# An External Replication of a Korson Experiment

**RR/162/94 [EFoCS-4-94]**

J Daly, A Brooks, J Miller, M Roper, and M Wood

Empirical Foundations of Computer Science (EFoCS)*

**Abstract**

*Carrying out empirical studies to understand the problems of software engineering is widely held to be of importance. A view less widely held within the software engineering community is that experiments should be replicated externally (i.e. by different researchers) to both verify and validate the original results and so build up a sound body of knowledge (as opposed to drawing conclusions from a single study).*

*This paper presents the results of an external replication of an experiment which tested the benefits to maintenance of using modular code against non-modular (monolithic) code. The results of the original experiment [8, 9] showed that a modular program could be maintained significantly faster than an equivalent monolithic version of the same program under the condition that modularity has been used to implement information hiding which localizes changes required by a modification.*

*The results of our replication, however, were strikingly different from those of the original and showed no significant difference between the average times taken to maintain modular and monolithic code. An inductive analysis was undertaken to investigate the reasons for this difference. Evidence was uncovered suggesting an ability effect (which was not observed in the original experiment), a lack of realism in the context in which subjects were asked to perform tasks, differing degrees of subjects' understanding of the programs, different approaches taken by subjects towards making the required modifications, and possible deficiencies of subject monitoring. This evidence, along with evidence for other sources of variability, is discussed in some detail.*

*It is concluded that external replications, combined with inductive analysis techniques, have a vital part to play in the realisation of generalizable results.*

Keywords : empirical, modifiability, modularity, replication

*Dept. Computer Science, University of Strathclyde, Livingstone Tower, Richmond Street, Glasgow G1 1XH, U.K., +44 (0)41 552 4400, jd@cs.strath.ac.uk

# 1 Introduction

Carrying out empirical studies to understand the problems of software engineering is widely held to be of importance. There are, however, inherent problems in drawing conclusions from a single experimental result: every stage of an experiment from design through to result interpretation is prone to error. As a consequence, scientists demand that experimental results are externally reproducable i.e. that an independent group of other researchers can do the experiment and obtain similar results. External replication is a cornerstone of modern scientific disciplines. Results can either support or question the claims arising from the original experiment. When results are similar, confidence is built up in both the procedure and the original claims. Without the confirming power of an external replication, empirically-based claims should be regarded at best as of limited importance and at worst with suspicion and mistrust.

Where the same researchers conduct a series of similar experiments, the replication is said to be internal: in this case, however, the replications carry much less confirming power. In a previous paper [3], we have argued that the experimentation framework of Basili et al [2] for software engineering should be extended to explicitly include external replications.

This paper presents the findings of an external replication of an empirical study carried out by Korson [8, 9]. The empirical study was concerned with the effects of modularity on program maintenance and was chosen as our first attempt at performing an external replication on several counts. Firstly, Korson claimed to have succeeded in providing internal replicability stating,

> ...the study has demonstrated that a carefully designed empirical study using programmers can lead to replicable, unambiguous conclusions.

Secondly, on a first reading, Korson's work qualified as well performed empirical work (unlike much empirical work in software engineering which can be virtually dismissed on a first reading). Thirdly, if Korson's work bears up to detailed scrutiny and is found to be externally replicable, then this would increase confidence in the commonly accepted guideline that modular programs are easier and, therefore, faster to maintain than non-modular ones (under the condition that modularity has been used to implement information hiding which localizes changes required by a modification).

# 2 Review and critique of Korson's work

## 2.1 Review

Korson [8, 9] designed a series of four experiments each testing some aspect of maintenance. The experiment of greatest interest to us (his Experiment 1) was designed to test if a modular program used to implement information hiding, which localizes changes required by a modification, is faster to modify than a non-modular but otherwise equivalent version of the same program. The non-modular (or monolithic) program was created by replacing every procedure and function call in the modular version with the body of that procedure or function. Programmers were asked to make functionally equivalent changes to an inventory, point of sale program - either the modular version (approximately 1000 lines long) or the monolithic version (approximately 1400 lines long). The changes required could be classified as perfective maintenance as defined by Lientz and Swanson [10] i.e. changes made to enhance performance, cost effectiveness, efficiency, and maintainability of a program. Korson hypothesized that the time taken to make the perfective maintenance changes would be significantly faster for the modular version.

The modification process the subjects participated in contained four phases:

**Phase 1 think:** on paper, the modifications are coded noting deletions, additions and changes to the original source code.

**Phase 2 edit:** at the computer, the original program is edited to reflect the modifications made on paper.

**Phase 3 syntax:** the syntax errors are interactively removed from the modifications.

**Phase 4 logic:** logic errors are interactively debugged until the program passes a standard test.

Beforehand, participants were given ample time to read the instructions and to ask questions on the experimental process and the perfective maintenance tasks. When subjects were satisfied that they understood the nature of the required modifications, each participant was given the appropriate program listing (modular or monolithic) and Phase 1 timings started. Phase 2 began when the participant had made the changes on paper and was ready to modify the actual program. Phase 3 started when the required changes had been entered but syntax errors had yet to be removed. Finally, Phase 4 was entered when syntax errors had been removed. The times for each of these phases were recorded and used as a basis for the statistical results (see Section 5.1), but note that

Korson regarded Phase 2 as simply an exercise in typing so he does not present timing data for this phase nor involve this phase in the analysis.

