%
Task 3	8	7	5	18	25.9%
Task 4	6.8	6.5	0	14	24.1%

Table 10 shows that the numbers of classes accessed were highest in Task 2 and decreased from 35.2% in Task 2 to 24.1% in Task 4. This indicates that the participants spent more time in fewer classes during the tasks. It was a considerable difference in the participants' use of classes; some accessed only the classes which needed modification while others visited almost all classes. The number of classes accessed ranged from 5 to 25 (median 9.5) in Task 2, from 5 – 18 (median 7) in Task 3 and from 0 – 14 (median 6.5) in Task 4. Examples of two different class profiles are shown in Figure 15 and 16. The participant in Figure 15 visited 21 classes while the participant in Figure 16 visited only the 5 classes which needed to be modified. The project file (.jpx) and "NotOfInterest" classes (belonging to the training task) are omitted.

The order in which the classes were visited varied. In Task 2 ten participants began by reading one of the classes that needed modification and six started with the main class StartClass.java. Only one of them started to read the classes in execution order (first StartClass.java then MainWindow.java).

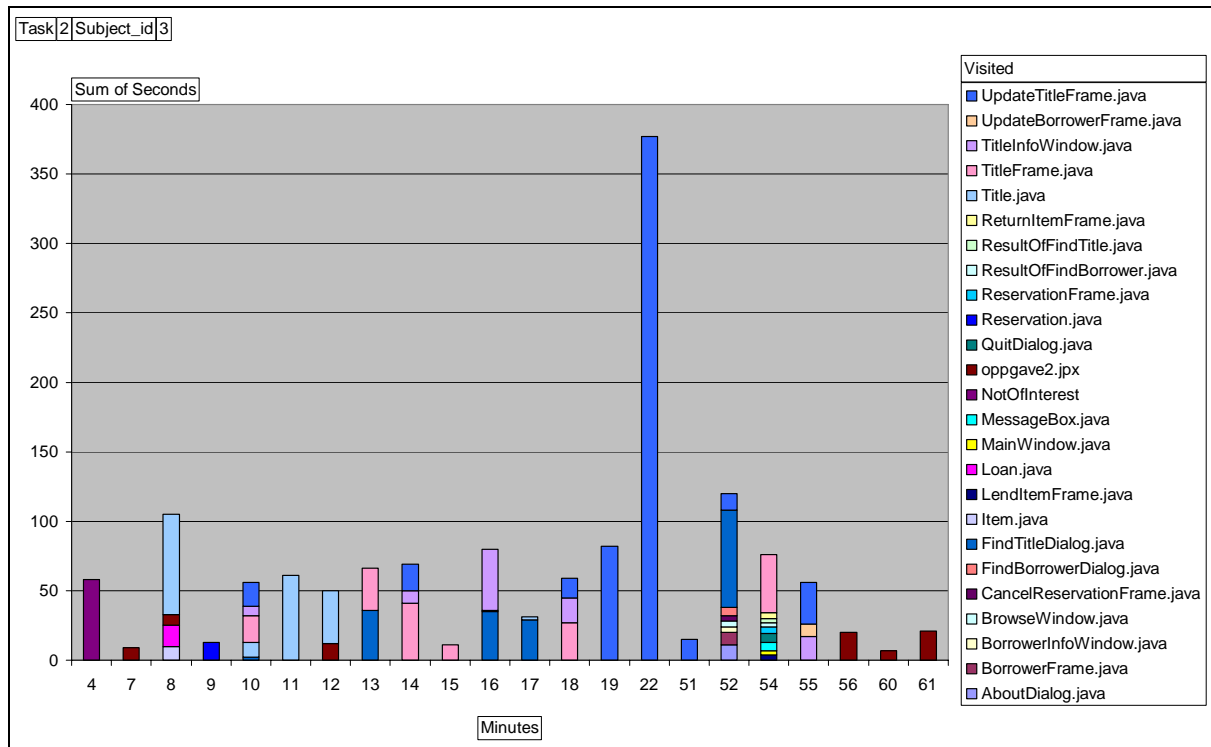


Figure 15. Example of class profile - many classes visited (participant 3, Task 2)

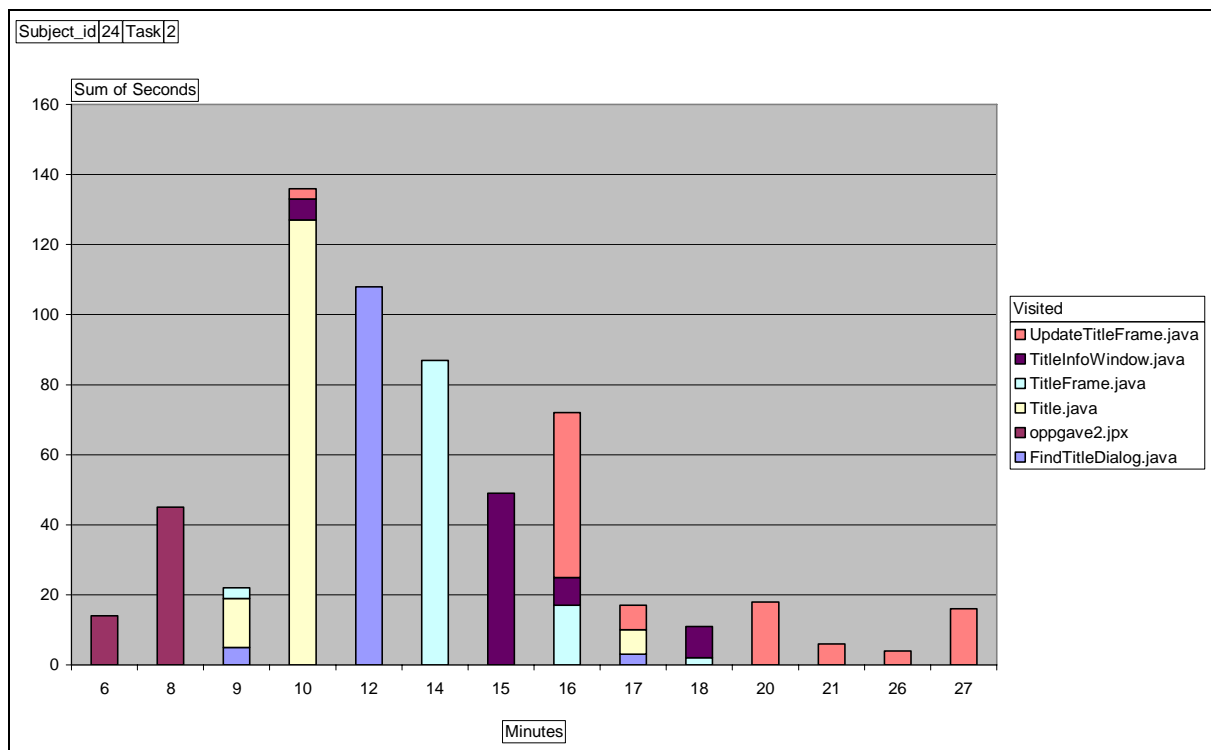


Figure 16. Example of class profile - few classes visited (participant 24, Task 2)

5.3.2 Documentation

During the experiment the participants had access to documentation about the application, task documentation and Java documentation. They could also use the internet to search for information. The participant could use what information source they desired. Descriptions of the different documents used by the participants are listed in Table 11.

