Empirical Software Engineering, 6, 231–277, 2001.
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Assessing the Changeability of two Object-Oriented Design Alternatives—a Controlled Experiment

ERIK ARISHOLM*, DAG I. K. SJØBERG* AND MAGNE JØRGENSEN*
erik@simula.no, dag@simula.no, magne@simula.no
Simula Research Laboratory, Oslo Research Park, Gaustadalléen 21, N-0349 Oslo, Norway

Abstract. An important motivation for the object-oriented paradigm is to improve the changeability of the software, thereby reducing lifetime development costs. This paper describes the results of controlled experiments assessing the changeability of a given responsibility-driven (RD) design versus an alternative control-oriented "mainframe" (MF) design. According to Coad and Yourdon's OO design quality principles, the RD design represents a "good" design. The MF design represents a "bad" design. To investigate which of the designs have better changeability, we conducted two controlled experiments—a pilot experiment and a main experiment. In both experiments, the subjects were divided in two groups in which the individuals designed, coded and tested several identical changes on one of the two design alternatives.

The results clearly indicate that the "good" RD design requires significantly more change effort for the given set of changes than the alternative "bad" MF design. This difference in change effort is primarily due to the difference in effort required to *understand* how to solve the change tasks. Consequently, reducing class-level coupling and increasing class cohesion may actually *increase* the cognitive complexity of a design. With regards to correctness and learning curve, we found no significant differences between the two designs. However, we found that structural attributes change less for the RD design than for the MF design. Thus, the RD design may be less prone to structural deterioration. A challenging issue raised in this paper is therefore the tradeoff between change effort and structural stability.

Keywords: Changeability, change effort, object-oriented design quality, controlled experiment, responsibility-driven design.

1. Introduction

Handling *change* is a fundamental problem in software engineering. Object orientation provides design mechanisms such as encapsulation and inheritance to delegate functional responsibilities into logically cohesive classes, which in turn may improve understandability. The same mechanisms may be used to reduce coupling between the classes, hence reducing the impact of changes. Still, when designing object-oriented software, designers are faced with tradeoffs between various quality characteristics, such as changeability, correctness, performance and reusability. For example, to achieve high performance one may have to take "short cuts" that may lead to more error-prone software. Furthermore, changeability may be compromised

* Also at: Industrial Systems Development, Department of Informatics, University of Oslo, PO Box 1080 Blindern, N-0316 Oslo, Norway

by designing reusable, but perhaps overly complex classes or components. Deciding the "optimal" tradeoffs between such quality characteristics is clearly a difficult task. Before one can design object-oriented software with these tradeoffs in mind, one needs to understand how different design solutions affect the various software qualities. Unfortunately, the current state of practice indicates that the design of object-oriented software is still more of an art than a science (Briand et al., 1999b).

The changeability of a software system characterizes the ease of implementing changes. The motivation for studying changeability is related to our empirical research on evolutionary development of object-oriented software (Arisholm et al., 1998, 1999). In a former study (Arisholm et al., 1999), we found that as much as 60% of the coding effort was rework, i.e., changes due to evolving requirements and other incremental improvements of the software system. Furthermore, the continuous changes prescribed by evolutionary development may result in early, structural deterioration (Lehman and Belady, 1985), which in turn may cause changeability decay and necessitate costly restructuring efforts (Arisholm and Sjøberg, 2000). Thus, the design of an open-ended object-oriented structure that easily supports change is critical. This paper attempts to improve our understanding of the changeability of object-oriented software by studying the impact of identical changes on two alternative designs.

The remainder of this paper is organized as follows. Section 2 describes the design of the study, including the chosen design alternatives, the change tasks and the dependent variables. Section 3 describes the results of the pilot experiment used to formulate the hypotheses. Section 4 describes the results of the main experiment. Section 5 summarizes the results, discusses validity issues and relates the results to existing research. Section 6 describes future work.

2. Design of the Study

An important goal of our research is to investigate how design characteristics affect the changeability of object-oriented software. However, changeability can also be affected by other characteristics of the software, such as programming style and the quality of documentation. Thus, to study how design decisions affect changeability, it is necessary to restrict our study to software systems where only software characteristics directly related to the structural attributes (e.g., coupling, class count) of the software are varied. In this study, both systems implemented the same functionality and had similar programming style, naming conventions and documentation. Subjects of similar skill level designed, coded and tested a given set of changes on one of the two alternative software designs. The study consisted of two experiments:

1. the pilot experiment—to evaluate experimental design and material, and formulate the hypotheses, and

2. the main experiment—to replicate the pilot experiment with different subjects and test formal hypotheses on a larger scale.

This section describes common aspects of the two experiments (e.g., the coffee machine design alternatives, experience questionnaire, calibration task, change tasks, the change task questionnaires and the dependent variables). The results are described in Sections 4 and 5, respectively.

2.1. Treatments: The Coffee Machine Design Problem

We wanted to find alternative designs of the same system, in which one alternative adhered to Coad and Yourdon's quality design principles (the "good" design) and one did not (the "bad" design). The coffee machine designs seemed to be good candidates for the experiment. These designs have been discussed at a workshop on object-oriented design quality at OOPSLA'97 and are described in two articles in C/C++ User's Journal (Cockburn, 1998):

> *This two-article series presents a problem I use both to teach and test OO design. It is a simple but rich problem, strong on "design," minimizing language, tool, and even inheritance concerns. The problem represents a realistic work situation, where circumstances change regularly. It provides a good touch point for discussions of even fairly subtle designs in even very large systems...* (Cockburn, 1998).

The initial problem statement was as follows:

> *You and I are contractors who just won a bid to design a custom coffee vending machine for the employees of Acme Fijet Works to use. Arnold, the owner of Acme Fijet Works, like the common software designer, eschews standard solutions. He wants his own, custom design. He is, however, a cheapskate. Arnold tells us he wants a simple machine. All he wants is a machine that serves coffee for 35 cents, with or without sugar and creamer. That's all. He expects us to be able to put this little machine together quickly and for little cost. We get together and decide there will be a coin slot and coin return, coin return button, and four other buttons: black, white, black with sugar, and white with sugar* (Cockburn, 1998).

2.1.1. Description of the Design Alternatives

The Main Frame Design

According to (Cockburn, 1998), the type of design that most students come up with when faced with the problem of designing the given coffee machine software is a so-called mainframe (MF) design. The MF design, adapted from "Design 3" in (Cockburn, 1998), consists of seven classes:

- CoffeeMachine. Initiates the machine, knows about the hardware components.
- CashBox. Knows amount of money put in; gives change; answers whether a given amount of credit is available.
- FrontPanel. Captures selection; knows price of selections, and materials needed for each; asks CashBox if enough money has been put in; knows how to talk to the dispensers.
- Dispensers (cup, coffee powder, sugar, creamer, water). Knows how to dispense a fixed amount; knows when it is empty.
- Output. Knows how to display text to the user.
- Input. Knows how to receive command-line input from the user.
- Main. Initializes the program.

The Responsibility-Driven Design

The alternative responsibility-driven (RD) design was a result of a restructuring effort after the "customer" had requested several changes to the coffee machine. For example, the coffee machine was extended to make bouillon. For this reason, we thought that the restructured, RD design would be an interesting design alternative to compare with the initial MF design. The RD design, adapted from "Design 4" in (Cockburn, 1998), consists of 12 classes:

- CoffeeMachine. Knows how the machine is put together; handles input.
- CashBox. Knows how much credit is available; handles money.
- FrontPanel. Knows products and selection; coordinates payment and drink making; knows the price of coffee.
- ProductRegister. Knows what products are available.
- Product. Knows its recipe.
- Recipe. Tells dispensers to dispense ingredients in sequence.
- DispenserRegister. Acts as a librarian for the dispensers; controls nothing.
- Dispenser. Controls dispensing; tracks amount it has left.
- Ingredient. Knows its name only.
- Output. Knows how to display text to the user.
- Input. Knows how to receive command-line input from the user.
- Main. Initializes the program.

Message sequence charts of the main functional scenario for the two designs were given to help clarify the flow of messages between the objects of the designs (Appendix D). The two designs were coded using similar coding style, naming conventions and amount of comments. Variable names and method names were long and reasonably descriptive. Two small code fragments from the MF and RD designs are given in Appendix E.

2.1.2. Comparing the Designs Against Coad and Yourdon's Quality Principles

According to Coad and Yourdon's design principles (Coad and Yourdon, 1991a; Coad and Yourdon, 1991b), a "good" design adheres to (among others) the following guidelines, based on (Briand et al., 1999a):

Coupling. Interaction coupling between classes should be kept low, by decreasing the number of messages that can be sent and received by an individual object.

Cohesion. A class should carry out one, and only one, function. The attributes and services should be highly cohesive, i.e., they should all be descriptive of the responsibility of the class.

Clarity of design. The names in the model should closely correspond to the names of the concepts being modeled. Second, the responsibilities of a class should be clearly defined and adhered to. Furthermore, the responsibilities of any class should be limited in scope.

Keeping objects and classes simple. First, avoid excessive number of attributes in a class. A class should map to a type of entity in the problem description.

Although these design principles require a certain degree of subjective interpretation (Briand et al., 1999a), clearly the RD coffee machine design adheres significantly better to these design principles than does the MF design. The RD design has lower class-level coupling, more cohesive classes, better clarity of design and simpler classes. The MF design is assessed as follows:

Although the trajectory of change in the MF approach involves only one object, people soon become terrified of touching it. Any oversight in the MF object (even a typo!) means potential damage to many modules, with endless testing and unpredictable bugs. Those readers who have done system maintenance or legacy system replacement will recognize that almost every large system ends up with such a module. They will affirm what sort of a nightmare it becomes (Cockburn, 1998).

Furthermore, Cockburn assessed the RD design as follows:

The design we come up with at this point bears no resemblance to our original design. It is, I am happy to see, robust with respect to change, and it is a much more reasonable "model of the world." For the first time, we see the term *"product" show up in the design, as well as "recipe" and "ingredient." The responsibilities are quite evenly distributed. Each component has a single primary purpose in life; we have avoided piling responsibilities together. The names of the components match the responsibilities* (Cockburn, 1998).

For this experiment, we had to do certain modifications to the designs presented by Cockburn, so that they delivered the same functionality. Primarily, we removed the modifications done on the RD design in order to make bouillon, since we thought this functionality to be a particularly good candidate for a change task. This also motivated leaving the price of coffee where it was originally—in the front panel class. However, it would be a simple task to move the price attribute from the front panel class to the product class in order to provide differentiated pricing (to make bouillon). Otherwise, the main concepts underlying the two designs have been kept as far as possible. Although the modified RD design may represent a slightly less "pure" design than the one presented by Cockburn, we believe his assessment is also applicable to the MF and RD design alternatives.

2.1.3. Structural Attributes of the Design Alternatives

Table 1 shows the values of coupling (*OMMIC_N*, *OMMIC_L* and *OMMEC*) and size (*MC* and *CS*) for the two designs. The RD design has about 40% lower class-level coupling to non-library classes (*OMMIC_N* and *OMMEC*). *OMMIC_L* quantifies the number of method invocations to library classes, which in this case are *String* and *Vector*. Because the RD design uses vectors to represent products and dispensers, the class level *OMMIC_L* measure is slightly higher for the RD design (mean = 1.3) than for the MF design (mean = 1.1). The MF design has larger classes and fewer methods per class than the RD design.