Prior to the experiment each subject completed a pretest. This was used simply as a way of familiarising the subjects with the experimental design and the computer hardware and software environments.

Korson summarised his results for his Experiments 1-4, as follows:

> The study provides strong evidence that a modular program is faster to modify than a non-modular, but otherwise equivalent version of the same program, when the following conditions hold: (a) Modularity has been used to implement "information hiding" which localises changes required by a modification. (b) Existing modules in a program perform useful generic operations, some of which can be used in implementing a modification. (c) A significant understanding of, and changes to, the existing code are required for performing a modification.
>
> In contrast, the study provides evidence that modifications not fitting into the above categories are unaided by the presence of modularity in the source code.

Condition (a), arising from his Experiment 1, is the condition of interest here. Korson's result summary is given in full to draw attention to his claim that not all modifications are necessarily made more quickly by using modularised code. His conditions can be viewed as an attempt at defining a subset of the state-space of all modular programs for which modularity allows program modifications to be made faster.

## 2.2 Critique

On a first reading, Korson's work could be classified as well performed empirical work. On closer inspection, however, there are a number of criticisms that can be made.

In the modular program for Korson's experiment, a global array variable *InventoryArray* is used instead of having a proper driver procedure with a local array variable formally passed as a parameter to the four modules which require access. So the modular program does not implement true information hiding and the result is biased: had full parameter passing been implemented for the array variable, modular subjects may have taken longer to perform the maintenance tasks. Korson discusses how common coupling is allowed for an information cluster but the implementation allows the whole program access to the array variable rather than restricting access to the four procedures which might be construed as making up an information cluster.

An additional criticism arises because of the global declaration of array variable *InventoryArray*. Modular subjects are specifically asked to delete the declaration but the declaration is immediately succeeded by the following very helpful comment:

{All access to inventory information is via the following four procedures}

In the act of deleting the global array variable declaration, modular subjects, by reading the next line, are given the biased information that they need only look at the next four modules which are conveniently placed at the top of the procedure section. Had there been a driver procedure with a local array variable, modular subjects would have been placed in the same position as the monolithic subjects in having first to take time to locate the relevant code. This criticism is all the more pertinent since Korson in [8] develops the explanation that linear searching of monolithic code is very time consuming whilst modular code allows the programmer to use a tree search strategy. So Korson [8, 9] incorrectly asserted that use made of comments was a variable held constant.

The instruction to delete a global variable declaration is questionable from an entirely different point of view: a very wary subject may suspect that such a variable is inadvertently used elsewhere in the program and so perform additional searches to check for references.

A major criticism of any piece of experimental work involving human subjects is the failure to supplement a traditional statistical approach with a more qualitative or ethnographic approach which seeks to provide alternative interpretations of the data at the level of individual subject behaviour. Korson relies solely on timing data in developing an interpretation of the data. Whilst product measures of performance are important, some understanding of the underlying processes can dramatically change the interpretation of the results. Alternative interpretations may have been realised if notes made by subjects on program listings had been analysed, if subjects had been debriefed as to the cause of difficulties which resulted in highly skewed timings, if analyses had been performed on the various saved versions of the modified program during the phases of the experiment, or if analyses had been made of any differences between subjects who were professional programmers and those who were advanced computer science students.

Another major criticism of any piece of experimental work involving human subjects is a lack of realism in either the tasks or the context in which subjects are asked to do the tasks. Korson's experiment has to be regarded as having imposed an artificial ordering of work into four phases. One criticism here is that the most natural way for a programmer to locate code to change is by using the search facility of a text editor: in the Korson experiment, subjects had to manually search and write changes on, a hardcopy listing of the program during Phase 1. So the validity of the experiment must be questioned - real programmers are unlikely to behave this way.

In later sections, evidence will be presented which corroborates much of the criticisms presented above.

# 3 Pilot study

## 3.1 Introduction

An initial pilot study was conducted with four members of EFoCS to ensure that there were no problems with, for example, the experimental materials or the execution of the tasks in our computer environment. Problems that were discovered are now discussed. Comments made by pilot subjects during an informal debriefing session were noted and these are also presented.

## 3.2 Problems encountered during the pilot study

The pilot study session consisted of performing the required modifications to both the pretest and experiment programs. No problems were encountered during the pretest pilot - it was a straight forward modification to a page of Turbo Pascal code which sorted words into alphabetic order. During the experiment pilot, however, the following problems were encountered:

1. The Alt and function keys used for testing the program worked before the modification but failed after it because of performance changes not taken into consideration. This problem was later remedied for the experiment proper using a hot key customisation tool.

2. Some spelling mistakes in the documentation were noted and remedied for the experiment proper.

3. Subjects were unfamiliar with some of the American-specific terminology.

4. One of the subjects asked what the name of the file to modify was despite it being mentioned in the instructions.

5. A large sheet of test results caused data identification problems for some of the subjects.

The solutions adopted for the last three of these problems are discussed in the section on experimental design differences (Section 4.1).