Table 11. Description of documentation

Documentation	Description
System documentation	Describes functionality and structure of the library application. Text and UML-diagrams.
Task documentation (Task 2)	Task documentation. Includes screen dumps and test examples.
Task documentation (Task 3)	Task documentation. Includes screen dumps and test examples.
Task documentation (Task 4)	Task documentation. Includes screen dumps, test examples and information about Java's Calendar class.
Java documentation	Short introduction to the Java programming language
Java API	Web page. Java online documentation
Calendar	Web page. Java class
Google	Web page. Search for 'Java' using the Google search engine.

Table 12 presents the use of documentation during the tasks. The number of participants who used each document is listed in the column on the right. The mean is calculated as total time spent on the documentation divided by all participants (n = 24).

Table 12. Statistics of use of documentation

	Time (seconds)				Used by number of participants
	Mean	Median	Min	Max	
Task 2					
System documentation	149	86.5	0	928	21
Task documentation	82	77	5	260	24
Java API	0.08	0	0	2	1
Java documentation	27.5	0	0	111	1
Task 3					
System documentation	13	0,5	0	119	12
Task documentation	185	178	46	455	24
Java API	2.08	0	0	43	3
Java documentation	0.13	0	0	2	2
Task 4					
System documentation	24	0	0	135	10
Task documentation	222	196.5	33	669	24
Java API	5.5	0	0	45	5
Java documentation	1.4	0	0	33	1
Calendar	36.5	0	0	241	10
Google	1.8	0	0	42	1

The time spent on the different documents varied during the tasks, especially for the task documentation and system documentation. The use of system documentation decreased from median 86.5 seconds in Task 2 to 0 in Task 4. For the task documentation the numbers were

turned around; the participants' median time in Task 2 were 77 (lower than the system documentation) and increased to 178 in Task 3 and 196.5 in Task 4 (considerably higher than the system documentation). Both the Java documentation and the Java API webpage were little used, but the Java API was slightly more used during the tasks. The Java documentation showed no special trends. The Calendar webpage and the search engine Google were only used in Task 4.

The number of participants who used the different documentation changed noticeably during the experiment. The system documentation was used by 21 participants in Task 2, by 12 in Task 3 and by 10 in Task 4. The task documentation was of course read by all participants in all tasks. Both the Java API and the Java documentation were used by very few. The number of participants who used the Java API ranged from 1 in Task 2 to 5 in Task 4. The use of the Java documentation ranged from 1 participant in Task 2 to 2 participants in Task 3. Only 1 participant used Google to search for information about Java in Task 4. In Task 4 the webpage about the Calendar Java class was used by 10 participants.

The system documentation and task documentation were the most used during the study. In Task 4 we can observe an increased use of web pages which are related to the use of the Calendar webpage. The median time for using the Calendar webpage was 0, but the mean time was 26.2 and the maximum time was 241 which indicate that those who visited the web page spent some time there. In Task 4 the participants had to use the Calendar Java class to solve the task. Some of the information was given in the task text, but not all the method required solving the task. The participants who wanted more information about the Calendar class therefore needed to visit the webpage.

5.3.3 Compilation

The statistics of the participants' use of compilation is presented in Table 13.

Table 13. Statistics of compilation during experiment

	Number			
	Mean	Median	Min	Max
Task 2	11	9.5	2	24
Task 3	23	19.5	5	72
Task 4	16	16.5	0	47

The difference in the participants' use of compilation was considerable. In Task 2 it ranged from 2 to 24 times, in Task 3 from 5 to 72 times and in Task 4 from 0 to 47 times. The participant with 0 times in Task 4 did not complete the task. The median indicate that compilation was often used by most participants. The median for compilation increased from 9.5 times in Task 2 to 19.5 in task 3. In Task 4 it decreased to 16.5 which may be due to problems related to solving Task 4 as discussed in Chapter 6, Validity. 19 participants compiled only 1 or 2 times during the initial phase of task 2. The most compiled in the initial phase was 7 times and this participant spent most time on classes in this phase.

I found that in addition to testing code modifications, compilation was also used to locate code fragments which needed to be modified. The feedback-collection showed that the participants

especially in the corrective task (Task 2) used compilation to find the places where the source code had to be modified:

“Have compiled and am searching for places which must be updated with assistance of debug messages” [Participant 15]

“I’m using the compiler to locate the use of ISBN in GUI, remove objects and see where compiler-faults arise” [Participant 10]

“I have begun to remove references to ISBN in the code. I began with commenting out the references to ISBN in the ‘Title’- class. Then I tried to compile, the error message which showed in JBuilder pointed at references to ISBN, which I also commented out.” [Participant 12]

“Have used the strategy to remove ISBN from the Title class and compile the code to comment out the code which don’t compile because of missing ISBN. To remove form GUI I have used “find in path” in package UI where I search for ISBN and comment our all occurrences. Realize that this is an effective, but high-risk strategy.” [Participant 15]

5.3.4 Execution

The participants’ use of execution is shown in Table 14.

Table 14. Statistics of execution during the experiment

	Time (seconds)			
	Mean	Median	Min	Max
Task 2	196	189,5	58	522
Task 3	305	292	78	645
Task 4	247	206	0	724

The median execution time was 189.5 seconds in Task 2 and increased to 292 seconds in Task 3 and 206 in Task 4. The average time was 196 seconds in Task 2 and 305 and 247 in Task 3 and 4. The results showed that executions were used by all participants except two in Task 4. One didn’t complete the task and the other compiled, but didn’t manage to run the application due to problems with JBuilder.

The total proportion of execution out of total time in each task was 13.6% in the initial phase of Task 2, 12.5% for the total Task 2, 12.3% for Task 3 and 7.2% in Task 4. The use of execution was quite similar for Task 2 and 3 and slightly higher during the initial phase of Task 2. The lower percentage in Task 4 is probably related to the complexity of the task. Only 50% had a correct solution and at least 3 solutions did not compile.

Execution was of course used to test modifications, but in addition the findings in this study indicate that execution was used to understand the application in the initial phase. The higher percentage of execution during the initial phase of Task 2 indicates that execution was used to get

an overview of the application. This is confirmed by the feedback-collection and is augmented in section 5.1. In Task 3 and Task 4 execution was mainly used to test the modifications.

5.4 Work Practises of the Best and Worst Participants

I examined the work practises of the participants who performed extremely well or bad in order to see what characterizes them and whether they were different or similar. I selected those who had the best mean score at the tasks and those who had the lowest score and no functional solutions. It was only two participants who had a mean score of 5 and only two who had no functional solutions. The two worst participants had a mean score of 1.33 and 1.67. The mean score of all participants was 3.22 for all tasks. The mean score for each task was 2.54 for Task 2, 4.2 for Task 3 and 2.9 for Task 4. The statistics of quality of solution for the two best, the two worst and the mean for all participants is presented in Table 15. The medians are in parentheses.

Table 15. Statistics of quality of solution

Participant	Mean score all tasks	Task 2	Task 3	Task 4
13	5	5	5	5
23	5	5	5	5
4	1.33	1	2	1
20	1.67	1	2	2
All (24)	3.22	2.54 (2)	4.2 (5)	2.9 (2.5)

5.4.1 Initial Strategy

The participants' solution times and thus the time spent on the initial phase of Task 2 were different. In order to compare the use of information sources I have therefore expressed each participant's use of information sources as a percentage out of the total time used. The proportion of information sources used by the two participants who performed best (13 and 23) and worst (4 and 20) are shown in Table 16.