Table 1. Descriptive statistics of structure and size attributes for the MF and RD designs.

Measure	Description	Design	Median	Mean	Sum
OMMIC_N(c)	The number of static method invocations from a (client) class *c* to non-library classes	MF	2	4.7	33
		RD	1	2.8	34
OMMIC_L(c)	The number of static method invocations from a (client) class *c* to library classes	MF	0	1.1	8
		RD	0	1.3	16
OMMEC(c)	The number of static method invocations to a (server) class *c*	MF	3	4.7	33
		RD	2	2.8	34
MC(c)	The number of implemented methods in a class *c*	MF	1	1.6	11
		RD	2	1.8	22
CS(c)	The size (in SLOC) of each class Note that the sum corresponds to system size	MF	9	11.0	77
		RD	7	8.9	107

At the *system* level, however, Table 1 (the "Sum" column) shows that the overall non-library coupling remains almost unchanged, whereas the coupling to library classes, the total number of methods and the total system size have *increased* for the RD design. Thus, to (1) reduce coupling, (2) increase cohesion, (3) improve the clarity of the design, and (4) keeping classes simple, the values for some system-level measures have increased.

2.2. The Mocca Programming Language

We had to make some decisions regarding the choice of programming language. It should be easy to understand for subjects with some prior experience with common OO programming languages—to minimize the learning curve. However, the programming language should also contain sufficient OO constructs and flexibility to allow the development of realistic code for the change tasks. Thus, we created a scaled-down version of Java, called *Mocca*. Mocca has the same syntax as Java, but is restricted in several ways:

- It does not contain inheritance mechanisms or constructs for interfaces.
- It has no explicit type casting.
- It has a globally available (static) INPUT and OUTPUT class.
- It contains only the elementary types void, int and boolean.
- It contains only two library classes: String and Vector.

A relatively complete documentation of the Mocca language was written in eight pages. While the restrictions in Mocca may be too limiting as a general purpose, experimental OO programming language, we believe that Mocca was a reasonable tradeoff between realism and simplicity for the given change tasks on the coffee machine designs.

2.3. The Programming Tasks

The programming tasks consisted of one *calibration task* and three *change tasks* (*c*1, *c*2 and *c*3) for the coffee machine. For practical reasons, the changes were coded with pen and paper. For small designs and change tasks, we believe this may be a better choice than using a computer to reduce the possibility of errors caused by technical- and tool-oriented problems. Each task description contained a test case that each subject used to manually "test" the solution. Of course, this is not a real test, which would require running the program on a computer. The main purpose of the test was to motivate the subjects to produce solutions of good quality before starting on the next change task. Judging from the actual correctness score of the solutions (Table 6), this strategy seems to have worked quite well.

2.4. The Calibration Task

The first programming task to be completed by all subjects was the calibration task. The calibration task consisted of adding transaction log functionality in an automatic teller machine, and was not related to the coffee machine designs. Since all subjects implemented the same change on the same design, the calibration task provided a common baseline for comparing the programming skill level of the subjects. The calibration task was almost the same size as the change tasks *c*1, *c*2 and *c*3 combined. The size of the calibration task ensured that most aspects of the Mocca programming language (e.g., class constructors, vectors, strings, input and output) had been exercised, thus reducing the influence of the programming language learning curve.

2.4.1. The Change Tasks

Each change task was coded by the students directly on the coffee machine code printout for the given design. The change tasks consisted of three changes to the coffee machine, to be implemented in the given order. The actual change task descriptions are given in Appendix B.

(*c*1) *Implement a coin return button*. The actual solution was identical for the RD and the MF design. In both cases, it involved a modification to the menu handling routine (to include the "Return Coins" menu choice) and the addition of corresponding event handling routine. In addition, the developers had to call the "ReturnCoins" method in the CashBox class.

(*c*2) *Make bouillon*. Extend the machine with a menu choice and the functionality to make bouillon in addition to coffee. Bouillon costs more than coffee. The solution involved making a menu choice and event handling routine for bouillon by modifying the front panel. It also involved making a new dispenser for bouillon, and checking whether the customer had deposited sufficient funds.

(*c*3) *Fix a bug: Check whether all ingredients are available for the selected drink*. If one or more dispensers are empty, the user should get an error message and can try another drink or get his money back. This change task was motivated by a "bug" found in both of the code listings for the original design alternatives presented in (Cockburn, 1998). If the machine was empty of a required ingredient (e.g., creamer), the machine would still produce the "drink" using only the remainder of the ingredients, i.e., the customer would receive black coffee when asking for white coffee. The solution involved checking whether all required ingredients were available before making the drink.

For subjects that managed to complete all change tasks within the allocated time, an "extra assignment" was given. This change task was included to ensure that none of the subjects finished before the end of the allocated time of the experiment. We did not want the subjects to leave early, disturbing the other subjects. Furthermore, without change task *c*4, subjects may have been inclined to use more time on change task *c*3 than they would otherwise. Change task *c*4 is not included in any subsequent analysis, since only very few subjects managed to complete the task:

(*c*4) *Add the option "make your own drink,"* by selecting among any meaningful combination of the available ingredients.

2.4.2. The Change Task Questionnaire

After completing each programming task (including the calibration task), the participants reported the effort to understand, code and test each task. In addition, the subjects reported on subjective task difficulty, solution strategy (explorative or systematic) and confidence in the correctness of their solution. The questionnaire is given in Appendix C.

2.5. Experimental Design

To ensure accurate and reliable results we had to deal with issues related to the learning curve and the skill level of the individuals who participated in the experiment. To control for the differences in skill level of the individuals, we considered a cross-over design where each developer implements the same (or similar) changes on both design alternatives. However, this experimental design does not control for the following learning effects:

- *Learning the system*—if the design alternatives have many similarities, most of the developer's initial system comprehension effort will be spent on the first design.
- *Learning the changes*—if a developer implements the same change on two alternative designs, it is likely that the developer will be more efficient during implementation of the change on the second design.

Thus, we used a design where each developer implements the same change only once, while still controlling for the differences in individual skill levels by assigning the developers in two groups by means of randomization and blocking. This is described further in the following sections.

2.5.1. Design of the Pilot Experiment

The subjects consisted of 12 graduate students and professionals enrolled in a course in software process improvement taught by one of the authors. The experiment was divided in three separate, 1-h sessions consisting of:

(Session 1) *Experience level assessment and training*. During this session, the students completed the experience questionnaire (Appendix A). Then, we trained the subjects in Mocca and distributed the programming language documentation.

(Session 2) *Skill level assessment and group assignment*. All students implemented the calibration task and completed a change task questionnaire (Appendix C). Based on the results of the calibration task, the students were divided into blocks and then assigned at random (within each block) into two groups, one for each design alternative.

(Session 3) *Coffee machine experiment*. The subjects to the first group implemented the change tasks on the MF design. The subjects assigned to the other group implemented the change tasks on the RD design.

2.5.2. Design of the Main Experiment

Subjects of the main experiment were mainly undergraduate students in computer science at University of Oslo. Unlike the pilot experiment, the subjects volunteered for the experiment and were paid to participate. The experiment took place within one 3-h session. The subjects were introduced to the experimental procedures and trained in Mocca during the first hour. During the next 2 h, the subjects first implemented the calibration task and then the change tasks.

With regards to the group assignment, we were unable to use blocking based on the calibration task because the experiment consisted of only one session. Furthermore, results from the pilot experiment suggested that it was not useful to use the reported experience level data (Appendix A) to create blocks. Consequently, students were assigned at random into two groups of equal size, one for each design alternative. Some students did not show up for the experiment, while some other students that had failed to register for the experiment showed up just prior to the session. They were assigned at random when they arrived. The resulting group assignment consisted of 17 subjects on the MF design and 19 subjects on the RD design. In the unlikely event that the randomization would fail to provide approximately equal groups, we could use the results from the calibration task in subsequent analyses to adjust for such differences (Section 5.2.3).

2.6. Dependent Variables

Figure 1 depicts the dependent variables of the study. They are explained further in the following sections.

2.6.1. Change Effort

Before starting on a task, the subjects wrote down the current time. When the subjects had completed the task, they reported the total effort (in minutes) to

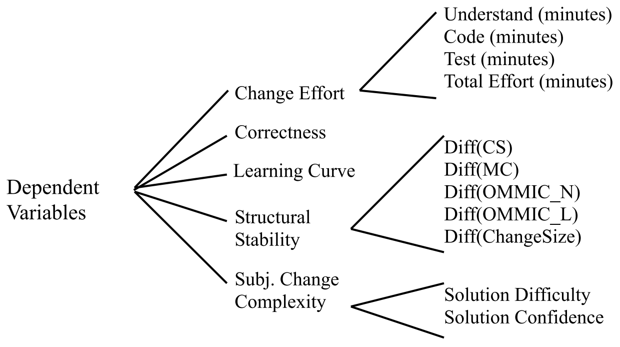

Figure 1. Summary of the dependent variables of the study.

complete the change task. The primary dependent variable of the study was the combined total effort to complete all change tasks. Furthermore, the total effort was divided in the effort to

- *understand* the task (analysis and design of the solution),
- *code* the solution, and
- "*test*" the paper solution against the test case.

2.6.2. Correctness

It is possible that one design is more error-prone than another design, resulting in errors that are not discovered and subsequently corrected by each subject in the experiment. Furthermore, the reported change effort for the change tasks may contain lower values for solutions that contain errors. This may bias the change effort results if one design is more error-prone than the other design. Consequently, each change task solution was reviewed and given a (subjective) correctness score by the first author of this paper. Table 2 gives the coding scheme.

2.6.3. Learning Curve

The experiment contains only a small number of change tasks. Thus, the recorded *total* change effort for the combined change tasks may fail to reflect trends in change effort caused by the system learning curve. For example, a given design may be difficult to change until the developer understands the intricate structural properties and abstraction mechanisms of the design. However, the subsequent changes may be easy to implement, hence resulting in a trend towards less change effort compared with another design.

To perform statistical tests on differences in the learning curve we need to quantify it such that the measure is normalized and hence comparable for different subjects.

Table 2. Coding scheme of the correctness measure.

Correctness score	Interpretation
6	Correct solution, passes the test case
5	Small deviations from the test case description, but no logical errors
4	Small logical errors that are estimated to be very simple to fix
3	Some errors that are estimated to take some time to fix
2	Incomplete solution that are estimated to take a long time to fix
1	Very incomplete solution

We measure the learning curve with respect to the given change tasks as the normalized difference in effort to understand the last change ($c3$) versus the first change ($c1$) for each subject, for design RD and MF, respectively:

$$\text{LearningCurve }(d) = \frac{\text{Understand}(d, c1) - \text{Understand}(d, c3)}{\text{Understand}(d, c1) + \text{Understand}(d, c3)}, \quad d \in \{\text{RD}, \text{MF}\}$$

A larger number indicates a stronger learning effect. The measure is not meaningful as an absolute measure of the learning curve since change task $c3$ is probably more difficult to solve than change task $c1$. Thus, in this case one may even get "negative" learning. The measure is only meaningful when comparing the relative *difference* in the learning curve on MF versus RD. The measure also assumes that most of the learning occurs early.

2.6.4. Subjective Change Complexity

We also asked the subject two questions that may reflect the perceived complexity of each change task:

- *Solution difficulty*—how difficult did the subjects think it was to solve each change task (1 = very simple; 6 = very difficult).
- *Solution confidence*—how confident were the subjects that the solution of a change task did not contain serious errors (1 = very unsure; 6 = very confident).