### 3.3 Results of pilot study debriefing

Subjects were informally debriefed after the pilot study. Their substantive comments (paraphrased here) on the pretest and experiment were as follows:

**Pretest:** - an unrealistic problem but useful for Turbo Pascal practice - use of a fixed array for file processing was unnatural

**Experiment:** - the pretest taught you the semantics of the task ( in pretest, use disk seek read and write pairs and replace with array assignment statements whilst in the experiment use array assignment statements and replace these with disk seek read and write pairs) - no intellectual capacity needed to perform changes (in either monolithic or modular programs) - in the modular program, there is an instruction to delete the global array variable *InventoryArray* but this happens to be the last variable declaration which is immediately followed by the comment that the following procedures are the only ones that ever use this array: as a consequence I no longer have to consider the program, just looking at the four procedures - in the monolithic program, the task turns into a global search and replace operation and as such is unrealistic because I can't use the editing environment and feel disadvantaged at having to manually search through listing for all the changes - in the modular program, unhappy that arrays are passed as global structures instead of being passed as parameters: means that the task is much easier than it should be - after first pass, editing and compiling wrapped into logic phase - suggest highlighting name of file - if you make a mistake, only then do you have to try and understand what is going on - you have to remember to restore the test data files properly if testing reveals you have made a mistake otherwise you can end up taking even more time - program layout and commentary could be improved - I think I had written down all the changes correctly on paper but had incorrectly typed in one of the changes

Some of these comments corroborate some of the criticism presented in Section 2.2 and some can be viewed as providing additional criticisms of Korson's study. In the discussion section later (Section 5.5), we find similarity between some of the comments above and some of those made by subjects of the experiment proper.

# 4 The replication study

## 4.1 Experimental design differences

The problem of deciding if any changes should be made to the experimental recipe is discussed later in Section 6, along with those suggestions for improvement which we chose not to implement.

Several changes were made to the experimental instructions in an attempt to improve their readability and highlight particular points. These were minor recipe-improving changes aimed at reducing sources of variability and as such we felt justified in making them.

As noted earlier, a problem was encountered with American-specific terminology e.g. layaway. The solution adopted was to create explanatory footnotes in the documentation given to experimental subjects. To avoid subjects experiencing problems locating the name of the file to modify, the name of the file was highlighted in bold in the documentation. (The filename was in parentheses in the middle of a large paragraph and was deemed to be easily missed on a first reading.) Similarly, in the testing phase of the instructions the Alt key numbers were highlighted by placing in bold. The participants were given a listing of the input data file and a listing of the modified file after the program was run with test data for comparison to their actual program output. As with Korson, the actual data changes they were looking for on the listings were circled in order to reduce possible variability in testing times: we had overlooked doing this for our pilot study.

As Korson did not record the number of participants to a monitor no comments can be made about differences. In our case, however, there were 4 or 5 participants to each monitor.

Korson provided a Turbo Pascal reference manual for each participant. We, on the other hand, only had one such manual between 4 or 5 participants although it was noted that no reference conflicts were caused by this. Furthermore, Korson allowed his subjects to keep the Turbo Pascal reference manuals as payment for their participation. We, on the other hand, did not take money from the participating undergraduate student subjects for photocopied lecture notes later in the semester.

Our experiments were run in the afternoon from 2pm until 6pm rather than between 6pm and 10pm as Korson had done. Both our pretest and experiment were run during this time; it is unclear, however, if Korson also ran his pretest on the same occasion as his Experiment 1, though we believe it unlikely. Between the pretest and the experiment we provided our subjects with a break and light refreshments.

## 4.2 Replication methodology

The replicated experiment required at least 16 subjects, the number used by Korson. Data for 17 out of 23 voluntary subjects was collected. To our surprise, especially so given the comment of one the pilot subjects who felt that the experimental task required no intellectual capacity, 6 subjects[1] failed to complete the experiment by 6pm. Since no payment was made, in contrast to Korson, these subjects were not asked to return later to complete the experiment. The subjects, all members of the Computer Science Department at the University of Strathclyde, were a mixture of 2nd year (5), 3rd year (2), final year students (9), research students (4), and research assistants (3). While this could be said to be a fairly heterogeneous group in terms of age, qualifications and ability, all subjects met the criteria set by Korson: "fluency in Pascal, knowledge of the IBM-PC, and an amount of programming experience" [8]. Subjects, randomly assigned to two groups, participated in the experiment in a single laboratory - subjects with monolithic programs sat next to subjects with modular programs to discourage plagiarism, although this was not a significant worry. Subjects were not told about the nature of the experiment, but were informed that different versions of the program existed and that, subsequently, certain subjects were likely to finish quicker than others. This was to reduce subjects concern about their performance. The procedure each subject followed was:

1. pretest to familiarize subjects with the environment; for example experimental procedure, editor, etc.

2. short break (10 - 15 minutes) for refreshments and to allow subjects to clear their thoughts,

3. experiment,

4. finally, each subject was asked for brief personal details and to complete a debriefing questionnaire (see Appendix B).