Table 16. Proportion of information sources used in the initial phase of Task 2 - best and worst

Participant	Execution	Source code	System documentation	Task documentation
13	8%	14%	69%	9%
23	26%	34%	7%	33%
4	4%	22%	56%	18%
20	5%	76%	0	18%

Participant 4 and 13 mainly used the system documentation during the initial phase. Participant 23 spent about the same time on source code as executing the application. All these participants used all information sources. Subject 20 spent distinctly most time on source code and didn't use the system documentation at all. As we can see the only difference between the best and the worst is that the best executed the application slightly more. The activities among the best and worst were quite different. One spent most time on system documentation while the other spent

most time on source code. The work practises of participant 13 (best) and 4 (worst) were in fact more corresponding.

The following excerpts from the feedback-collection explain the participants' use of information sources:

"I read the task description and the system documentation to get an impression of the application and how it works. Thereafter I began to look at the references to the "getISBN"-method in the Title class. I commented out the reference, began to search through all the source code after ISBN-references to comment these out." [Participant 4]

"Looked quickly at the system documentation, started with business object to remove field. Worked further with removal of ISBN of all GUI that had title in the name. Tested the application" [Participant 13]

"Found a place in the code where the ISBN number was referred. Commented out, got error messages, and commented out until no more error messages. Tested the application and saw that the ISBN number was displayed in the GUI, went into the GUI code and removed all code where the ISBN number was referred. Didn't bother to think about what really happened in the code, just removed code and gambled that it worked well. And so it did." [Participant 20]

"First I read the system documentation. Read the task documentation. Searched for all places where ISBN appeared, commented out all these (plus diversions as isbnField, connected fields as label1). Checked if there was any inheritance dependence since the fields are named the same in all views, but there was not. Am now checking the application to see if it does what it is supposed to." [Participant 23]

From these reports we can see that three of the participants (both the best) used the system documentation to get an impression of the application worked before they performed the task. One of the participants who performed worst explicitly wrote that he didn't want to understand the application. It is also worth noticing that the participant who spent almost 70% of the time on the system documentation stated that he *"looked quickly at the system documentation"*. This is an example of a statement which is in contrast to what the participant actually did. The excerpts also show that three of them (both the worst) used JBuilder's search function to find the references to ISBN when they solved Task 2.

5.4.2 Work Practise during the Tasks

Table 17 presents the statistics of the best (13 and 23) and worst (4 and 20) participants' use of source code, documentation and execution during the tasks. The percentage distribution of information sources is calculated out of each participant's usage of time on the task.

Table 17. Statistics of use of information sources during for each task – best and worst

	Proportion of total time in task:			Total time (seconds)
	Source code	Documentation	Execution	
Task 2				
Participant 13	56%	30%	14%	1594
Participant 23	70%	12%	18%	658
Participant 4	80%	17%	3%	2248
Participant 20	86%	5%	8%	1292
Task 3				
Participant 13	75%	9%	16%	1715
Participant 23	67%	2%	30%	2118
Participant 4	84%	6%	10%	3255
Participant 20	78%	12%	10%	2121
Task 4				
Participant 13	77%	15%	7%	2793
Participant 23	80%	3%	17%	2216
Participant 4	85%	5%	9%	1396
Participant 20	78%	13%	9%	2728

As we can see the differences in use of information sources are not very distinct, but there are some differences. In Task 2 the subjects who performed best had different working practises, but they spent noticeable less time on source code and more time executing the application than those two who performed worst. This is also repeated for Task 3. In Task 4 the difference is not as distinct concerning execution but they still spent less a less proportion of time on the source code.

All participants alternated between classes, documentation and execution, but the length of these working cycles differed. Counting the alternations between the different information sources showed that the two best alternated 32 and 17 times in Task 2 while the two worst alternated 23 and 20 times. The alternations in Task 3 were 43 and 13 times for the best and 79 and 52 for the worst. In Task 4 the best alternated 68 and 61 times while the worst alternated 16 and 55 times. There were no specific differences between the best and the worst in Task 2. The worst alternated most in Task 3 while the best alternated most in Task 4. The best alternated distinctly more in Task 4 than in Task 3 while the numbers for the worst were turned around; they alternated far more in Task 3 than in Task 4. The total profile graphs showed that one of worst participants executed the application only during the first third of Task 4. The lack of execution during the rest of the task was probably due to problems with solving the task. The assessment of correctness confirms that his solution did not compile.

5.4.3 Detailed Use of the Information Sources

Source code

The number of classes visited by the best (13 and 23) and the worst (4 and 20) are presented for each task in Table 18. The best participants visited fewer classes both in the beginning and during the task. The number of classes accessed decreased during the tasks, especially for the best participants. The worst accessed fewer classes from Task 2 to Task 3, but participant 4 accessed the same number of classes in Task 2 and 4. The best participants spent a lower percentage of

their time on source code during the study, see Table 17 section 5.4.2.

Table 18. Number of classes visited - best and worst

	Task 2	Task 3	Task 4
Subject 13	10	6	5
Subject 23	5	5	4
Subject 4	9	7	9
Subject 20	24	8	7

Use of the documentation

The two best participants used the system documentation in Task 2 and 3, and one of them also used it in Task 4. One of the worst didn't use the system documentation at all. The task documentation was used by all, but the worst spent more time on it. The Calendar webpage was used by both the best, but only by one of the worst. The main difference between the best and worst was that the best participants used more documentation and the worst spent more time in the task documentation.

Compilation and execution

The two best participants executed the application more in all tasks (see Table 23), but compiled noticeably fewer times than the worst in Task 2 (4 and 2 versus 12 and 18) and Task 3 (13 and 5 versus 46 and 25). In Task 4 the best compiled most which might be related to the worst participants' problems in solving the task.

5.5 Summary and Discussion

5.5.1 Initial Strategy

The results showed that the participants used various information sources to familiarize themselves with the application. They spent most time on the source code which confirms that source code is an important source of information even when the application is unknown. The second most used information source was the system documentation which was used by 83% of the participants. In the study by Corritore and Wiedenbeck (2001) they found that object-oriented programmers used the documentation files heavily during the study period of an 822 LOC program. Our results are similar, even though these participants studied the program *before* conducting the maintenance tasks in contrast to the participants in our experiment who were given the documentation and the program *at the same time*, and decided themselves whether they wanted to understand the application before conducting the tasks.

Compilation and execution proved to be important for comprehension. All the participants compiled at least one time and 96% ran the application median 35 seconds when familiarizing themselves with the program. Statements from the feedback-collection showed that they executed the program to get an impression of the system and how it worked. Programmers' use of execution as an information source to comprehend a program has not been emphasized in previous comprehension studies and the availability of compilation and execution facilities differs. Many studies have used hard-copy versions of source code (Pennington 1987; Soloway

and Ehrlich 1989; Ramalingam and Wiedenbeck 1997; O'Brien, Buckley et al. 2004) or lack the provision of these facilities (Parkin 2004). Especially studies of program comprehension in the procedural paradigm have presented hard-copy versions of source code. These are listed in O'Brien et al.'s review (2005) of previous empirical work in the area of program comprehension. There are various reasons for using hard-copy representation as described by O'Brien et al. (2004), but omitting the compilation and execution facilities in program comprehension studies can cause limitations on the validity of the findings. DeLine et al. (2005) claims that a programmer today typically uses a development environment, like Emacs, Visual Studio or Eclipse, both to learn about the unfamiliar code and to perform the development task. The provision of these facilities makes the experiment more realistic.