2.6.5. Structural Stability

When studying the changeability of an object-oriented design, it may be appropriate to assess the impact changes have on the design. Consider Lehman and Belady's "law of increasing complexity":

> *As a large program is continuously changed, its complexity, which reflects deteriorating structure, increases unless work is done to maintain or reduce it* (Lehman and Belady, 1985).

Table 3. Summary of structural stability measures.

Definition	Detailed explanation
Diff(CS(c))	The difference in average class size before and after a change task
Diff(MC(c))	The difference in average number of implemented methods in a class *c*
Diff(OMMIC_N(c))	The difference in the average number of static method invocations for a class *c*
Diff(OMMIC_L(c))	to non-library classes (OMMIC_N) and to library classes (OMMIC_L)
ChangeSize	SLOC added + deleted + modified for each change task

In general, changes in structural attributes do not necessarily indicate decay; there could be restructuring or re-engineering going on. However, in the coffee machine experiment, there is no "restructuring" of the design. The change tasks represent functional additions (*c*1 and *c*2) and bug fixes (*c*3). Thus, when assessing the changeability of design alternatives it may also be appropriate to measure trends in structural attributes (i.e., structural stability) that may indicate changeability decay (Arisholm and Sjøberg, 2000). The *differences* in the average values of the measures before and after each change (e.g., from change task *c*1 to change task *c*2) may be used to assess whether the structural attributes of one design change faster than an alternative design.

A summary of the structural stability measures is given in Table 3. To measure the structural change of the solutions, five paper solutions were selected at random for each of the two design alternatives, for a total of 10 solutions to each of the change tasks. To ensure accurate structural attribute measures, only solutions with a correctness score of five or six (i.e., "correct" solutions) for all three change tasks were considered. The selected paper solutions were coded into a computer by one of the authors, and subsequently compiled and tested to ensure that the solutions actually were correctly implemented. A Java parser was used to collect the measures. In addition to the change in structural attributes, the size of each change was calculated. For the *ChangeSize* measure, we manually counted (based on the paper solutions) the number of lines of code added, deleted or modified for *all* solutions that were correctly implemented.

3. Results of the Pilot Experiment

The goals of the pilot experiment were

- To evaluate and improve the quality of the experimental materials (e.g., questionnaires, change tasks and programming language). This is described further in conjunction with analysis of threats (Section 5.2).
- To evaluate the usefulness of different blocking strategies to reduce random errors, i.e., blocking on the results from the calibration task and blocking based on data from the experience level questionnaires.
- To formulate hypotheses and to develop meaningful dependent variables through an exploratory analysis of the preliminary results from the pilot experiment.

3.1. Evaluation of Blocking Strategies

The pilot experiment used the correctness score of the calibration task to create blocks on skill level. An equal number of subjects were assigned at random to each design (MF and RD) from each block. Unfortunately, only eight of the 12 subjects attended session 3. This resulted in that the average skill level was slightly higher for subjects assigned to the RD design, despite the randomized block scheme (Figure 2). Clearly, the usefulness of blocking may be limited unless one can be sure that the assigned subjects will attend the experiment.

We also evaluated whether data from the experience level questionnaire could be used to create blocks for the main experiment. We found no significant correlation between the experience level data and the results of the calibration task in the pilot experiment. This suggests that it may be ineffective to use the experience level data to create a randomized block design.

3.2. Preliminary Assessment of Change Effort for MF and RD

Figure 3 depicts the difference in total effort to change tasks *c*1 and *c*2 (most subjects did not have time to complete *c*3 within the 1-h session of the pilot experiment). Although subjects assigned to design RD had performed better on the calibration task in the previous session, they still needed on average 30% more time to complete change tasks *c*1 and *c*2 than the subjects assigned to design MF.

4. Results of the Main Experiment

The explorative analysis of the results from the pilot experiment was used to formulate the hypotheses of the main experiment. According to design principles such as Coad and Yourdon's, we would expect that the RD design enables a more efficient and correct implementation of changes. However, the results of the pilot experiment

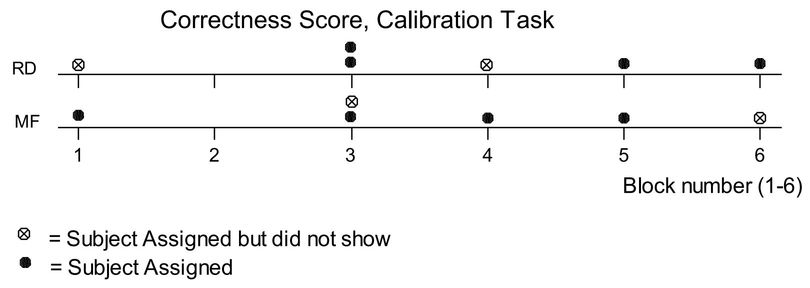

Figure 2. Dot-plot of group assignment with the correctness score (1–6) from the calibration task as the blocking factor.

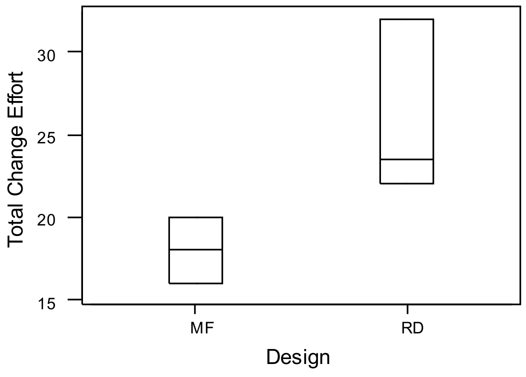

Figure 3. Box-plot of the total change effort (in minutes) to complete change tasks *c*1 and *c*2 for design MF and RD, respectively. A line is drawn across the box at the median. The box represents the 95% confidence interval (CI) for the median.

do not support this theory. The theory underlying the formulation of the hypotheses is that the more fine-grained delegation of responsibilities of the RD design results in added complexity that more than outweighs its theoretical advantages with regards to change effort and correctness (H1–H4). However, the improved delegation of responsibilities of the RD design should result in a more stable design (H5).

Formal Hypotheses

(H1) *Change effort*: The RD design requires more change effort than the MF design.

(H2) *Learning curve*: The RD design has a stronger learning effect than the MF design. We regard H2 as a validity check on H1. If both H1 and H2 are accepted, it will be difficult to determine whether H1 would have been valid if we had included even more change tasks.

(H3) *Correctness*: The solutions for the RD design contain more errors than the MF design.

(H4) *Change complexity*: The RD design has higher change complexity than the MF design.

(H5) *Structural stability*: The RD design has better structural stability than the MF design.

The statistical tests will attempt to reject the null hypotheses, which are just the opposites of H1–H5. For H1 and H2, a one-sided two-sample *t*-test (assuming unequal variances) on the difference in means was used. Before using the *t*-tests, the samples were checked for normality using the χ^2 based Kolmogorov–Smirnov normality test. No significant deviations from the normal distribution were found. For H3 and H4, the tests were performed using Mood's median test, which is a robust, non-parametric sign scores test for ordinal scale measures such as the correctness score and subjective task difficulty. H5 was not tested formally, but was assessed based on a subjective interpretation of the results.

The more tests are performed on the same dataset, the more likely is it that one will find significant results occurring by chance. Thus, to make a scientific statement with a reasonable degree of confidence, the significance level for the hypotheses tests were initially set to $\alpha = 0.1$, and subsequently reduced to account for multiplicity using Holms multiple test procedure (Holm, 1979). Holm has shown that, for K statistical tests, the adjusted significance level must be set equal to $\alpha/(K - i + 1)$, where $(i = 1, \ldots, K)$ is the index of each test ordered by the p-value $(p_1 \leq p_2 \leq \cdots \leq p_K)$. This means that p_1, the smallest p-value, must be compared with $\alpha_1 = \alpha/K$. The largest p-value p_K must be compared with $\alpha_K = \alpha$. In our case, adjusting the significance level using Holms procedure is more appropriate than using the even more conservative Bonferroni adjustment, i.e., α/K, since the Bonferroni adjustment ignores the correlation between tests. A practical discussion of the power of tests and presetting the level of significance is provided in (Briand et al., 1999a).

4.1. Change Effort (H1)

*Hypothesis H*1 *is supported* ($p = 0.0072$, two-sample t-test on the difference in mean total change effort to implement $c1$, $c2$ and $c3$). On average, the total change effort on RD was about 20% higher than the total effort on MF (Table 4). Most of the difference in change effort is due to differences in time to understand how to implement the change tasks ($p = 0.0006$). There are smaller differences in the coding

Table 4. Summary of change effort (in minutes) for the change tasks $c1$, $c2$ and $c3$.

Changeability indicator	Group	N	Mean	StDev	SE Mean	H1: μ(MF) < μ(RD) (p-value)	Holms α = 0.1/(15-i+1)
Total $c1+c2$	MF	17	26.88	8.28	2.0	**0.0004**	0.0067
	RD	19	38.30	10.20	2.3		(i=1)
Total $c1+c2+c3$	MF	16	49.20	12.60	3.1	**0.0072**	0.0083
(H1)	RD	18	59.22	9.29	2.2		(i=4)
Understand	MF	16	16.03	7.31	1.8	**0.0006**	0.0071
$c1+c2+c3$	RD	17	26.06	8.88	2.2		(i=2)
Code $c1+c2+c3$	MF	16	27.13	9.26	2.3	0.42	0.0143
	RD	15	27.77	7.95	2.1		(i=9)
Test $c1+c2+c3$	MF	16	6.09	4.07	1.0	0.43	0.0200
	RD	14	6.36	3.77	1.0		(i=11)

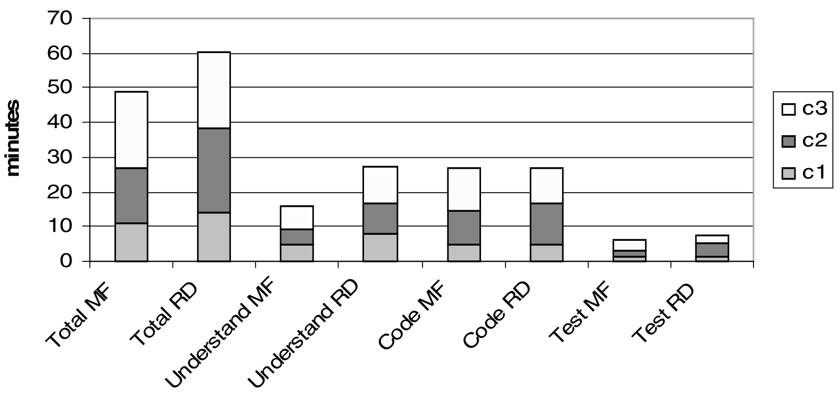

Figure 4. Average change effort (in minutes) for each change task ($c1$, $c2$, $c3$) for MF and RD.

and testing effort, but all results are in favor of the MF design. Figure 4 depicts the average change effort for the change tasks.

Only eight out of 19 subjects assigned to the RD design reported that they completely finished change task $c3$, whereas 16 out of 17 subjects assigned to the MF design completed change task $c3$. The analysis of the effort data on the RD design therefore includes data points for those subjects that *almost* finished, i.e., some testing remained but they still reported effort data. All subjects completed the first two change tasks. When only counting change tasks $c1$ and $c2$, the results show more than 40% difference in change effort ($p = 0.0004$). Based on the results of the pilot experiment, we had estimated that more subjects would have completed all change tasks within the allocated time.[1] In retrospect, we should have allocated more time for the change tasks.