Both pretest and experiment were of the same format that Korson used (see Section 2) and the reader is directed to his thesis [8] for details of the instructions, documentation, modification specifications, and full source code listings.

[1] Three from the modular group and three from the monolithic group.

| | Group 1 (Modular) | | | | | Group 2 (Monolithic) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | think | syntax | logic | total | | think | syntax | logic | total |
| Subject A | 15 | 1 | 2 | 18 | Subject I | 15 | 4 | 5 | 24 |
| Subject B | 14 | 7 | 4 | 25 | Subject J | 32 | 6 | 3 | 41 |
| Subject C | 16 | 1 | 10 | 27 | Subject K | 38 | 2 | 4 | 44 |
| Subject D | 22 | 1 | 20 | 43 | Subject L | 44 | 1 | 4 | 49 |
| Subject E | 39 | 3 | 14 | 56 | Subject M | 28 | 11 | 11 | 50 |
| Subject F | 34 | 19 | 5 | 58 | Subject N | 36 | 22 | 1 | 59 |
| Subject G | 31 | 29 | 1 | 61 | Subject O | 29 | 3 | 29 | 61 |
| Subject H | 46 | 25 | 25 | 96 | Subject P | 55 | 2 | 36 | 93 |
| | | | | | Subject Q | 38 | 15 | 58 | 111 |
| Mean(mins.) | 27.1 | 10.8 | 10.1 | 48.0 | | 35.0 | 7.3 | 16.8 | 59.1 |
| (Korson | 14.9 | 2.9 | 1.5 | 19.3 | | 51.9 | 18.9 | 15.1 | 85.9) |

Figure 1: Result Times of Experiment

# 5 Experimental results

## 5.1 Statistical results

The timing data collected during the experiment is presented in Figure 1 (the results in parentheses are Korson's mean times) and and the bar charts in Figure 2 display the mean modular times versus the mean monolithic times for both Korson and ourselves. Given the size of the effect he discovered, we hypothesized that our results would be similar to Korson's i.e. that the times for modular subjects would be significantly faster than for the monolithic subjects. Surprisingly, however, this did not turn out to be the case: mean syntax time actually took longer for the modular version. Furthermore, the difference between our modular and monolithic mean total times is relatively small: 48 minutes compared to 59.1 minutes. In contrast, Korson's times were 19.3 minutes compared to 85.9 minutes (see Table 1). On average, Korson's monolithic subjects took more than 4 times as long as his modular subjects yet our monolithic subjects took less than 1.3 times as long as our modular subjects. In addition to the mean total time, standard deviation has been calculated to represent the spread in the subjects' times, something Korson did not discuss in his thesis or article [8, 9] other than to say that he had tested for homogeneity of variance using Bartlett's test and that on the basis of test failures for his Experiments 2 and 3 had decided to use the non-parametric Wilcoxon Rank Sum test. Our calculation shows that Korson has a small standard deviation for his modular group, yet an exceptionally large one for his monolithic group: so he was correct in choosing a non-parametric test. On the other hand, the standard deviations

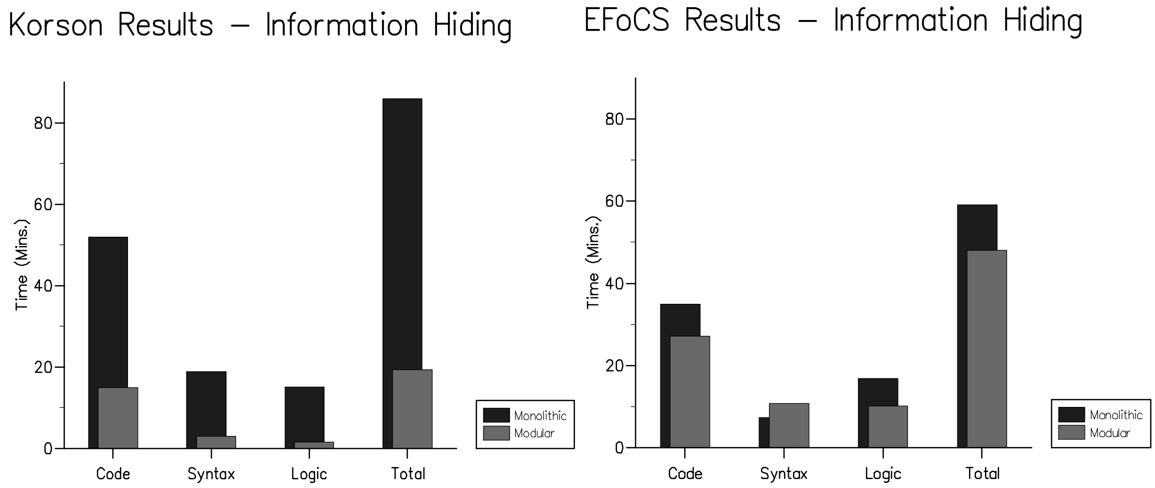

Figure 2: Bar Charts Displaying Total Mean Times

for both our groups were similar, so justifying our use of the t-test (see later).