5.5.2 Work practises during the tasks

The participants spent distinctly most time on source code during the tasks. This is not surprising due to that the maintenance tasks implied editing the source code. The source code is however also a source of information about the application. Source code as programmers main source of information were reported in both Singer's survey (1998) of maintenance engineers work practises and Seaman's (2002) survey of maintainers information gathering strategies. The importance of source code as an information source was also observed by Singer et al.'s in an observational study (1997) of programmers' work practises when conducting maintenance on a large system (several millions LOC) written in a proprietary high-level language. They found that the programmers spent considerable time just looking at source code in addition to writing code and other activities. Ko et al. (2005) studied programmers conducting maintenance on a 503 LOC Java program and reported that activities related to reading and editing code counted for nearly 50% of the registered events. Programmers' use of source code as information source and the fact that maintenance implies modification of source code justify that much of previous research in program comprehension has specifically focused on source code as programmers' main source of information.

The second most used information source was the system documentation. In the initial phase the system documentation was used by 87.5% of the participants. The results show that system documentation is useful to get an overview of an unknown application. The importance of system documentation were also reported in the survey by Lethbridge et al. (2003) where more than 50% percents of the professional programmers perceived system documentation effective when learning a new system. Koenemann et al. (1991) reported that the abstract charts were the only documentation frequently used in their study of professional programmers performing maintenance on 636 LOC procedural application. Singer's interview study of maintenance practises (1998) also reported the importance of abstract documentation. She found that the more abstract the documentation was, the more the programmers trusted it. Our system documentation contained both a short user manual in how to use the library application and UML-diagrams which can be described as abstract. The participants may have used the system documentation because of the UML-diagrams, but our results don't show whether they read the manual or the diagrams.

The results of this study show that the use of information sources changed during the phases of the experiment. The higher proportion of use of the system documentation during the initial phase together with the use of execution to get an overview of the application indicates that the participants used a top-down approach when familiarizing themselves with the new application. The use of system documentation decreased distinctly during the tasks indicating an increasingly bottom-up strategy. The scope was also influenced by the phase of the experiment because the participants accessed distinctly fewer classes from Task 2 to Task 4. The participants' greater knowledge of the application may result in less need for the system documentation during the tasks and made it easier to find the classes which needed modification. This results are consistent with the findings of Corritore and Wiedenbeck (2001) although their object-oriented program was only 822 LOC compared to ours with 3600 LOC. This indicates that their results can be generalised to larger programs.

I found that even though the use of system documentations decreased throughout the tasks, it was still used for short periods during Task 3 (12 participants) and Task 4 (10 participants). This is probably because the participants needed to check out the structure of the application. This corresponds with the results of von Mayrhauser and Vans (1996) who suggested a mixed comprehension strategy in large applications when they observed professional programmers maintenance work practises.

The two most used documents in the study were the system documentation and the task documentation, but the use of these was very different during the tasks. The system documentation was used more in Task 2 (corrective task) and less in the other task. The task documentation was used less in Task 2 and more in Task 3 and Task 4 (enhancement tasks). This can be due to that Task 3 consisted of three subtasks and Task 4 contained information about the Calendar Java class which was important for solving Task 4. The frequent use of task documentation in the enhancement tasks can also be due to that the participants verified their modifications against the task documentation, as suggested by Parkin (2004). He argued that the use of documentation depended on the type of task. He studied programmers performing either corrective or enhancement tasks on 281 LOC procedural program and found that the programmers performing the correction task utilized system documentation significantly more than programmers undertaking an enhancement. Even though our program is much larger and written in the object-oriented paradigm our results support the findings of Parkin.

The participants had access to internet and search options, but only one used a search engine to search for Java related topics. The Java API web page was little used, except in Task 4 where 10 participants visited the Calendar webpage which was very relevant for solving the task. This indicates that the participants had an as-needed approach to the Java API and that the possibility to access the Java API is important for providing realistic working conditions.

The order in which the classes were accessed varied. Even though six participants started with the main class, StartClass.java, in task 2, I found only one who visited the classes in execution order. This may be due to that the application was too large to be read in execution order and that the execution gave an impression of how the application worked. Burkhardt et al. (1998) characterized reading the classes in execution order as dynamic guidance and found that object-oriented experts tended to use this less than novices while they conducted documentation or reuse tasks on a 550 LOC program. Our results is in contrast to the findings of Nanja and Cook (1987)

who found that the experts employed a comprehension approach by first reading the program in execution order before they debugged a 73 LOC procedural program. This may be due to that their program was very small, written in the procedural paradigm and the lack of system documentation in the study.

The findings show that compilation and execution were often performed by the participants. In addition to testing the modifications, compilation was also used in the corrective task (2) to locate the code that must be altered. In the initial phase execution was used to understand the new program, as described in the initial strategy, section 5.5.1. Parkin (2004) argued that in corrective maintenance the experts appear to utilize compilation and execution sparingly, but our findings don't support this. Execution was mostly used in the corrective task (2). Ko et al.'s (2005) study of object-oriented maintenance programmers work practises, Singer et al.'s observational study (1997) of professional programmers work practises and studies of procedural programmers' debugging behaviour (Nanja and Cook 1987; Takada, Matsumoto et al. 1994) confirm that compilation and execution facilities are prevalently used.

By studying the participants work practises I found that their work cycles consisted of program modifications, reading documentation, program compilation and execution. The length of program modification before reading documentation and testing the modification were different between the participants. This can be due to expertise as claimed by Nanja and Cook (1987). They found that the programmers performed the same repeated steps of compilation, execution and program modification, but the experts corrected multiple errors before executing the program and thus had less number of runs. The experts performed best, but this is in contrast to our findings because the two best participants in our study executed the program distinctly more than the worst in Task 2 and Task 3. The effects of different levels of expertise are not addressed in this thesis.

5.5.3 Work Practises of Best and Worst Participants

The main differences between the two best and the two worst participants during the tasks were that the best executed the application more, compiled less and accessed fewer classes. The two worst spent more time on source code and the task documentation. Regarding the initial phase, there was no particular difference except that the best executed the application slightly more.

The best participants' access of fewer classes can be explained by that they achieved a greater knowledge of the application and therefore easily found the classes which needed modification. The worst participants' less use of execution may indicate that they had problems in utilizing this facility to comprehend the dynamic of the program. It is also possible that they didn't see the benefit of program execution. Both the best stated in the feedback-collection that they executed the program to get an overview, but none of the worst did that.

6 Validity

This chapter discusses the most important threats to validity of this study. Some parts of the sections are also discussed in Kværn (2006), but they are especially related to her research.

6.1 Experimental Design

In this experiment we wanted to log all actions performed by the participants in order to see what information sources they used and how they spent the time during the tasks. All available material was thus presented electronically and they didn't receive any documentation in hard-copy. We wanted to make the experimental environment as close to everyday practise as normal, but the restrictions on use of hard-copy documentation may have caused limitation regarding that. Some participants may be accustomed to getting the documentation in hard-copy and therefore had a less normal work practise. However, none of the participants expressed a lack of hard-copies or books in the feedback-collections.

Surveys of professional programmers' maintenance work practises (Singer 1998; Seaman 2002) identified in addition to source code and documentation, several other sources of information as valuable e.g. other humans, especially when the code was unknown, CASE tools, maintenance logs and previous lessons learned. It is difficult to make all these accessible in a controlled experiment.