4.2. Learning Curve (H2)

The hypothesis H2 is not supported. The results indicate that there is no significant difference in the relative learning effect for designs MF and RD (Table 5). The average values for the effort to understand each change is shown in Figure 5. The results show that the time to understand each change task on MF is lower than the time to understand the same tasks on RD. Furthermore, there is no visible difference in the trend in the learning curve (from $c1$ to $c3$).

Table 5. Two-sample t-test on difference in learning curve.

Changeability indicator	Group	N	Mean	StDev	SE Mean	H1: μ(RD) < μ(MF)	Holms $\alpha = 0.1/(15-i+1)$
Learning curve ($c1$, $c3$)	MF	16	−0.096	0.354	0.088	p=0.52	0.0250
	RD	17	−0.103	0.370	0.090		(i=12)

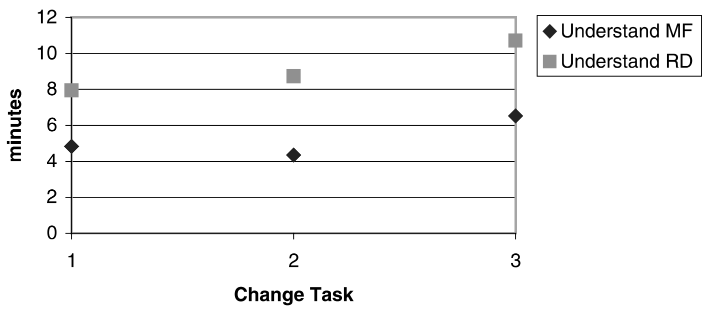

Figure 5. Trend in the comprehension effort from change task *c*1 to change task *c*3.

4.3. Correctness (H3)

*Hypothesis H*3 *is not supported.* There is no significant difference in correctness (Table 6). On average, the solutions had high quality. This suggests that the change effort results are reliable—they are not confounded by low correctness or differences in correctness.

4.4. Subjective Change Complexity (H4)

*Hypothesis H*4 *is partially supported.* There are no significant differences for change tasks *c*1 and *c*2 (Table 7). For change task *c*3, subjects assigned to the RD design were less confident about the correctness of the solution. There is some evidence that subjects assigned to the RD design also thought it was more difficult to solve change task *c*3 ($p = 0.033$). However, this result is not significant with respect to Holms adjusted α-value, which in this case requires a p-value less than 0.0091.

Table 6. Summary of Mood's median test on correctness on MF and RD.

Change-ability indicator	Popula-tion median	Group	N	Group median	$N<$	$N\geq$	χ^2	p-value	Holms α=0.1/(15-i+1)
Correctness *c*1	6	MF	17	6	4	13	0.04	0.847	0.1000
		RD	19	6	5	14			(i=15)
Correctness *c*2	4	MF	16	6	6	10	2.29	0.130	0.0111
		RD	19	4	12	7			(i=7)
Correctness *c*3	5	MF	16	5	9	7	0.32	0.571	0.0500
		RD	9	6	4	5			(i=14)

Table 7. Summary of Mood's median tests on subjective task difficulty.

Change-ability indicator	Population median	Group	N	Group median	$N<$	$N\geq$	χ^2	p-value	Holms $\alpha=0.1/(15-i+1)$
Subject task difficulty $c1$	1	MF	17	1	12	5	0.63	0.429	0.0167
		RD	19	1	11	8			(i=10)
Subject task difficulty $c2$	3	MF	17	2	10	7	1.74	0.187	0.0125
		RD	19	3	7	12			(i=8)
Subject task difficulty $c3$	3	MF	16	2	10	6	4.57	0.033	0.0091
		RD	16	3	4	12			(i=5)
Confidence $c1$	5	MF	17	5	11	6	0.34	0.559	0.0333
		RD	19	5	14	5			(i=13)
Confidence $c2$	4	MF	17	5	7	10	2.70	0.101	0.0100
		RD	19	4	13	6			(i=6)
Confidence $c3$	3	MF	16	4	3	13	15.2	**0.000**	0.0077
		RD	16	3	14	2			(i=3)

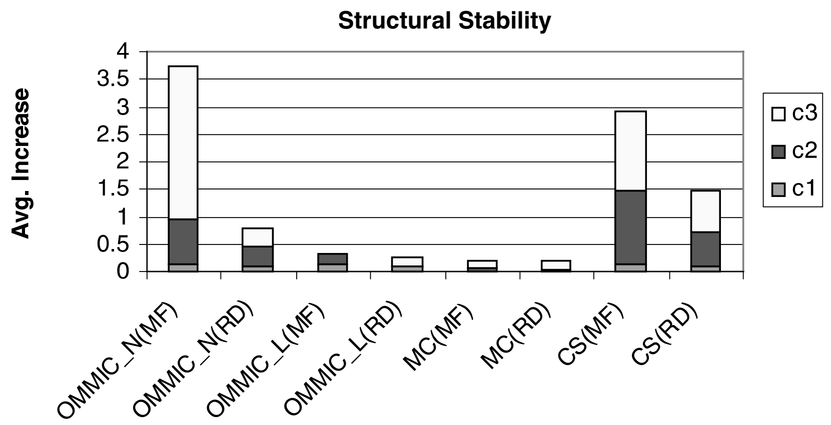

Figure 6. Changes in structural attribute measures of the MF and RD designs after implementing change task $c1$, $c2$ and $c3$. The data is based on 10 randomly selected solutions to the change tasks, five for the MF design and five for the RD design.

4.5. Structural Stability (H5)

Figure 6 suggests that the structure of the MF design is affected more than the RD design when subjected to the changes $c1$, $c2$ and $c3$. In particular, the average class size (*CS*) and the average import coupling to non-library classes (*OMMIC_N*) change much more for the MF design than for the RD design. The results also indicate that the RD design requires smaller changes (in SLOC added + delet-

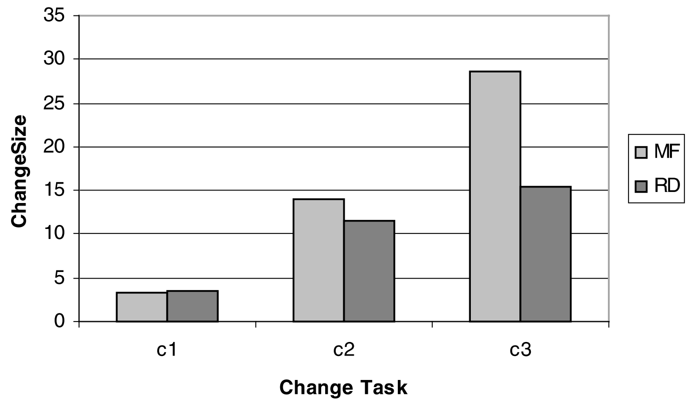

Figure 7. Comparison of the size of each change task for the MF and RD designs.

ed + modified) for the third change task (Figure 7). More interestingly, the changes in these measures are not reflected by corresponding changes in change effort, correctness and subjective change complexity. However, a qualitative assessment of the resulting MF design after changes *c*1, *c*2 and *c*3 suggests that the *FrontPanel* class of the MF design is indeed becoming a "maintenance nightmare", piling up with more and more responsibilities and high class-level coupling.

4.6. Attempting to Explain the Results

By investigating the delivered solutions and the comments given by the subjects on the change task questionnaires, we attempt to explain why there is a significant difference in change effort for the design alternatives.

One difference between the designs is size. The RD design is larger than the MF design (Table 1). Although size in general may be an important contributor to complexity, we do not believe that the difference in size is that important for the MF and RD designs. Both designs are "small". Furthermore, most of the difference in size is due to simple initialization code in the RD design (e.g., declaration of identifiers and construction of the objects). In our opinion, this initialization code is quite simple compared with the remainder of the RD design.

For change task *c*1 (the "return button"), the solution is *identical* for the two design alternatives, involving two small changes to the *CoffeeMachine* class after determining that the *CashBox* class already contains a *returnCoins* method. Still, it required less effort to understand how to solve *c*1 on the MF design compared with the effort to understand the same solution for the RD design. With regards to the RD design, and in particular for change tasks *c*2 and *c*3, we found comments such as "*I keep nesting through the classes, but it is too complicated—I give up*". Although the MF design has classes with higher coupling, the dynamic depth of the message interactions among classes to implement a given functional scenario is significantly smaller than for the RD design. Thus, it may be more difficult to perform a systematic trace of the RD design. The fact that the subjects had access to a message sequence chart (Appendix D) did not seem to help much. Furthermore, for change task *c*3, the RD design involved changing four classes whereas only two classes had to be changed in the MF design. The same change task was also reported to have higher subjective complexity for the RD design than for the MF design. Thus, we believe that the number of classes changed and the depth of the message interactions among classes are important contributors to the complexity of a design.

These results are supported by existing theory. The first mental representation programmers build to understand completely new code is a control flow abstraction of the program (von Mayrhauser et al., 1997). Some developers use a *systematic* approach (e.g., line by line) to build the control-flow abstraction. Other developers used a more *opportunistic* approach, studying code in an as-needed fashion based on hypotheses guided by clues in the code (von Mayrhauser et al., 1997). According to this theory, one may expect an increase in the time required to understand how to implement a change when the amount of collaboration between objects participating in the implementation of a functional scenario increases. Furthermore, the effect may be larger for programmers with a systematic approach.

To assess whether the solution approach affected the change effort for each change tasks, we performed a one-way analysis of variance using "solution approach" as the explanatory factor with three levels, and the total time for a given change task (e.g., *c*2) as the response variable. The solution approach was given by the subjects on the change task questionnaire (Appendix C), and coded as follows for the analysis.

1. Explorative ("trial and error"): Subject characterized solution approach as 1 or 2.
2. Mixed: Subject characterized solution approach as 3 or 4.
3. Systematic: Subject characterized solution approach as 5 or 6.

The solution approach does not necessarily correspond directly to "opportunistic" versus "systematic" program understanding using von Mayrhauser's terminology. However, it seems plausible that explorative programmers are also more "opportunistic" than the systematic programmers.

For change task *c*1 there was no difference in the change effort depending on the solution approach. The change task may be too small to uncover differences in change effort due to the solution approach. For the larger change task *c*2, the exploratory programmers were significantly faster than the systematic programmers on the RD design (Table 8). For change task *c*2, the exploratory programmers were actually slower than the systematic programmers on the MF design (Table 9), although the difference in change effort is not significant. For change task *c*3, the effect of solution approach was similar to change task *c*2. However, there were too few

Table 8. ANOVA for solution approach on change task *c*2 for the RD design.

```
Analysis of variance for total c2
Source       DF       SS       MS       F            P

Strategy      2      663.7   331.9   5.59         0.014
Error        16      950.3    59.4
Total        18     1614.0
                                     Individual 95% CIs For mean based
                                       on pooled StDev
Level         N       Mean    StDev  -------+---------+---------+---------
expl.         9     19.111    5.442    (----*-----)
mixed         8     25.875    9.015           (-----*-----)
syst.         2     38.500   12.021                 (-----------*--------)
                                     --------+--------+--------+--------
Pooled StDev =        7.707                  20        30        40
```

Table 9. ANOVA for solution approach on change task *c*2 for the MF design.

```
Analysis of variance for total c2
Source      DF      SS       MS     F                      P

Strategy     2      99.0    49.5   1.26                   0.314
Error       14     551.0    39.4
Total       16     650.0
                                 Individual 95% CIs For mean based on
                                 pooled StDev
Level        N      Mean    StDev ---------+---------+---------+-------
expl.        7    18.857    6.594                  (---------*---------)
mixed        7    14.286    5.851         (----------*---------)
syst.        3    13.333    6.506 (---------------*--------------)
                                  ---------+---------+---------+-------
Pooled StDev =       6.273                10.0      15.0      20.0
```

subjects completing the *c*3 task for the RD design to give reliable results. The results based on change tasks *c*1 and *c*2 can be summarized as follows.