A possible reason for the relatively poor performance of our modular subjects could be the ability of our subjects relative to Korson's. If our subjects were less able than Korson's, however, much slower results for the modular version would surely have been reflected by much slower results for the monolithic version. As can be seen from Figure 2 this is not the case. Indeed, the opposite has occurred; in 2 out of 3 phases our monolithic times were substantially quicker than the corresponding Korson times, leading to a faster overall mean time by greater than 25 minutes. Possible reasons for this are discussed later (Section 5.5).

| | EFoCS | | Korson | |
|---|---|---|---|---|
| Version | Mean | S. Dev. | Mean | S. Dev. |
| Modular | 48.0 | 25.4 | 19.3 | 8.1 |
| Monolithic | 59.1 | 27.0 | 85.9 | 47.8 |

Table 1: Statistical Values For The Total Time

Korson used a Wilcoxon Rank Sum test in an attempt to reject the null hypothesis, $H_0$: information hiding has no effect on maintainability. (According to Korson, a t-test yielded a similar level of statistical significance.)

The probability that Korson produced his results by chance was $\rho < .001$ and, consequently, $H_0$ was rejected. Though the direction of our result is the same as Korson's, our probability level was calculated at $\rho < .4$ (two-tailed, independent t-test with df = 15, t = -0.870).

This level is so well removed from Korson's significance level and even the minimal level of 0.05 usually required to reject the null hypothesis that it begs the question 'why are our results so strikingly different, especially since Korson claimed his results were replicable?'. As a consequence, an inductive analysis was undertaken to investigate the reasons for this difference, the results of which are discussed in Section 5.4.

## 5.2 Inductive analysis

Traditional statistical approaches require identification of independent, dependent, and random variables with no errors of omission or commission. These approaches usually involve assumptions (often erroneously made) about the parent distributions from which the sample has been collected (for example, the assumption of normality). Furthermore, the statistical methods, for example Pearson's product moment correlation coefficients, tend not to give the reader any real insight into the actual data. An inductive analysis is an experimental paradigm which Brooks and Vezza argue [4] overcomes this problem through the production of facts and rules. Moreover, they argue that while this paradigm does not forbid the use of such traditional statistics, on the condition that they are the basis of a controlled and planned experiment, data gathering should remain as unrestricted as possible in order to allow an inductive analysis. The result of the inductive analysis can be used to supplement statistical meaning and to help explain individual results.

An inductive analysis, which can manipulate both ordered and unordered variables, enables the experimenter to "knowledge engineer to the full; to build a database of experimental results without restriction" [4]. The rule induction system, IRIS [1] based on [11, 7], takes each database variable in turn as the dependent variable. New variables, noted after the experiment, may be introduced to the database, as long as variable values are obtainable.

It is usually the case that the induction data has to be described at different levels of detail while observing the effect on the induction rules. Table 4 in the appendix lists the first level of data before logical groupings took place. Various logical grouping of the numerical attributes were then made, and attempts at splitting the induction database into two separate ones (one for monolithic and one for modular) took place. Before discussing the rules produced by the inductive analysis (see Section 5.4) the variables used are summarised in Table 2. A more detailed description of variables, where required, is provided below.

| Variable | Description | Logical Values |
|---|---|---|
| (1) think | time to note changes to be made to program | 1 .. 3 |
| (2) edit | time to edit changes made in think | 1 .. 3 |
| (3) syntax | time to remove syntax errors | 1 .. 3 |
| (4) logic | time to remove logical errors | 1 .. 3 |
| (5) total 1 | total time to complete task | 1 .. 3 |
| (6) total 2 | total 1 - edit | 1 .. 3 |
| (7) program | type of program subject had | mono, mod |
| (8) numbchars | number of characters subject wrote in think | |
| (9) age | age of subject | |
| (10) sex | sex of subject | m, f |
| (11) position | university status of subject | ug, rs, ra |
| (12) easy | mention of task being easy | y(es), n(o) |
| (13) editor | mention of making good use of editor | y(es), n(o) |
| (14) diff | what caused the most difficulty | 1 .. 3 |
| (15) code | understanding the code | 0 .. 3 |
| (16) learn | learned anything | 1 .. 4 |
| (17) extra | any other comments | 1 .. 5 |
| (18) numbchanges | number of changes unidentified in think | 1, 2 |
| (19) StoRdiff | difference between syntax and working programs | y(es), n(o) |

Table 2: Induction Variables

## 5.3 Induction database

Subjects' times, variables (1) to (6), were graded into three groups. These groups were determined by a) plotting a histogram of the data, b) choosing the 2 best split points on the histogram, and c) grading into the groups. Subjects were categorized by position, variable (11), within the department in order to distinguish between undergraduates (ug), research students (rs) and research assistants (ra). Variables (12) to (17) are concerned with the answers the subject wrote on the questionnaire at the end of the experiment. These were graded into logical groups with as similar meanings as possible. Variable (14) diff was graded 1 - pretest ; 2 - finding the correct places to modify; 3 - remaining mixture of individual answers. Variable (15) code was graded 0 - no understanding; 1 - only the relevant parts; 2 - fairly well; 3 - well or very well. Variable (16) learn was graded 1 - nothing; 2 - read the instructions fully; 3 - mixture of individual answers; 4 - to make modifications the code does not have to be understood. Variable (17) extra was graded 1 - comments in the code not noticed; 2 - no extra comment; 3 - comments in code read but not of any help; 4 - pascal syntax forgotten; 5 - comments in code helped. Subjects' programs were saved after: (i) editing the changes made in the think phase, (ii) removing all syntax errors, (iii) the program was logically correct.