6.2 Programs and Tasks

The library application used in the experiment was 3600 LOC, and can be considered to be a medium-size application, according to the classification given by von Mayrhauser and Vans (1995). The application and tasks were larger than those typically used in software engineering experiment. However, the application and tasks were smaller than real-world programs and tasks, as reported by the observational field studies of Singer et al. (1997) and von Mayrhauser and Vans (1997). It is possible that the results would be different for larger applications and more complex tasks. Our results are thus limited to situations in which the programmers had to comprehend and maintain a medium system, previously unknown to them and developed by others.

The tasks required Graphical User Interface (GUI) programming which comprises special Java classes. Some participants stated in the feedback-collected that they had little knowledge of GUI classes. The participants' skills in GUI programming can thus have affected the results because the use of GUI classes may have been more difficult to understand for those who had little knowledge of them. The participants could access the Java API which has detailed information about all Java classes. This was done by only a few which indicates that most of the participants had sufficient knowledge about the classes used.

Task 4 was the most complex task and many participants had problems in solving this. It was still

included in the analysis because I was interested in what activities they performed during *all* phases even though they didn't manage to solve the task. The results for Task 4 are thus somewhat different than for the previous tasks because not all of the participants completed the solution. I have used the correctness of the solution as a measure whether they completed the task or not and equalled the solutions marked as correct (1) as completed. This is not entirely correct because participants might have completed the task and yet got an incorrect solution due to major defects. 16 solutions in Task 2 and 4 solutions in Task 3 were marked as not correct (0) even though all of them were completed.

6.3 Participants

All the participants were professional developers who worked in five different software companies. The companies selected the participants for this experiment. The differences in skills among developers are considerable, and we don't know whether those selected were representative for the companies or if they sent their least valuable employees. It is also possible that they selected their best programmers in order to make a good impression of the company. The results indicate that there was a mixture of highly skilled and novices who participated in the experiment.

6.4 Data Cleaning and Preparation

The process of data cleaning and preparation was difficult. Even though we carefully verified all data, there may be activities which have been left out. We excluded activities which were not related to solving the tasks e.g. reading on-line newspapers. In the initial strategy I excluded the activities related to unpacking the project file and classes which belonged to the training task. This was done because the main interest was to analyse the information sources used by the participants to understand the application in the initial phase. By omitting these activities the results may not give the entire picture of the participants' actions during this phase. The activities were therefore included in analysis of the work practises during the phases of the maintenance tasks.

6.5 Measuring Time

The solution time was measured as the difference between `end_time` and `start_time` subtracted major non-productive breaks as lunch breaks and breaks longer than 10 minutes. The graphs of total activity show that during these solution times there were still minutes with no activities registered which may have been spent on for instance SESE. The total times spent on the different information sources is thus calculated as the sum of times spent on documentation, classes, compilation and execution. These total times will therefore differ from the solution times.

6.6 Analysis

The participants' use of classes was used to determine the participants' employment of source code as an information source. However, we can't see whether they edited or read the code. Ko et al. (2005) reported that the programmers in their study spent about 20% of the time reading source code and code related activities as read, edit, search and navigate represented nearly 70% of all activities. The feedback-collections were used to get supplementing information about how the participants used the code, but they covered only parts of the time spent. A separation of the reading and editing would make the results more accurate regarding the use of source code as an information source.

The graphs scales of the different participants varied depending on the time spent on the various activities. If a participant spent a short time on all activities, they can appear as longer than they really are compared to scales where the participant spent longer time on some of the activities. It was therefore important to look at the scales during the analysis of the graphs.

I discovered that there sometimes were differences in what the participants said and what they actually did. It was thus useful to compare the results of logging with the feedback collection. Singer et al. (1997) also reported this in the observational study of programmers. The developers most self reported activity was reading documentation but the actual results showed that this only counted for a small part of the events registered. In addition the feedback collection was useful for complementing the data collected by GRUMPS. The feedback-collection gave reasons for the participants' activities like for instance the use of execution and compilation in the initial phase.

7 Conclusions and Future Work

Software maintenance is an important part of a programmer's work and successful modifications inevitably rely on an understanding of the program to be maintained. Programmers rely on various information sources to comprehend an unknown program. The objective of this research was to examine the participants' comprehension-related activities during the tasks regarding to:

- Initial strategy
- Work practises during the tasks
- Best and worst participants

A controlled experiment was conducted to address these issues. 24 professional programmers were introduced to an unknown 3600 LOC Java application where they performed three maintenance tasks. The participants' actions were logged during the experiment and written feedback was collected. Together this gave us a detailed picture of the participants' activities during the tasks.

7.1 GRUMPS

We have made a further development of GRUMPS which includes data cleaning and analytical preparation of low-level usage data. The extended functionality provides detailed information about each participant's use of compilation, execution and various documentations. The analysis tool combines the data from the added functionality together with the existing data and gives an overview of how the participants solved the tasks. It shows the chronological actions and time spent on source code, compilation, execution, web pages and documentation throughout the experiment. This information is useful regarding program comprehension research because we can identify the participants' comprehension-related activities when familiarizing themselves with an unknown application and during the maintenance tasks. The written documentation and SQL-code can be reused by researchers in similar studies. The data preparation and meaningful interpretation of low-level usage data is a difficult process and the reuse of our work can simplify this process.

7.2 Initial Strategy

I analysed the participants' use of system documentation, source code, task documentation, compilation and execution in their initial familiarization with the application. The results showed that the participants used a combination of 3 – 5 information sources and 66.7% used all of them to understand the application. Source code was most used, and the system documentation next. The most important finding was that execution proved to be important for the initial comprehension. The extent to which execution assists the program comprehension process has not been assessed in previous studies. 96% of the participants spent about 14% of the time executing the application. Statements from the feedback-collection showed that they executed the program to get an impression of the system and how it worked.

Identifying the use of execution as an information source to comprehend a program is useful for researchers conducting studies of program comprehension. A programmer today typically uses a development environment like JBuilder both to learn about an application and to perform the maintenance task. The lack of facilities like compilation and execution in previous comprehension studies can cause threats to the validity of the findings and reduce the experimental environment's equality to the normal work practise. Our results show that the findings of these studies can not be trusted without reservation.

7.3 Work Practises during the Tasks

The participants' work practises during the phases of the experiment were analysed in detail regarding their use of information sources. The results showed the following:

- Compilation and execution were used prevalently during the study. They were also used for other purposes but debugging.
- The use of information sources varied during the phases of the experiment and especially the initial phase of Task 2 was distinctly different from the rest of the tasks. The use of system documentation was highest in the initial phase and decreased distinctly during the tasks, but it was used by half of the participants for short periods during Task 3 and Task 4. The number of classes accessed decreased from Task 2 to Task 4.
- There was little use of dynamic guidance. Only one participant visited the classes in execution order.
- The participants work cycles consisted of program modifications, reading documentation, program compilation and execution. The length of this work cycles varied among the participants.
- Source code was the most used information source during the study.
- The system documentation and task documentation were the most used documentation.
- The Java API webpage was used when needed.

The findings showed that compilation and execution were often performed by the participants. In addition to test the modifications, compilation was also used in the corrective task (2) to locate the code that must be altered. Execution was also used to comprehend the unknown application as described in section 7.2. Parkin (2004) argued that in corrective maintenance the experts appear to utilize compilation and execution sparingly, but our findings don't support this.

The participants work cycles consisted of (1) program modifications (2) reading documentation (3) compilation and (4) execution. Previous studies (Nanja and Cook 1987; Takada, Matsumoto et al. 1994) have not included the reading of the documentation.

The results showed that the participants visited the Java API webpage when they needed more information about Java classes, especially the Calendar class which was used in Task 4. The access of the Java API was important for providing realistic work conditions.