- The solution approach may have a significant impact on the change effort.
- The effect of the solution approach on the change effort depends on the size of the change and on the design approach. The RD design seems to be better suited for an explorative solution approach than the MF design.

5. Summary of Results

In the coffee machine study, cohesion was effectively increased by splitting the "MF" class, and delegating some of its functional responsibilities to several smaller classes, resulting in the RD design. The RD design also has significantly lower class-level coupling. The RD design adhered better to Coad and Yourdon's design principles than the MF design. However, the RD design contained twice as many classes and slightly more code (in SLOC) compared with the initial, MF design. With respect to the given change tasks, the results can be summarized as follows:

- The RD design requires significantly (20–50%) more change effort than the alternative MF design.
- The RD design is harder to understand for the "average" programmer than the MF design, and the learning curve is not better for the RD design than for the MF design.
- The RD design does not result in fewer errors than the MF design.
- The RD design may have higher structural stability than the MF design.

Although one must be careful when generalizing results based on a single study, the results indicate that using delegation of responsibilities to reduce class-level coupling and increase class cohesion may not necessarily improve the changeability of a design. On the contrary, the resulting structure may contain deeply nested class interactions, which average programmers may find difficult to understand. Thus, decreasing coupling and increasing cohesion may *increase* complexity and the costs of changes.

The only indicator that shows potential benefits of the RD design is regarding structural stability. In the RD design, new functionality is divided among the collaborating classes. In the MF design, most subjects piled the code onto the already overloaded "MF" class. However, what are the practical consequences of the potentially increased stability? In our study, the changes in structural attributes are not reflected by external quality attributes (e.g., increased change effort and decreased correctness). Thus, when does the added "stability" of the RD design justify the increased complexity and costs of changes? For the coffee machine, it is difficult to envision a sufficient number of future changes to justify the more complex RD design. The MF approach works well for the types of changes likely to occur. Furthermore, we believe that a prerequisite for achieving the potential advantage of the RD design is that the programmers are confident with the design, and understand the abstract delegations of responsibilities so that they do not break the underlying structure. Our results indicate that this understanding may be difficult to achieve for the average programmer.

5.1. Comparing the Results with Related Research

The current state of research in object-oriented design quality often concludes, based on empirical data, that classes with high coupling and/or low cohesion are less error-prone, easier to maintain, etc. There is a growing body of results indicating that measures of structural attributes such as coupling, cohesion, inheritance depth, etc.

can be reasonably good predictors of development effort and product quality (Basili et al., 1996; Briand et al., 1999c; Briand et al., 1999d; Chidamber and Kemerer, 1994; Daly et al., 1996; Li and Henry, 1993). Thus, it seems conceivable that such measures can be used to compare the changeability of alternative designs. However, for practical reasons, many of these studies have validated the measures by comparing *different* systems or different classes within one system. For example, in (Briand et al., 1997; Briand et al., 1999a), they investigated whether a "good" design (adhering to Coad and Yourdon's design principles) was easier to maintain than a "bad" design. The results strongly suggest that Coad and Yourdon's design principles have a beneficial effect on the maintainability of object-oriented designs. However, as pointed out by the authors, the designs represented two different systems—a temperature controlling system and an automatic bank teller machine, respectively. Thus, it is difficult to determine what the practical consequences of the results are. For example, *how* can coupling be reduced without increasing other attributes that also contribute to the complexity of the software? In this paper, we compare alternative designs of the *same* system. While this approach introduces new problems (e.g., group assignments, how not to bias the designs and the change tasks), it enables us to assess how different design tradeoffs of the same system actually affect the overall complexity.

In (Sharble and Cohen, 1993), one of the few experiments comparing alternative OO technologies was reported. The authors conducted an experiment where they compared a data-driven and a RD design method. Two systems were developed based on the same requirement specification—using the data-driven and the RD design method, respectively. Structural attribute measures of the two systems were collected and compared. Based on the measured values, the authors suggested that RD design produced higher quality software than data-driven design, because the RD method resulted in designs with less coupling and higher cohesion than the data-driven method. We believe it may be premature to draw such conclusions. Whether the design measures used in the experiment actually measured "quality" was not empirically validated. In other words, the experiment did not involve any direct measurement of external quality attributes.

The combined results of (Briand et al., 1997; Briand et al., 1999a; Sharble and Cohen, 1993) and the results presented in this paper can be summarized as follows:

- A system adhering to Coad and Yourdon's design quality principles is easier to maintain than *another* system not adhering to those principles (Briand et al., 1997; Briand et al., 1999a).
- RD design may result in lower coupling between classes and higher class cohesion (Sharble and Cohen, 1993).
- However, a practical concern is *how* to adhere to design quality principles such as those proposed by Coad and Yourdon. Reducing coupling and increasing cohesion of the *same* system may result in changing other aspects of the design that contribute to an *increase* in system complexity.

5.2. Threats to Validity

The external validity of this study depends on, for example, the choice of design alternatives, the choice of change tasks, and how representative the sample is of the population. We cannot eliminate these threats within the context of this study. The ultimate means to improve the validity of the study is by replication, using other subjects, other design alternatives and other change tasks. The main experiment may be viewed as a replication of the pilot experiment with different population samples and group assignments. However, it may be more important to use different design alternatives and other change tasks. In addition, internal validity may be threatened by for example skewed group assignments, ambiguous questions and otherwise unclear experimental materials. This is elaborated in the following sections.

5.2.1. Experimental Materials

Doing pilot experiment effectively results in "throwing away" data, but such an investment may significantly reduce threats and, hence, improve the validity of the study. For example, we used the pilot experiment to evaluate and improve the quality of the experimental materials before the main experiment took place.

Mocca

With regards to the Mocca programming language, the subjects of the pilot experiment reported that Mocca was very easy to learn. Two of the subjects had problems understanding how to program in Mocca, but they had no previous experience with object-oriented programming. Furthermore, informal discussions with the subjects indicate that Mocca did not restrict their choice of solutions for the given change tasks on the coffee machine designs. Among the subjects of the main experiment, all subjects had previous experience with Java and similar OO programming languages.

Written Materials

The evaluation of the pilot experiment resulted in important improvements of the design descriptions and the change task descriptions. Some subjects had misunderstood certain aspects of the change task descriptions. Furthermore, one subject had misunderstood how and where the code was supposed to be written. Although the process had been explained in detail during session 1 of the pilot experiment, we discovered that there was a need to be *extremely* clear and explicit in the written materials to avoid confusion and misunderstandings.

One problem we found after the main experiment was that one of the messages in the message sequence chart for the RD design had an incorrect sequence number. For developers relying on the MSC for understanding the design, this may have influenced the time to implement the tasks, in particular for task *c*2. Although this threat cannot be ruled out, it is in our opinion very unlikely that this "bug" can explain the large difference in change effort. In the preliminary results from the think-aloud follow-up experiment (Section 6), we have found that almost none of the developers actually used the message sequence charts. Otherwise, we believe that the written materials of the main experiment were of high quality.

Size and Choice of Design Alternatives and Change Tasks

The coffee machine designs and the changes to them are small. As pointed out in Section 4.6, the RD design may support "opportunistic" programmers better than the MF design. As the size of the programs increases, memory limitations may eventually result in that it becomes too difficult to use a systematic approach, even for the "MF" type of designs. Thus, different results may have been obtained if the programs were larger. This threat to external validity should be considered in future experiments.

With regards to internal validity, it is possible that adding more than three change tasks would have produced different results:

- Adding a fourth change task may have resulted in a total "breakdown" of the MF design.
- It is possible that most of the system learning occurred during change task *c*3 for the RD design, after which subsequent changes would have been simple to understand.

Pen and Paper

The changes were coded with pen and paper. This represents another important threat to the external validity. Using a computer one has access to advanced editors, multiple windows, class browsers, etc. Some subjects preferred an exploratory approach to changing the program, which may be difficult to do with pen and paper compared with using a computer. For this experiment, the designs and the change tasks were small. Furthermore, there was a quite even distribution of subjects characterizing their solution approach as exploratory for the MF and RD designs (Section 4.6). Finally, the students are accustomed to working with pen and paper programs on their written exams. This means that the advantage of using a computer is probably not that great.

Using a computer would have introduced many new problems regarding training, learning effects and biases towards certain solution approaches depending on the available tool functionality. In this particular experiment, it was in our opinion a better approach to use pen and paper rather than a computer. Still, the only way to eliminate the resulting threats is to replicate the experiment using computers instead of pen and paper.

5.2.2. Subject Selection

One important question regarding external validity is whether the subjects are a representative sample of the population. The subjects (mostly undergraduate students, but also some graduate students and professional developers) of the experiment may not be representative of the "general programmer". Furthermore, according to Cockburn, the MF design is typical of the initial designs most students propose. Thus, it is possible that the MF design has an unfair advantage when using students as experimental subjects. The results may have been quite different if the subjects were OO design experts. Thus, we cannot rule out that the subject selection may have biased the results.

Another threat is whether some subjects actually have read Cockburn's article series or otherwise knew the details of the designs prior to the experiment. Because of randomization and the number of subjects involved in the experiment, we believe it is very unlikely that this have affected the results of the experiment.

5.2.3. Group Assignment

A serious threat to the validity of the results of *between-subject* experiments is the group assignment (Briand et al., 1999a). It is difficult to ensure that the skill levels of the two groups are approximately equal. In our case, we used the results of the common calibration task as an indicator of the skill level of each subject. The randomized block group assignment in the pilot experiment became skewed towards higher skills for subjects assigned to the RD design, because some subjects did not attend the third session of the experiment (Figure 2). Still, the average change effort for the RD design was significantly higher than for the MF design (Figure 3). Thus, for the pilot experiment, the uneven group assignment actually *strengthens* the results.

In the main experiment, we checked the skill level of the two randomized groups by calculating confidence intervals for the mean effort to implement the common calibration task:

```
Calibration                                95% CIs for mean (minutes)

Group      N       Mean    StDev    --+---------+---------+---------+----
MF        17      46.24    14.22     (------------*-------------)
RD        19      50.37    13.70            (------------*-------------)
                                    --+---------+---------+---------+----
                                     40.0      45.0      50.0      55.0
```

The confidence intervals show that the 17 subjects assigned to the MF design on average performed slightly better on the calibration task than the 19 subjects assigned to the RD design, that is, the opposite of what was the case in the pilot experiment. The difference in means is not significant, however. We also checked

Table 10. Initial and adjusted group assignment cross-tabulated on skill level.