| | Induction Variables | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| subject | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| A | 1 | 1 | 1 | 1 | 1 | 1 | mod | 237 | 26 | m | rs | y | y | 3 | 3 | 1 | 1 | 1 | n |
| B | 1 | 2 | 1 | 1 | 1 | 1 | mod | 58 | 22 | m | ra | y | y | 1 | 1 | 2 | 5 | 1 | n |
| C | 1 | 1 | 1 | 1 | 1 | 1 | mod | 98 | 21 | m | ug | y | y | 2 | 2 | 1 | 1 | 2 | n |
| D | 1 | 1 | 1 | 2 | 1 | 2 | mod | 105 | 22 | m | ug | n | y | 3 | 0 | 3 | 5 | 1 | n |
| E | 2 | 2 | 1 | 1 | 2 | 2 | mod | 116 | 20 | m | ug | y | y | 1 | 3 | 3 | 1 | 1 | n |
| F | 2 | 1 | 2 | 1 | 2 | 2 | mod | 144 | 33 | m | rs | y | y | 2 | 2 | 3 | 2 | 2 | n |
| G | 2 | 1 | 3 | 1 | 2 | 2 | mod | 69 | 27 | m | ra | n | n | 3 | 1 | 3 | 3 | 2 | n |
| H | 3 | 2 | 3 | 2 | 3 | 3 | mod | 176 | 23 | m | rs | n | y | 3 | 2 | 3 | 2 | 2 | y |
| I | 1 | 2 | 1 | 1 | 1 | 1 | mono | 180 | 20 | m | ug | y | y | 2 | 0 | 4 | 2 | 2 | n |
| J | 2 | 3 | 1 | 1 | 2 | 2 | mono | 333 | 19 | f | ug | y | n | 3 | 0 | 4 | 2 | 1 | n |
| K | 2 | 2 | 1 | 1 | 2 | 2 | mono | 679 | 20 | m | ug | y | y | 2 | 2 | 3 | 2 | 1 | n |
| L | 3 | 2 | 1 | 1 | 2 | 2 | mono | 652 | 22 | m | ug | y | n | 2 | 0 | 1 | 4 | 1 | n |
| M | 2 | 3 | 2 | 1 | 2 | 2 | mono | 562 | 22 | m | ug | y | n | 2 | 1 | 3 | 4 | 1 | y |
| N | 2 | 3 | 3 | 1 | 3 | 2 | mono | 429 | 33 | m | rs | y | n | 2 | 1 | 1 | 2 | 2 | n |
| O | 2 | 3 | 1 | 2 | 2 | 2 | mono | 445 | 22 | m | ug | y | y | 3 | 1 | 3 | 2 | 1 | y |
| P | 3 | 3 | 1 | 3 | 3 | 3 | mono | 661 | 26 | m | ra | n | y | 3 | 0 | 1 | 2 | 1 | y |
| Q | 2 | 2 | 2 | 3 | 3 | 3 | mono | 644 | 33 | m | ug | y | n | 3 | 3 | 2 | 2 | 1 | y |

Table 3: Final Data for Induction Package

Saving these programs allowed the introduction of two new variables. The number of changes unidentified in the think phase, variable (18), was divided in to two groups, 1 (0 or 1 change) and 2 (2 or more changes) - calculated by comparison of the edited version to the completed, working version. Saved programs were also examined to determine if any changes were made from the syntax phase to the logic phase, variable (19) - yes or no. In addition to the experiment times, times for the pretest were also categorised as with variables (1) to (6) as in Table 2; for example, pretest variable (1) was the think time for the pretest. The resulting database (shown in Table 3) for the subjects who completed the experiment contains the final data used by the induction package.

## 5.4 Induced rules and interpretation

Rules were produced for each variable (including the pretest variables) in an attempt to explain the difference between our findings and Korson's. Many of the rules induced were highly fragmented, revealing few or no patterns in the data, which is indicative of several sources of variability, overlapping in their influence. The rule induction package produced two rules, however, which drew attention to two interesting patterns:

| | think | syntax | logic | total | | think | syntax | logic | total |
|---|---|---|---|---|---|---|---|---|---|
| Subject A | 9 | 1 | 1 | 11 | Subject I | 17 | 3 | 2 | 22 |
| Subject B | 26 | 7 | 18 | 51 | Subject J | 16 | 2 | 0 | 18 |
| Subject C | 17 | 2 | 2 | 21 | Subject K | 20 | 1 | 1 | 22 |
| Subject D | 13 | 11 | 25 | 49 | Subject L | 17 | 1 | 7 | 25 |
| Subject E | 17 | 1 | 30 | 48 | Subject M | 17 | 1 | 1 | 19 |
| Subject F | 25 | 5 | 8 | 38 | Subject N | 25 | 1 | 13 | 39 |
| Subject G | 22 | 1 | 21 | 44 | Subject O | 15 | 1 | 0 | 16 |
| Subject H | 23 | 4 | 19 | 46 | Subject P | 14 | 1 | 18 | 33 |
| | | | | | Subject Q | 10 | 4 | 1 | 15 |

Figure 3: Result Times of Pretest

1. A suggested relationship was found between the total times taken for the experiment and the pretest. All 9 of the monolithic subjects appeared in the top 12 places when ranked by pretest timings (see data in Figures 1 and 3).