The following findings confirm and extend previous studies and today's knowledge of program

comprehension; The direction of comprehension was top-down in the initial phase of Task 2 and gradually more bottom-up during the tasks (Corritore and Wiedenbeck 2001), the short visits to the system documentation in Task 3 and Task 4 indicate a mixed comprehension strategy (von Mayrhauser and Vans 1996), the scope was influenced by the phases of the experiment as the participants accessed fewer classes throughout the experiment (Corritore and Wiedenbeck 2001), object-oriented programmers little use of dynamic guidance (Burkhardt, Detienne et al. 1998), the length of the work cycles varied among the participants (Nanja and Cook 1987), source code as programmers main source of information (Singer, Lethbridge et al. 1997; Singer 1998; Seaman 2002) even though we can't see whether they read or edited the code and the importance of system documentation when learning a new application (Lethbridge, Singer et al. 2003).

These findings add body to the knowledge of program comprehension because they give a detailed overview of the participants work practises over the duration of maintenance tasks. This knowledge is useful for the development of more effective working methods and can be used as guidelines for novice programmers. Furthermore, the results identify the information sources which are important to gain an adequate understanding of an unknown application and therefore should be available for programmers both in controlled experiment and ordinary work.

7.4 Best and Worst Participants

The main differences between the two best participants and the two worst during the tasks were that the best executed the application more, compiled less and accessed fewer classes. Both the best stated in the feedback-collection that they executed the program to get an overview, but none of the worst did that.

Identifying the work practices of best participants is useful, particularly for training programmers in successful techniques. The results showed that the two best had a more successful employment of the JBuilder programming environment, and exploited the functionality of JBuilder to understand the dynamic of the program.

7.5 Future Work

This research has some limitations and in further research the following should be addressed:

- The analysis of the participants' use of source code did not include whether they edited or read the code because the extracted data didn't contain this information. The material in the GRUMPS database include all actions performed by the participants and can be further developed to address this issue. The identification of reading and editing will extend the knowledge of source code as an information source, especially related to the initial comprehension of an unknown application.
- The data logged by GRUMPS gives a detailed picture of how the participants executed the application. In this study I have used this material to analyse how much and when the participants executed. This material can be further analysed to see *how* the best

participants executed the application in order to identify successful execution, especially in the initial phase. Did they just register a borrower or did they systematically access all the windows in the application? This knowledge can be useful for industrial training of programmers.

- The system documentation in this experiment contained both a user manual for the library application and UML-diagrams, but the results did not explain whether the participants looked at the user manual or the diagrams. By separating these in two different documents researchers can identify whether the programmers prefer the written description or the UML-diagrams in the initial phase and during the maintenance.

Bibliography

Arisholm, E., D. I. K. Sjøberg, G. J. Carelius and Y. Lindsjørn (2002). "A web-based support environment for software engineering experiments." Nordic Journal of Computing **9**(3): 231-247.

Arisholm, E., D. I. K. Sjøberg and M. Jørgensen (2001). "Assessing the Changeability of two Object-Oriented Design Alternatives - a Controlled Experiment." Empirical Software Engineering **6**(3): 231 - 277.

Brooks, R. (1983). "Towards a Theory of the Comprehension of Computer Programs." International Journal of Man-Machine Studies **18**(6): 543-554.

Burkhardt, J.-M., F. Détienne and S. Wiedenbeck (2002). "Object-Oriented Program Comprehension: Effect of Expertise, Task and Phase." Empirical Software Engineering **7**(2): 115-156.

Burkhardt, J. M., F. Detienne and S. Wiedenbeck (1998). The Effect of Object-Oriented Programming Expertise in Several Dimensions of Comprehension Strategies. Proceedings of the 6th International Workshop on Program Comprehension, IEEE Computer Society.

Busvold, J. (2006). Problems and Problem Solving Strategies during Maintenance of Object-Oriented Systems [in preparation]. Department of Informatics, University of Oslo.

COOL. The Comprehensive Object -Oriented Learning project. [Online]. Available: http://www.intermedia.uio.no/projects/research/cool_en.html [12th April 2006].

Corritore, C. L. and S. Wiedenbeck (2000). Direction and scope of comprehension-related activities by procedural and object-oriented programmers: an empirical study. 8th International Workshop on Program Comprehension, IEEE Computer Society.

Corritore, C. L. and S. Wiedenbeck (2001). "An exploratory study of program comprehension strategies of procedural and object-oriented programmers." Int. J. Human-Computer Studies. **54**(1): 1-23.

DeLine, R., A. Khella, M. Czerwinski and G. Robertson (2005). "Towards understanding programs through wear-based filtering." Proceedings of the 2005 ACM symposium on Software visualization: 183-192.

Eriksson, E. and M. Penker (1998). Case Study. In: UML toolkit, John Wiley & Sons, Inc.

Evans, H., M. Atkinson, M. Brown, J. Cargill, M. Crease, S. Draper, P. Gray and R. Thomas (2003). "The pervasiveness of evolution in GRUMPS software." Software-practice and experience **33**(2): 99-120.

Kajko-Mattsson, M. (2005). "A Survey of Documentation Practice within Corrective

Maintenance." Empirical Software Engineering **10**(1): 31-55.

Karahasanović, A. (2005). Comprehension of Object-Oriented Systems.

Karahasanović, A., A. K. Levine and R. Thomas (2005). "Comprehension strategies and difficulties in maintaining object-oriented systems: an explorative study." Submitted to The Journal of Systems and Software.

Ko, A. J., H. Aung and B. A. Myers (2005). Eliciting design requirements for maintenance-oriented IDEs: a detailed study of corrective and perfective maintenance tasks. Proceedings of the 27th international conference on Software engineering, St. Louis, MO, USA, ACM Press.

Ko, A. J. and B. Utzl (2003). Individual differences in program comprehension strategies in unfamiliar programming systems. 11th IEEE International Workshop on Program Comprehension.

Koenemann, J. and S. P. Robertson (1991). Expert problem solving strategies for program comprehension. New Orleans, Louisiana, United States, ACM Press.

Kværn, K. (2006). Effects of Expertise and Strategies on Program Comprehension in Maintenance of Object-Oriented Systems: A Controlled Experiment with Professional Developers. Department of Informatics, University of Oslo.

Lethbridge, T. C., J. Singer and A. Forward (2003). "How Software Engineers Use Documentation: The State of the Practice." IEEE Software **20**(6): 35-39.

Levine, A. (2005). A Study of Comprehension Strategies and Difficulties by Novice Programmers Performing Maintenance Tasks of Object-Oriented Systems. Department of Informatics, University of Oslo.

Littman, D. C., J. Pinto, S. Letovsky and E. Soloway (1986). "Mental models and software maintenance." Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers: 80-98.

Misanchuk, E. R. and R. Schwier (1992). "Representing interactive multimedia and hypermedia audit trails." Journal Of Educational Multimedia and Hypermedia **1**(3): 413-431.

Nanja, M. and C. R. Cook (1987). An analysis of the on-line debugging process. Empirical studies of programmers: second workshop, Ablex Publishing Corp.: 172-184.

O'Brien, M. P. and J. Buckley (2005). Modelling the Information-Seeking Behaviour of Programmers - An Empirical Approach. Proceedings of the 13th International Workshop on Program Comprehension - Volume 00, IEEE Computer Society: 125-134.

O'Brien, M. P., J. Buckley and C. Exton (2005). Empirically Studying Software Practitioners " Bridging the Gap between Theory and Practice. Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05) - Volume 00, IEEE Computer Society: 433-

442.