Block (minutes)	Initial group assignment (count)		Adjusted group assignment (count)	
	MF	RD	MF	RD
<49	10	7	7	7
≥49	7	12	7	7
Total	17	19	14	14

whether more even group assignments might have produced results that are inconsistent with the results of the main experiment. First, we created two blocks based on the calibration task effort, using the median (49 min) as a boundary. We then "balanced" the group assignment by randomly removing subjects from the initial groups such that an equal number of subjects (seven, in our case) remained in each block for each group (Table 10).

Furthermore, we tested whether the mean calibration task efforts for the adjusted groups were different, using a two-sided t-test on the difference in means (H1, Table 11). Finally, we tested whether the mean total effort to implement change tasks $c1+c2+c3$ were lower on the MF design than on the RD design, using a one-sided t-test on the difference in means (H2). This process was repeated six times (Run 1–6).[2] The results are shown in Table 11. The sub-samples of the initial groups have very even mean effort to implement the calibration task (p-values from 0.66 to 0.87). Furthermore, the differences in mean total effort to implement $c1+c2+c3$ are consistent with the results presented in Section 4.1 (p-values from 0.0012 to 0.024). Thus,

Table 11. Adjusted results based on sub-samples of the original group assignment.

Run	Group	N	Effort calibration	H1: Unequal groups?	Effort $c1+c2+c3$	H2: $\mu(\text{MF}) < \mu(\text{RD})$?
1	MF	14	46.6	$p = 0.84$	47.4	$p = 0.0023$
	RD	14	47.6		59.6	
2	MF	14	44.9	$p = 0.66$	49.9	$p = 0.0087$
	RD	14	46.7		60.5	
3	MF	14	45.3	$p = 0.86$	50.5	$p = 0.0071$
	RD	14	46.0		61.3	
4	MF	14	44.8	$p = 0.85$	50.9	$p = 0.024$
	RD	14	45.6		59.7	
5	MF	14	46.6	$p = 0.66$	47.4	$p = 0.0012$
	RD	14	45.1		60.4	
6	MF	14	45.6	$p = 0.87$	47.1	$p = 0.0024$
	RD	14	46.2		59.1	

we have no reasons to believe that the group assignment threatens the results of this study.

6. Future Work

The results of this study seem to contradict commonly accepted design quality principles, such as those proposed by Coad and Yourdon. The "good" design was more difficult to change than the "bad" design. We believe that by reducing coupling and increasing cohesion one may change other aspects of the design that contribute to an increase in complexity.

To further explain the results of this experiment, we are in the process of conducting a follow-up experiment with professional programmers in industry where the subjects "think aloud" while trying to understand the change tasks. The comments and actions made by the subjects are carefully recorded and subsequently analyzed. The preliminary results from that experiment suggest that instead of creating "hypotheses" about how the design works (i.e., a more opportunistic solution approach), many subjects perform a systematic trace of the functionality related to a given change task. Thus, the deeply nested interactions among classes to implement a given functional scenario of the RD design may, to some extent, explain the increase in change effort compared with the MF design. For this reason, we are also investigating whether *dynamic* coupling measures (to measure the scenario depth) are useful in building predictive models of the changeability of object-oriented designs.

Acknowledgments

The authors thank the students at the Informatics Department at University of Oslo who participated in this study. We thank the anonymous reviewers for providing constructive and very insightful comments. We thank Alistair Cockburn for providing the example coffee machine source code from which we based our design alternatives. Finally, we thank Barbara Kitchenham and Lionel Briand for giving useful feedback on the design and statistical analysis of the experiment. This research has been funded by the PROFIT research project in Norway.

Appendix A: Experience Level Questionnaire

Name:

Total number of credits:

Total number of credits in programming:

Estimate the total number of lines of code you have written in the following programming languages:

Programming language	0	0, but knows some	<100	<1000	<10000	10000+
Java	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
C++	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
Simula	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
SmallTalk	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
C	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
Pascal	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
Other language ( )	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
Other language ( )	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
UML/rose	[ ]	[ ]	[ ]			
OMT	[ ]	[ ]	[ ]			
RD design	[ ]	[ ]	[ ]			
CRC	[ ]	[ ]	[ ]			
Role-modeling	[ ]	[ ]	[ ]			
Structured analysis and/or structured design	[ ]	[ ]	[ ]			
Data-driven design	[ ]	[ ]	[ ]			
Other ( )	[ ]	[ ]	[ ]			
Other ( )	[ ]	[ ]	[ ]			

Appendix B: Change Tasks

NAME: <THIS FIELD WAS FILLED OUT BY THE AUTHORS BEFORE THE EXPERIMENT>

IMPORTANT:

- THE CODE YOU WILL CHANGE IS ATTACHED AT THE END OF THIS DOCUMENT. DO YOUR CHANGES DIRECTLY IN THE SOURCE CODE LISTING.
- BEFORE YOU START THE CHANGE TASK, WRITE DOWN THE START TIME IN THE QUESTIONNAIRE THAT FOLLOWS THIS CHANGE TASK DESCRIPTION.
- WHEN YOU HAVE FINISHED THE CHANGE TASK, YOU COMPLETE THE REMAINDING QUESTIONS OF THE CHANGE TASK QUESTIONNAIRE, BEFORE YOU START THE NEXT ASSIGNMENT.

In this experiment you shall implement changes to a "virtual" coffee machine. At the moment, the machine can make four different types of coffee (black, white, black w/sugar, white w/sugar). The customer must give textual commands to insert money and select coffee, and will subsequently receive the "coffee," given that he has deposited sufficient funds and assuming all necessary ingredients are available. At present, coffee costs five credits. The test run given below shows how the machine works at present:

Example Test Run:

```
Menu: I = insert S = select Q = quit
I
Amount >
4
    CashBox: Depositing 4
    You now have 4 credits.
Menu: I = insert S = select Q = quit
S
Select Drink (1 = Black Coffee, 2 = Coffee w/Cream, 3 = Coffee w/Sugar, 4 = Coffee
w/Sugar & Cream) >
2
    FrontPanel: Insufficient funds
    Menu: I = insert S = select Q = quit
I
Amount >
2
    CashBox: Depositing 2
    You now have 6 credits.
Menu: I = insert S = select Q = quit
S
Select Drink (1 = Black Coffee, 2 = Coffee w/Cream, 3 = Coffee w/Sugar, 4 = Coffee
w/Sugar & Cream) >
2
    Dispensing cup
```

```
  Dispensing coffee
  Dispensing water
  Dispensing cream
  CashBox: Returning 1
```

CHANGE TASK 1[3]

In this assignment, you shall extend the coffee machine with “return button” functionality that returns the deposited funds. The menu choice is called “Return”.

Test Case:

```
Menu: I = insert S = select R = return Q = quit
I
Amount >
4
    CashBox: Depositing 4
    You now have 4 credits.

Menu: I = insert S = select R = return Q = quit
R
    CashBox: Returning 4

Menu: I = insert S = select R = return Q = quit
```

CHANGE TASK 2

In this assignment, you shall extend the machine to make bouillon. Bouillon costs more than coffee. While coffee costs five credits, bouillon costs six credits.
HINT: You must, among others, make a “dispenser” for bouillon powder.

Test Case:

```
Menu: I = insert S = select R = Return Q = quit
I
Amount >
6
    CashBox: Depositing 6
    You now have 6 credits.

Menu: I = insert S = select R = Return Q = quit
S
Select Drink (1 = Black Coffee, 2 = Coffee w/Cream, 3 = Coffee w/Sugar, 4 = Coffee
w/Sugar & Cream, 5 = Bouillon) >
5
    Dispensing cup
    Dispensing bouillon
```

```
     Dispensing water
     CashBox: Returning 0
Menu: I = insert S = select R = Return Q = quit
```

CHANGE TASK 3

Unfortunately, there is a quite serious problem with the coffee machine at present. If the user chooses for example ‘‘coffee with cream’’, and the cream dispenser is empty, the machine gives a small error message, after which it dispenses black coffee (without cream). If the machine does not contain any more cups, the machine dispenses the drink right into the drain. The user will of course get quite irritated over having to pay for this!

The simplest solution to this problem is that the user receives a message if the machine is out of a required ingredient of the selected drink. Then, the user is given the option to choose another drink. The following test case illustrates what should happen when the machine runs out of cream:

Test Case:

```
Menu: I = insert S = select R = Return Q = quit
I
Amount >
5
     CashBox: Depositing 5
     You now have 5 credits.

Menu: I = insert S = select R = Return Q = quit
S
Select Drink (1 = Black Coffee, 2 = Coffee w/Cream, 3 = Coffee w/Sugar, 4 = Coffee
w/Sugar & Cream, 5 = Bouillon) >
2
     Dispensing cup
     Dispensing coffee
     Dispensing water
     Dispensing cream <after this the machine is out of cream>
     CashBox: Returning 0

Menu: I = insert S = select R = Return Q = quit
I
Amount >
5
     CashBox: Depositing 5
     You now have 5 credits.
Menu: I = insert S = select R = Return Q = quit
```

```
S
Select Drink (1 = Black Coffee, 2 = Coffee w/Cream, 3 = Coffee w/Sugar, 4 = Coffee
w/Sugar & Cream, 5 = Bouillon) >
2
        Sorry, no more cream! Select another.
Menu: I = insert S = select R = Return Q = quit
```

Appendix C: Change Task Questionnaire

Time for start of the change task:

Time for completing the change task (not including answering this questionnaire):

Effort (in minutes) to solve the change task:
A. Effort to understand how to solve the change task:
B. Effort to code the change task:
C. Effort to evaluate/test the solution (run test-case):

How would you characterize your strategy to solve the task?
Very explorative (1) – Very systematic (6):
(Very explorative = ''*trial and error*'')
(Very systematic = ''*analysis, design, code, test*'')

What is your subjective assessment of your skill level as a programmer?
Very poor (1) – Very skilled (6):

What is your subjective assessment of the quality of your solution?
Very poor (1) – Very good (6):

How confident are you that the solution does not contain serious faults?
Very unsure (1) – Very confident (6):

How difficult did you think the change task was?
Very easy (1) – Very difficult (6):

OTHER COMMENTS:

Appendix D: Message Sequence Charts for the Designs

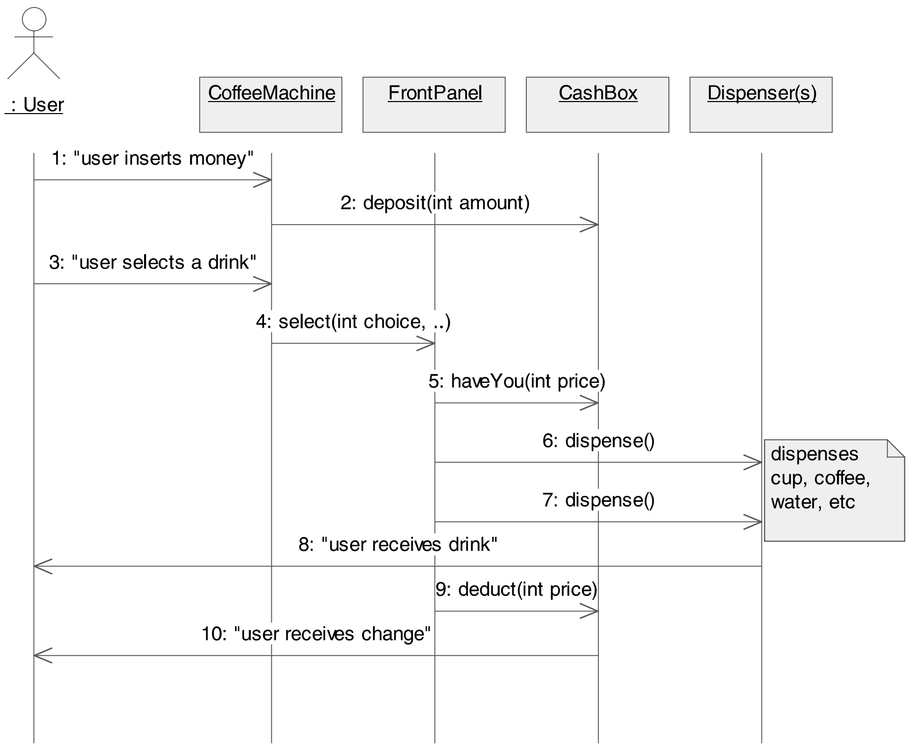

Figure 8. Message sequence chart for the MF design.