2. A high number of changes missed in the think phase in the experiment suggested either an excessively high syntax or logic time.

The first rule or pattern can be interpreted as an ability effect (Brooks [5] and Curtis [6] have reported on empirically discovered programming ability differences ranging from factors of 4 to 1 to 25 to 1). Also note that our modular Subject A, who finished both the fastest on the pretest and the experiment, was known to be a person with high ability. If we accept the ability effect interpretation, an underlying problem with the replication is revealed as the majority of the higher ability participants were assigned to the monolithic task (remembering participant assignment was random and not based on pretest timings). Korson, on the other hand, claimed to have found no such ability effect with the exception of one subject who consistently finished first in his series of experiments. So the suggested ability effect is one source of variability between the two experiments.

Only subjects B, Q, and O deviate from this ability interpretation in the sense of a performance mismatch between pre-test and experiment. Comments made during debriefing helped explain two of the mismatches. Subject B, who had the longest pre-test time but the second fastest modular time, commented that the instructions were long and difficult to read, that he had not taken enough time to read pre-test instructions, but that he had learned from the pre-test. Subject Q, who took the longest with the monolithic code despite being the second fastest on the pre-test, inadvertently deleted a line during the experiment and was taking so long during the logic phase that one of the monitors advised him of his problem. Subject Q also commented that he had not read the instructions fully to begin with. Subject O seems a genuine anomaly: no substantive evidence emerged to explain a good pre-test performance combined with a relatively poor performance on the experiment. This subject, however, did comment on a difficulty having arisen in the experiment due to having read data into the wrong variable.

The second rule or pattern draws attention to the fact that subjects who missed a number of changes (2 or more) in the think phase had either a higher syntax or logic time. What is the significance of this? Firstly, of the 4 subjects whose syntax was highest, all made edits correcting semantics during the syntax phase, specifically against instructions. Significantly, 3 out of the 4 subjects performed their tasks on modular code. These 3 subjects totalled 73 minutes of syntax between them and are the main reason that the mean syntax time for the modular version is greater than the mean syntax time for the monolithic version. Secondly, it tends to indicate that there exists a certain element of artificialness surrounding the experiment. The timing of one phase appears to be overlapping into the timing of the following one, creating potentially inaccurate timings of think, syntax, and logic. Explanations for this include: (a) the experiment itself was to blame due to poor design and the environment of think, edit, syntax and logic phases being artificial, or (b) the monitors who timed the subjects (one monitor to 4 or 5 subjects) were not strict enough in controlling when a subject was ready to move from one phase to the next.

## 5.5 Discussion

Korson required that the various versions of subjects' programs were saved as they worked through the phases of the experiment, but he does not present any results or analysis from such important data. We, however, did carry out analysis of these programs. The Unix utility diff allowed comparison of the saved syntax program to the saved logic program and gave an indication of what caused the subject extra time, if any, to debug. As shown in Table 3, only 5 subjects actually had any difference between the two programs, and this tends to indicate that subjects who didn't have any difference should have had relatively low logic times: all they appeared to be doing was running through the testing procedure. In reality this was not the case. Logic times for those in the modular group with no difference between the syntax and working programs (variable (19)) ranges from 1 to 20 minutes, while the equivalents in the monolithic version only range from 1 to 5 minutes. Why? One explanation of this variability is the procedure the subjects followed to test the program. As in Korson's experiment, there were three distinct ways of carrying out a full test of the program: (i) using a single hot-key, (ii) using a series of hot-keys, and (iii) manually entering test data. The optimal results of each way are as follows, (i) 1m 15s, (ii) 55s, and (iii) 2m 30s.[2] In addition, if the subject was to reread the instructions on the testing procedure another minute or so may have been spent. Consequently, any time between 1 and 5 minutes may be regarded as explainable. Perhaps, it may have been better practice for the monitor to test the program for the subject in attempt to reduce the significant variability that appears to have arisen. From a hypothetical viewpoint, what would happen to the results if all these subjects whose saved syntax program was no different to the saved logic program were given a logic time of 1 minute i.e. if they were given logic times assuming that they had performed the testing procedure in the minimum time possible?[3] The answer is there would be still be no significant difference between modular and monolithic subjects with $\rho < .259$ (two-tailed, independent t-test with df = 15, t = -1.172), although the difference between the total time means does widen so that on average our monolithic subjects took 1.38 times as long as our modular subjects. We have conservatively used two-tailed significance levels but even if one-tailed levels had been used our results would remain well outside the minimal $\rho$ level of 0.05 usually required.