O'Brien, M. P., J. Buckley and T. M. Shaft (2004). "Expectation-based, inference-based, and bottom-up software comprehension." Journal Of Software Maintenance And Evolution-Research And Practice **16**(6): 427-447.

Parkin, P. (2004). An exploratory study of code and document interactions during task-directed program comprehension. Australian Software Engineering Conference.

Pennington, N. (1987). "Comprehension strategies in programming." Empirical studies of programmers: second workshop: 100-113.

Ramalingam, V. and S. Wiedenbeck (1997). An empirical study of novice program comprehension in the imperative and object-oriented styles. Alexandria, Virginia, United States, ACM Press.

Reeves, T. C. and J. G. Hedberg (2003). Interactive Learning Systems Evaluation. Englewood Cliffs, NJ, Educational Technology Publications.

Renaud, K. and P. Gray (2004). Making sense of low-level usage data to understand user activities. Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, Stellenbosch, Western Cape, South Africa, South African Institute for Computer Scientists and Information Technologists.

Seaman, C. (2002). The Information Gathering Strategies of Software Maintainers. Proceedings of the International Conference on Software Maintenance (ICSM'02), IEEE Computer Society: 141.

Singer, J. (1998). Practices of Software Maintenance. Proceedings of the International Conference on Software Maintenance, IEEE Computer Society: 139.

Singer, J., T. Lethbridge, N. Vinson and N. Anquetil (1997). An examination of software engineering work practices. Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada, IBM Press.

Soloway, E. and K. Ehrlich (1989). Empirical studies of programming knowledge. Software reusability: vol. 2, applications and experience, ACM Press: 235-267.

Takada, Y., K. Matsumoto and K. Torii (1994). A programmer performance measure based on programmer state transitions in testing and debugging process. Proceedings of the 16th international conference on Software engineering, Sorrento, Italy, IEEE Computer Society Press.

Thomas, R., G. Kennedy, S. Draper, R. Mancy, M. Crease, H. Evans and P. Gray (2003). Generic usage monitoring og programming students. Proceedings of the 20th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCILITE), Adelaide, Australia.

Thomas, R. C. and R. Mancy (2004). Use of large databases for group projects at the nexus of teaching and research. Leeds, United Kingdom, ACM Press.

Torchiano, M. (2004). Empirical Investigation of a Non-Intrusive Approach to Study Comprehension Cognitive Models. Proceedings of the Eighth Euromicro Working Conference on Software Maintenance and Reengineering (CSMR'04), IEEE Computer Society.

von Mayrhauser, A. and A. M. Vans (1994). Comprehension processes during large scale maintenance. Proceedings of the 16th international conference on Software engineering, Sorrento, Italy, IEEE Computer Society Press.

von Mayrhauser, A. and A. M. Vans (1995). "Program Comprehension During Software Maintenance And Evolution." Computer **28**(8): 44-55.

von Mayrhauser, A. and A. M. Vans (1996). "Identification of Dynamic Comprehension Processes During Large Scale Maintenance." IEEE Transactions on Software Engineering **22**(6): 424-437.

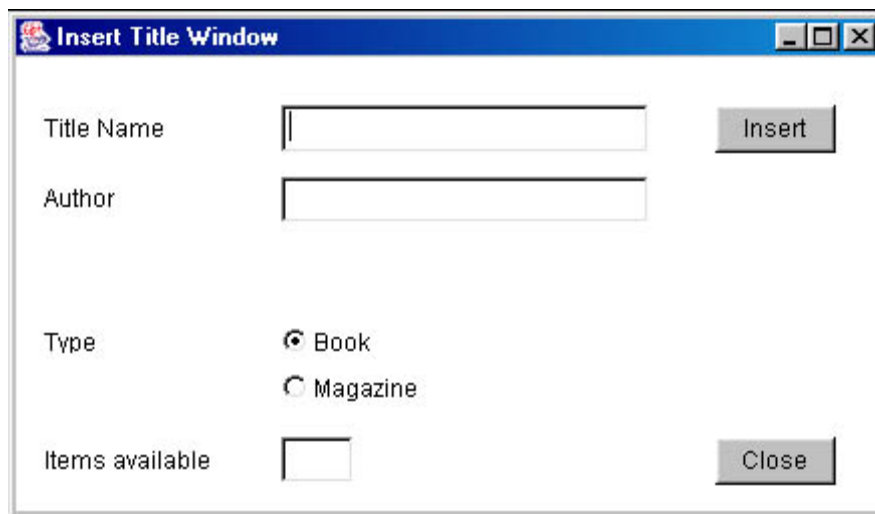
von Mayrhauser, A. and A. M. Vans (1997). Program understanding behavior during debugging of large scale software. Alexandria, Virginia, United States, ACM Press.

Appendix A - Tasks

This appendix contains the full task texts and detailed descriptions of the tasks.

Task 2

Until now the ISBN-numbers have been stored in the system. A new international system for categorisation has been accepted, so this information is no longer needed. You must remove ISBN information from all the places where it has been used (you may comment it out). Also remember to remove all ISBN-text fields from the user interface. Here is an example on how one of the windows should look when you complete the task:



The screenshot shows a dialog box titled "Insert Title Window" with a blue header bar and standard window controls (minimize, maximize, close). The dialog contains the following fields and controls:

- Title Name:** A text input field with a vertical cursor on the left.
- Author:** A text input field.
- Type:** Two radio button options: "Book" (selected) and "Magazine".
- Items available:** A small text input field.
- Buttons:** An "Insert" button is located to the right of the Title Name field, and a "Close" button is located to the right of the Items available field.

Task 3A

You are going to add the text "E-mail" to the window to insert new borrowers. You don't have to create a new text field for reading the E-mail in this part of the task. The text "E-mail" should be placed under "State" as shown in the picture:

The screenshot shows a Java Swing window titled "Insert Borrower". The window has a standard title bar with minimize, maximize, and close buttons. Inside the window, there are seven text input fields arranged vertically, each with a label to its left: "Last Name", "First Name", "Address", "City", "Zip", "State", and "E-mail". To the right of the "Last Name" field is a button labeled "Insert". To the right of the "State" field is a button labeled "Close". The "E-mail" field is at the bottom of the form and does not have a corresponding button.

Task 3B

You should add a text field for E-mail to the class that contains data regarding borrowers. To write this field to the file and read from it you have to change the methods `read()` and `write()`. You must change the window for inserting borrowers so that E-mail can be entered. Remember to remove the *.dat-files before testing. The window should look like the screenshot below. You can test if you have written E-mail to the file by opening the .dat-file in Notepad.

Insert Borrower

Last Name

First Name

Address

City

Zip

State

E-mail

Task 3C

You should now change the window for updating borrower so that E-mail can be shown. The window should look like the screenshot below. Then you should change all other windows that show borrower's information in the same way.

Update / Delete Borrower

Last Name

First Name

Address

City

Zip

State

E-mail

Task 4

You should add functionality for presenting the borrower loan due. The loan is due 4 weeks after the loan was made. You are going to show in the BorrowerInfoWindow-class that a borrower's loan has expired. An illustration of the window after new functionality has been added is shown below:

The screenshot shows a window titled "Borrower Information" with the following fields and data:

Field	Value
Last Name	nordmann
First Name	ola
Address	postveien 3b
City	byen
Zip	1881
State	norge

Buttons: Find, OK

Title Reservations:

- lesebok
- java II

Title Loans / Expiry date:

- Sangboken (Item:1) / EXPIRED!
- boka mi (Item:1) / 23.10.2003
- Learning UML (Item:3) / 11.11.2003

You can use Java's Calendar-class

<http://java.sun.com/j2se/1.4.2/docs/api/java/util/Calendar.html>) to handle dates (java.util.Calendar).