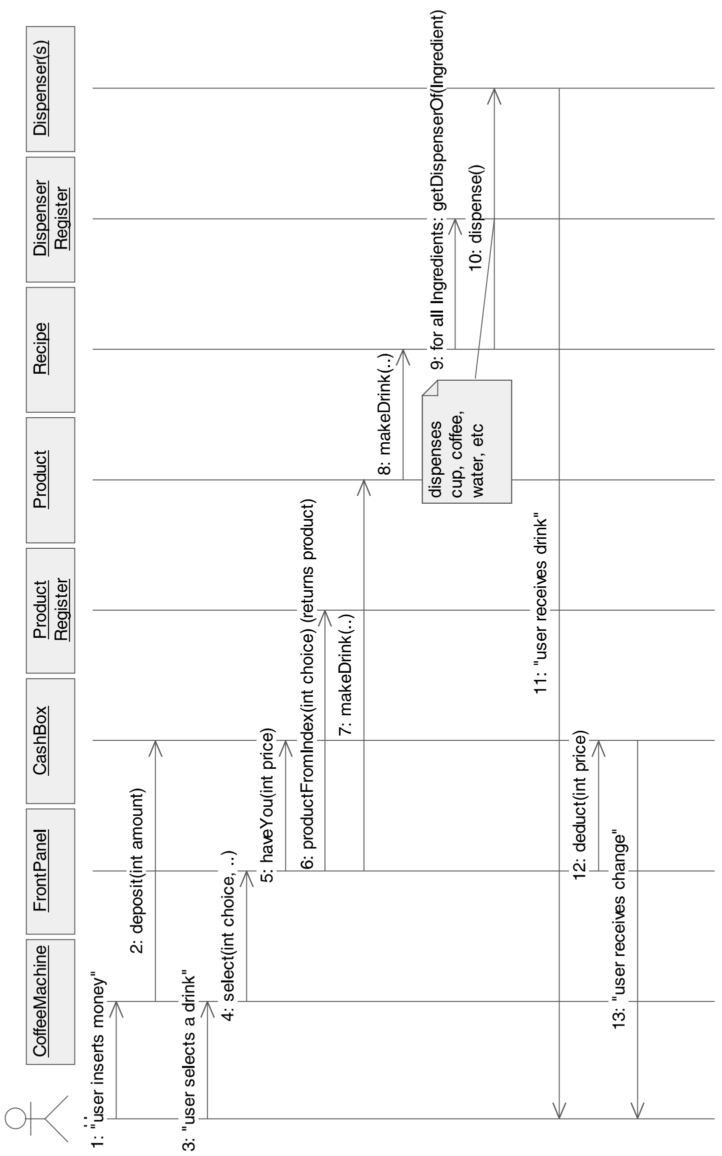
:
CoffeeMachine
FrontPanel
CashBox
Product Register
Product
Recipe
Dispenser Register
Dispenser(s)
1: "user inserts money"
2: deposit(int amount)
3: "user selects a drink"
4: select(int choice, ..)
5: haveYou(int price)
6: productFromIndex(int choice) (returns product)
7: makeDrink(..)
8: makeDrink(..)
9: for all Ingredients: getDispenserOf(Ingredient)
10: dispense()
dispenses cup, coffee, water, etc
11: "user receives drink"
12: deduct(int price)
13: "user receives change"

Figure 9. Message sequence chart for the RD design.

Appendix E: Code Fragments from the Designs

Table 12. Code fragmnt from the MF design.

```
class FrontPanel
{
        // knows price of selection;
        // knows ingredients needed for each selection
        // asks CashBox how much money was put in
        // instructs dispensers
        <…snip…>

        // constructor method for the FrontPanel class
        FrontPanel() {
                cupDisp = new Dispenser("cup", 50); // 50 cups
                waterDisp = new Dispenser("water", 50);
               <…snip…>
        }

        // user selected a drink. Make drink!
        void select(int choice, CashBox cashBox)
        {
            if (cashBox.haveYou(drinkPrice))
            {
                //1 = Black Coffee, 2=Coffee w/Cream, 3 = Coffee w/Sugar,
                //4 = Coffee w/Sugar & Cream
                if (choice == 1)
                {
                        cupDisp.dispense();
                        coffeeDisp.dispense();
                        waterDisp.dispense();
                }
                <…snip…>
                else // cream & sugar
                {
                        cupDisp.dispense();
                        coffeeDisp.dispense();
                        waterDisp.dispense();
                        sugarDisp.dispense();
                        creamDisp.dispense();
                }
                cashBox.deduct(drinkPrice);
         }
         else
         {
             Output.print("\tFrontPanel: Insufficient funds");
         }
  }
```

Table 13. Code fragmnt from the RD design.

```
class FrontPanel
{
        // knows price,
        // knows products,
        // asks CashBox how much money was put in
        // instructs the correct product to make the drink
        // instructs CashBox to return change

        private int drinkPrice = 5;

        void select(int choiceIndex, CashBox cashBox, ProductRegister
                        productReg, DispenserRegister dispenserReg) {

            if (cashBox.haveYou(drinkPrice))
            {
                Product product;
                product = productReg.productFromIndex(choiceIndex);
                //1 = Black Coffee, 2=Coffee w/Cream, 3=Coffee w/Sugar,
                //4 = Coffee w/Sugar & Cream

                product.makeDrink(dispenserReg);
                //make product using the dispensers

                cashBox.deduct(drinkPrice); //deduct price
            }
            else
            {
                Output.print("\tFrontPanel: Insufficient funds\n");
            }
        }
    }
```

Appendix F: Raw Data from the Main Experiment

Table 14. Summary measures of change effort.

Subject	Design	Total c1+c2	Total c1+c2+c3	Understand c1+c2+c3	Code c1+c2+c3	Test c1+c2+c3	Learning Curve
1	RD	32	56	30	23	3	−0.520000
2	MF	26	48	15	23.5	9.5	0.200000
3	RD	42	58	29	26	3	−0.090909
4	RD	40	53	36	13	4	0.076923
5	MF	38	63	30	25	8	0.000000
6	RD	56	*	*	*	*	*
7	RD	26	61	12	40	9	0.333333
8	RD	28	42	18	19	5	−0.333333
9	RD	23	38	23	*	*	−0.500000
10	MF	43	79	13	54	12	−0.111111
11	RD	29.5	57.5	20	33.5	4	0.411765
12	RD	31	56	8	38	10	−0.200000
13	MF	30	56	23	28	5	−0.375000
14	RD	49	64	35	*	*	0.200000
15	MF	20	35	17	13	5	−0.166667
16	RD	39	48	25	21	2	0.000000
17	MF	21	36	14	18	4	0.111111
18	RD	54	69	34	26	9	0.142857
19	MF	17	42	8	27	7	−0.428571
20	MF	13	36	10	26	0	−0.250000
21	RD	45	65	35	24	6	−0.500000
22	MF	24	44	11	22	12	−0.666667
23	MF	25	40	11	25	4	0.250000
24	RD	51	67	21	29	*	0.384615
25	MF	24	48	8	34	6	0.000000
26	RD	33.5	61.5	*	*	*	*
27	MF	35	41	9.5	26.5	5	0.846154
28	RD	49	71	39	28	9	0.166667
29	MF	26	52	13	31	8	0.000000
30	MF	40	65	30	35	0	−0.200000
31	MF	31	61	20	29	12	−0.333333
32	RD	29	64	33	29	2	−0.818182
33	RD	30	70	20	42	8	0.000000
34	MF	24	41	24	17	0	−0.411765
35	RD	40	65	25	25	15	−0.500000
36	MF	20	*	*	*	*	*

Table 15. Data for the calibration task.

Subject	Total Cal	Understand Cal	Code Cal	Test Cal	Strategy Cal	Subj Qual. Cal	Confidence Cal	Correctness Cal	ChangeSize Cal
1	55	15	35	5	2	2	2	2	39
2	39	20	14	5	2	4	3	3	17
3	60	28	30	2	4	4	5	6	25
4	65	35	20	10	2	3.5	3	3	14
5	50	25	20	5	5	4	4	5	22
6	56	25	15	16	3	4	4	3	21
7	38	10	23	5	2	5	4	4	19
8	31	14	15	2	2	5	5	4	26
9	55	30	20	5	2	2	3	2	19
10	23	10	10	3	5	5	5	4	18
11	29	14	12	3	3	3	5	5	20
12	37	10	20	7	3	3	6	6	28
13	62	20	40	2	5	5	5	5	23
14	55	10	40	5	2	3	4	3	23
15	40	20	15	5	4	3	5	6	18
16	87	37	40	10	5	4	4	3	30
17	42	16	25	1	5	4	5	4	21
18	50	25	20	5	2	2	3	2	11
19	55	17.5	32.5	2.5	3	4	5	6	24
20	50	20	20	10	3	2	2	2	14
21	55	20	30	5	2	4	4	6	25
22	25	7	15	3	4	5	5	6	15
23	33	7	20	5	4	5	6	4	24
24	51	14	30	7	4	4	3	3	20
25	40	8	25	7	2	5	6	6	28
26	57	30	25	2	4	5	5	4	16
27	50	20	28	2	5	4	5	3	23
28	43	20	10	13	5	4	5	5	19
29	52	10	30	12	4	4	4	6	24
30	45	20	20	5	5.5	4.5	6	5	21

Table 15. Data for the calibration task (*continued*).

Subject	Total Cal	Understand Cal	Code Cal	Test Cal	Strategy Cal	Subj Qual. Cal	Confidence Cal	Correctness Cal	ChangeSize Cal
31	45	20	20	5	4	4	5	6	24
32	55	20	30	5	5	6	2	2	15
33	33	15	15	3	4	5	6	3	9
34	50	40	10	0	6	6	6	4	15
35	45	20	20	5	3	4	4	4	18
36	85	60	20	5	3	1	1	3	19

Table 16. Data for change task *c*1.