Calendar.getInstance() returns a GregorianCalendar-object where the time fields are initialized with date and time.

Some examples (you might need some methods in this task):

```
Calendar rightNow = Calendar.getInstance();
int year = rightNow.get(Calendar.YEAR);
int day = rightNow.get(Calendar.DATE);
int month = rightNow.get(Calendar.MONTH) + 1;

// Subtract 5 days from the date
rightNow.add(Calendar.DATE, -5);

// Returns true when rightNow is earlier than baseCal
boolean b = rightNow.before(baseCal);

// Set the Calendar to a specific date (year, month, day, hour,
minute, second)
rightNow.set(2003, 7, 21, 10, 30, 30);
```


Detailed description of the tasks

Task 2

This was an alteration task which did not require the participant to make any addition of code, just remove some of the existing code. However, the application was rather large, and the task required the participants to make changes several places in the code. Although the task required changes in quite a few classes, it could be done simply by performing a search for ISBN through all the files (classes) and deleting the findings. From the total of 27 classes in the application, there were 5 classes containing ISBN. Table 19 shows an overview of the classes that had to be altered in this task, with a description of what had to be done for each of the classes.

Table 19. Necessary alterations for task 2

File name	Changes required
Title.java	This is the entity object in the bo-package where isbn was stored. Changes needed on: <ul style="list-style-type: none">- Constructor- Get/set methods- Read and write methods for persistency- A find method (if else)
FindTitleDialog.java	Ui-package. Changes needed <ul style="list-style-type: none">- Fields and labels with corresponding getText/setText methods- Declarations- FindButton_Clicked
TitleFrame.java	Ui-package. Changes needed: <ul style="list-style-type: none">- Fields and labels with corresponding getText/setText methods- Declarations- AddButton_Clicked
TitleInfoWindow.java	Ui-package. Changes needed: <ul style="list-style-type: none">- Fields and labels with corresponding setText method- Declarations
UpdateTitleFrame.java	Ui-package. Changes needed : <ul style="list-style-type: none">- Fields and labels with corresponding getText/setText methods- Declarations- UpdateButton_Clicked

Task 3

This task was divided into three parts to make it somewhat less complex for the participants:

The first part was merely to add a piece of text to the user interface. A screen shot of the window to be altered was given in the task text, so the participants could easily see in which class to do the editing.

The second part was a bit more complex. The participants should add an input field to the UI-class they had already altered in the former part. Also, they had to alter two methods in the entity class – read and write - to make sure that E-mail could be saved to file. They were given a hint to which class they had to alter (“the class containing information about a borrower”), and also which methods to alter (“read” and “write”).

The third and last part was the most extensive of the three parts. The participants had to alter all the windows in the user interface which contained information about E-mail. However, the alteration task itself was similar to the one performed in the previous parts of this task. Table 20 shows an overview of the alterations that had to be made.

Table 20. Necessary alterations for task 3

File name	Changes required
BorrowerInformation.java	This is the entity object in the bo-package where email should be stored. Changes needed on: <ul style="list-style-type: none"> - Constructor - Get/set methods - Read and write methods for persistency
BorrowerInfoWindow.java	Ui-package. Changes needed: <ul style="list-style-type: none"> - Fields and labels with corresponding getText/setText methods - Declarations
BorrowerFrame.java	Ui-package. Changes needed: <ul style="list-style-type: none"> - Fields and labels with corresponding getText/setText methods - Declarations
FindBorrowerDialog.java	Ui-package. Changes needed: <ul style="list-style-type: none"> - Fields and labels with corresponding getText/setText methods - Declarations
UpdateBorrowerFrame.java	Ui-package. Changes needed: <ul style="list-style-type: none"> - Fields and labels with corresponding getText/setText methods - Declarations

Task 4

This task was the most complex of the three. The participants had to introduce a new variable containing the date of a loan, and this variable had to be saved and loaded from file. The participants were given no hints as to where to place this variable (in which class). They also had to change the user interface, calculate an expiry date (4 weeks after the loan was made) and include logic to check the expiry date against today’s date. They were given the name of the class in which to do this. What also complicated this task was that they had to use a class – Calendar – which was previously unknown to most of the participants. Some information on the usage was

given in the task text, but not all the methods required solving the task. Table 21 shows an overview of the alterations that had to be made during Task 4.

Table 21. Necessary alterations for task 4

File name	Changes required
Loan.java	This is the entity object in the bo-package where the loan date should be stored. Changes needed on: <ul style="list-style-type: none">- Constructor- Get/set methods- Read and write methods for persistency- Logic for calculating expiry date
BorrowerInfoWindow.java	Ui-package. Changes needed: <ul style="list-style-type: none">- Method for displaying the expiry date in the window

Appendix B - The Application

The application consisted of four packages. The sections below contain an overview of the classes in each package.

User Interface Package

The User Interface package (Ui) is on top of the other packages. It presents the services and information in the system to the user. The package cooperates with the business objects package. The User Interface package calls operations on the business objects to retrieve and insert data into them. See Table 22 for an overview of the classes in this package.

Table 22. Classes in the User Interface package

Class name
AboutDialog.java
BorrowerFrame.java
BorrowerInfoWindow.java
BrowseWindow.java
CancelReservationFrame.java
FindBorrowerDialog.java
FindTitleDialog.java
LendItemFrame.java
MainWindow.java
MessageBox.java
QuitDialog.java
ReservationFrame.java
ResultOfFindBorrower.java
ResultOfFindTitle.java
ReturnItemFrame.java
TitleFrame.java
TitleInfoWindow.java
UpdateBorrowerFrame.java
UpdateTitleFrame.java

Business Objects Package

This is the Business Objects package (bo) in the design. All business object classes inherit the Persistent class in the Database package which is described in the next section. Each class has its own read and write method, but it is the services from the Database package that allows the classes to be stored persistently. See Table 23 for an overview of the classes in this package.

Table 23. Classes in the Business Object package

Class Name	Description
BorrowerInformation.java	Information about a borrower. A borrower can be a person or another library. A borrower can have loans

	and reservations.
Item.java	Represents an item, a physical instance of a Title. The library can have several items of one title.
Loan.java	Represents a loan. The loan refers to one title and one borrower.
Reservation.java	A reservation reserves a title for a specific borrower, meaning the borrower should be first in line when any item of the title is returned
Title.java	Represents a book or magazine title. A title can exist in many physical items and can have reservations connected to it.

Database Package

The Database package (db) is necessary to provide persistent storage of the objects in the Business Object package. All the implementation of persistent storage handling is done in a class called Persistent, which classes that need persistent objects must inherit. See Table 24 for an overview of the classes in this package.

Table 24. Classes in the Database package

Class Name	Description
Persistent.java	Super class for classes that wants to be persistent. Subclass must implement read and write operations that serializes the object to disk. This class reads files sequentially.

Utility Overview

The Utility package (util) contains only one class, ObjId.java, which is used to refer to persistent objects throughout the system. The class is used both in the user-interface, business object, and database package. See Table 25 for an overview of the classes in this package.

Table 25. Classes in the Database package

Class Name	Description
ObjId.java	Represents an object id, a class that works as "pointer" to any persistent object in the system.