Subject	Total	Understand	Code	Test	Strategy	Subj Qual	Confidence	Difficulty	Correctness	ChangeSize
	*c*1	*c*1	*c*1	*c*1	*c*1	*c*1	*c*1	*c*1	*c*1	*c*1
1	12	6	5	1	4	5	5	2	6	3
2	10	6	3.5	0.5	2	4	5	2	6	3
3	13	10	2	1	2	4	3	1	4	2
4	20	14	5	1	4	5	3	1	4	2
5	16	10	5	1	4	5	5	1	6	3
6	9	7	1	1	2	4	2	1	4	2
7	14	4	8	2	1	6	5	2	6	3
8	8	3	4	1	2	5	6	1	6	6
9	10	5	5	0	2	1	6	1	6	3
10	10	4	3	3	2	6	6	1	6	3
11	20.5	12	8	0.5	5	5	5	1	6	8
12	10	2	7	1	6	5	6	1	4	2
13	12	5	6	1	2	4	4	2	6	4
14	21	15	4	2	2	5	5	2	6	3
15	10	5	3	2	5	5	5	2	6	3
16	8	5	2	1	3	3	3	2	6	3
17	8	5	1	2	5	5	5	1	6	3
18	26	12	10	4	4	4	4	2	6	3
19	10	2	7	1	4	6	6	1	4	2
20	7	3	4	0	4	5	5	1	4	3
21	10	5	4	1	4	5	5	1	6	3
22	4	1	2	2	1	5	6	1	6	3
23	10	5	5	0	2	6	6	1	6	3
24	14	9	2	3	3	5	5	1	6	3
25	7	3	2	2	1	6	6	1	4	3
26	13.5	10	3	0.5	4	5	5	1	6	3
27	18	6	10	2	3	3	3	2	6	3
28	19	14	3	2	4	4	4	3	6	3
29	12	5	5	2	3	4	4	1	6	6
30	20	10	10	0	6	5.5	6	1	6	3
31	11	5	4	2	4	5	5	2	4	2
32	10	3	6	1	3	6	6	1	6	3
33	13	10	2	1	5	6	6	2	4	2
34	10	5	5	0	2	6	4	1	6	3
35	20	5	10	5	4	4	3	2	6	3
36	10	2	5	3	4	5	5	1	6	3

Table 17. Data for change task *c*2.

Subject	Total *c*2	Understand *c*2	Code *c*2	Test *c*2	Strategy *c*2	Subj Qual. *c*2	Confidence *c*2	Difficulty *c*2	Correctness *c*2	ChangeSize *c*2
1	20	5	14	1	2	3	4	3	4	12
2	16	5	8	3	2	5	4	3	6	13
3	29	7	20	2	4	5	5	3	4	17
4	20	10	7	3	1	3	2	4	3	5
5	22	10	10	2	3	4	4	2	6	10
6	47	30	5	12	5.5	3	2	4	4	6
7	12	6	4	2	1	5	5	2	4	8
8	20	9	9	2	2	5	5	3	4	7
9	13	3	8	2	1	4	4	2	4	6
10	33	4	26	3	1	1	5	5	6	27
11	9	3	5.5	0.5	3	5	5	1	3	4
12	21	3	16	2	3	3	5	2	4	14
13	18	7	10	1	2	4	4	2	4	8
14	28	10	15	3	4	5	4	3	6	8
15	10	5	3	2	3	3	4	3	6	11
16	31	15	15	1	2	2	1	4	3	8
17	13	5	7	1	5	6	5	1	4	11
18	28	13	10	5	3	5	4	3	4	7
19	7	1	5	1	5	6	6	1	3	7
20	6	2	4	0	4	5	5	1	3	11
21	35	15	15	5	3	4	4	3	6	15
22	20	5	10	5	2	4	5	3	6	14
23	15	3	10	2	3	5	5	1	6	11
24	37	8	15	14	4	5	3	3	6	17
25	17	2	13	2	2	5	5	2	5	16
26	20	*	*	*	2	5	5	1	5	10
27	17	3	12	2	4	4	3	3	6	12
28	30	15	16	4	5	3	3	4	6	8
29	14	3	8	3	2	4	4	2	6	16
30	20	5	15	0	5	4	6	1	4	25
31	20	5	10	5	3	4	5	4	6	9
32	19	0	18	1	1	5	4	2	6	9
33	17	0	15	2	2	4	3	3	5	14
34	14	7	7	0	2	5	5	2	3	14
35	20	5	10	5	3	3	2	2.5	4	6
36	10	2	8	0	4	4	4	4	*	*

Table 18. Data for change task *c*3.

Subject	Total *c*3	Understand *c*3	Code *c*3	Test c3	Strategy *c*3	Subj Qual. *c*3	Confidence *c*3	Difficulty *c*3	Correctness *c*3	ChangeSize *c*3
1	24	19	4	1	2	3	3	4	2	*
2	22	4	12	6	2	3	4	2	5	15
3	16	12	4	0	4	4	3	4	*	*
4	13	12	1	0	3	*	*	*	*	*
5	25	10	10	5	4	3	3	3	6	67
6	*	*	*	*	5	*	*	*	*	*
7	35	2	28	5	3	3	3	5	6	16
8	14	6	6	2	1	2	3	3	4	11
9	15	15	*	*	6	1	1	1	*	*
10	36	5	25	6	*	3	3	5	5	22
11	28	5	20	3	3	4.5	4	3	6	18
12	25	3	15	7	3	4	3	2	4	10
13	26	11	12	3	1	4	4	2	2	60
14	15	10	*	*	3	*	*	3	*	*
15	15	7	7	1	2	1	3	3	6	70
16	9	5	4	0	1	1	1	*	*	*
17	15	4	10	1	5	4	5	2	4	62
18	15	9	6	0	4	3	3	3	*	*
19	25	5	15	5	2	5	5	2	3	28
20	23	5	18	0	4	4	4	2	3	60
21	20	15	5	0	2	2	3	3	*	*
22	20	5	10	5	4	4	5	4	6	13
23	15	3	10	2	2	3	4	2	3	56
24	16	4	12	*	3	2	2	2	6	14
25	24	3	19	2	1	5	4	2	6	24
26	28	*	*	*	1	4	3	2	6	16
27	6	0.5	4.5	1	5	5	4	1	1	2
28	22	10	9	3	5	4	4	3	5	9
29	26	5	18	3	*	3	4	4	6	16
30	25	15	10	0	6	4	6	2	5	11
31	30	10	15	5	2	3	4	4	6	29
32	35	30	5	0	2	2	1	4	*	*
33	40	10	25	5	3	3	3	5	6	20
34	17	12	5	0	5	6	5	2	6	20
35	25	15	5	5	3	2	1	3	*	*
36	*	*	*	*	*	*	*	*	*	*

Notes

1. It is interesting that the professional programmers and graduate students of the pilot experiment implemented *c*1 and *c*2 significantly faster than the undergraduate students of the main experiment, eventhough the undergraduate students had more experience with Java. On average, the graduate students/professionals used about 40% less time than the undergraduate students.
2. There are (12!/7!5!)(10!/7!3!) = 95040 ways to select such balanced sub-samples from the initial group assignment. We selected only six of these possible samples at random.
3. In the actual handouts, each change task was described on a separate printed page. The subjects were instructed not to look at the next change task before they had completed the change task questionnaire for the previous change task.
4. The subjective skill-level question was asked only once, in the calibration change task questionnaire. The other questions were answered for all change tasks.

References

Arisholm, E., and Sjøberg, D. I. K. 2000. Towards a framework for empirical assessment of changeability decay. *Journal of Systems and Software* 53(10): 3–14.

Arisholm, E., Benestad, H. C., Skandsen, J., and Fredhall, H. 1998. Incorporating rapid user interface prototyping in object-oriented analysis and design with Genova. *Proceedings of NWPER'98 Nordic Workshop on Programming Environment Research*, Sweden, pp. 155–161.

Arisholm, E., Skandsen, J., Sagli, K., and Sjøberg, D. I. K. 1999. Improving an evolutionary development process—a case study. *Proceedings of the EuroSPI'99 (European Software Process Improvement Conference)*, Pori, Finland, pp. 9.40–9.50.

Basili, V. R., Briand, L. C., and Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 22(10): 751–761.

Briand, L. C., Bunse, C., Daly, J. W., and Differding, C. 1997. An experimental comparison of the maintainability of object-oriented and structured design documents. *Empirical Software Engineering* 2(3): 291–312.

Briand, L., Bunse, C., and Daly, J. W. 1999a. A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *ISERN Technical Report 99-07*.

Briand, L. C., Arisholm, E., Counsell, S., Houdek, F., and Thevenod, P. 1999b. Empirical studies of object-oriented artifacts, methods, and processes: State of the art and future directions. *Empirical Software Engineering* 4(4): 387–404.

Briand, L. C., Daly, J. W., Porter, V., and Wust, J. 1999c. A comprehensive empirical validation of product measures for object-oriented systems. *To appear in Journal of Systems and Software*. Also available as *ISERN Technical Report 98-07*

Briand, L. C., Wust, J., Ikonomovski, S. V., and Lounis, H. 1999d. Investigating quality factors in object-oriented designs: An industrial case study. *21st International Conference of Software Engineering (ICSE'99)*, Los Angeles, CA, pp. 345–354.

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

Coad, P., and Yourdon, E. 1991a. *Object-Oriented Analysis*. Prentice-Hall.

Coad, P., and Yourdon, E. 1991b. *Object-Oriented Design*. Prentice-Hall.

Cockburn, A. 1998. The Coffee Machine Design Problem: Part 1 & 2. *C/C++ User's Journal*, May/June 1998.

Daly, J., Brooks, A., Miller, J., Roper, M., and Wood, M. 1996. Evaluating inheritance depth on the maintainability of object-oriented software. *Empirical Software Engineering* 1(2): 109–132.

Holm, S. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics* 6: 65–70.

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

Li, W., and Henry, S. 1993. Object-oriented metrics that predict maintainability. *Journal of Systems and Software* 23(2): 111–122.

Sharble, R. C., and Cohen, S. S. 1993. The object-oriented brewery: A comparison of two object-oriented development methods. *Software Engineering Notes* 18(2): 60–73.

von Mayrhauser, A., Vans, A. M., and Howe, A. E. 1997. Program understanding behaviour during enhancement of large-scale software. *Software Maintenance: Research and Practice* 9: 299–327.

Erik Arisholm received a B.Sc. degree in computer science from the University of Oslo in 1988, a M.A.Sc. degree in electrical engineering from the University of Toronto in 1991, and a Ph.D. degree in computer science from University of Oslo in 2001. He has seven years industry experience in Canada and Norway as a Lead Engineer and Design Manager. He is now an Associate Professor in software engineering and a member of the research group Industrial Systems Development in the Department of Informatics, University of Oslo. He is also a researcher in the Software Engineering group of Simula Research Laboratory. His research interests include empirical software engineering, software process improvement and object-oriented methods.

Magne Jørgensen received the Diplom Ingeneur degree in Wirtschaftswissenschaften from the University of Karlsrube, Germany, in 1988 and the Dr. Scient. degree in informatics from the University of Oslo, Norway in 1994. He has 10 years industry experience as consultant and manager. He is now an Associate Professor in software engineering and member of the research group Industrial Systems Development in the Department of Informatics, University of Oslo. He is also a researcher in the Software Engineering group of Simula Research Laboratory. His research interests include software process improvement, experience databases, software estimation and software measurements.

Dag Sjøberg received an M.Sc. degree in computer science from University of Oslo in 1987 and a Ph.D. degree in computing science from University of Glasgow in 1993. He has five years industry experience as consultant and Group Leader. He is now a Professor in software engineering and is the leader of the research group Industrial System Development in the Department of Informatics, University of Oslo and Head of the Software Engineering group of Simula Research Laboratory. Among his research interests are software evolution, software process improvement, programming environments, object-oriented methods and persistent programming.