Another source of variability was the approach the subjects took towards making the modifications. It was noted by monitors during the experiment that several subjects attempted to follow the execution flow of the program, rather than just concentrating on finding the modifications to be made. As a consequence, subjects understanding of the code ranged from not at all to very well. In the case of subjects I and J, the two fastest finishers in the monolithic version, both subjects explained in the subsequent debriefing questionnaire that they had no understanding of the code and both subjects made the added comment that no understanding of code was necessary to modify it. (This latter comment is in agreement with the feelings of some of the pilot study subjects - see Section 3.3). On the other hand, examination of the listings with the written modifications showed that subjects G and H had indeed attempted to follow the flow of program control. These subjects took the longest time to finish the tasks for the modular version. Yet another modification variation was the fact that subjects L and O both wrote procedures for the monolithic version, an action which could have taken extra time.

As already mentioned the monitors may not have been strict enough compared to the monitors of Korson's experiment. This may be one of the reasons for the overlapped

[2] As performed by a person competent with both the testing procedure and working program.

[3] Bringing them into line with the majority of Korson's subjects.

phases. However, a comment made by one of the subjects who didn't complete the actual experiment said the nature of the experiment made him take much longer than if he had been able to implement the modifications in the way he was accustomed to; for example compiling, running and experiencing the program first and then making changes. The phased approach was artificial to him. Subjects K and P made unsolicited comments to the same effect. Other participants (B and Q) commented that the instructions were too long and difficult to read resulting in an additional delay.

# 6 Some replication problems

When conducting a replication a decision has to be made as to its purity. It is rare not to have ideas on how the experimental recipe can be improved upon. As stated earlier, making major improvements to the experimental methods was resisted as the main interest lay in trying to replicate. Minor recipe-improving changes that were implemented have already been described (Section 4.1). We list here various suggestions for improvement, which were not implemented, to highlight the problem of deciding the scale of any recipe improving: improve program layout and commentary, remove biased comment in the modular program, replace the global variable in the modular program to fully implement information hiding, and reduce the size of the experimental instructions given to subjects. One can never be sure of obtaining similar results had some or all of these improvements also been implemented.

Even though Korson's work is reported in his doctoral thesis, several details were absent from his reporting of the experiment which introduced uncertainty into our replication study. Firstly, Korson did not present any information as to the number of monitors he used: we found a ratio of one monitor to 4 or 5 subjects was only just satisfactory and had we known that Korson had worked with a better monitor-subject ratio we would have taken steps to provide additional monitors. Secondly, with hindsight, we could have ensured subjects had properly completed each phase by instructing monitors accordingly. From Korson's written instructions it would seem that subjects were allowed to decide when to move on but it may well have been that Korson's monitors applied additional control. We say this because we are genuinely surprised that Korson did not report any problems with subjects moving prematurely between phases in all the preliminary and primary experiments he conducted. Thirdly, Korson gave his subjects three independent methods of testing the program for correctness, yet did not publish the likely timings involved for these different methods: providing variability in the method of testing may have been a source of variability in both his and our results. Fourthly, he does not present any information to explain why certain subjects took so long to complete the monolithic version, something which might have been of importance to researchers interested in replication. Finally, criticism can be made about the criteria Korson set for subject selection: retrospectively, we feel that it was perhaps insufficiently specific and as such we suspect that one or two subjects whose experience or ability was not of the required level may have passed the selection criteria for our replication study.

We must not, however, be overly critical of Korson. With the benefit of hindsight these points are easy to make as we had to analyse our results in full in order to explain the vast difference between these and Korson's results. As a consequence of our experiences of attempting a replication study we have produced a report [3] giving some guidance as to the level of detail necessary to better enable others to attempt a meaningful replication.

# 7 Conclusions

Our results were markedly different to those of Korson and therefore do not support his claims of replicability. Korson chose to rely on a quantitative analysis of timings. In contrast, our approach was to combine traditional methods with an inductive analysis: this proved to be vital in understanding the behaviour of our subjects and we conclude that such a combined approach must be viewed as necessary if future software maintenance experiments are to be properly analysed. There is no getting away from the human component.

We found evidence for an ability effect, a major source of variability. There were several other influences on subject timings. Some subjects failed to identify all the necessary changes during the think phase. The simple act of incorrectly deleting a line took some time to be spotted by one subject. Two subjects even began to proceduralise the monolithic code. Several subjects commented on the artificiality of the experiment: one in particular wanted to engage in a prototyping approach and was unhappy at having to fall-in with the constraints of the experiment.

Of great importance was our discovery that the two fastest finishers on the monolithic code mentioned that they had no understanding of the code and that a genuine understanding was not necessary to perform the modifications. In contrast, the two slowest finishers on the modular version had taken some trouble to try and understand the program. The former behaviour we call pragmatic maintenance - constraining ones understanding to a minimum. So we also conclude it vital to observe a number of maintenance activities to understand the role of such pragmatic maintenance.

If generalizable results are to be realised, we are left with the inevitable conclusion that software engineering experiments must be externally replicated at research centres around the world and that traditional analysis methods must be combined with inductive analysis techniques. Computer scientists must behave more as conventional scientists.


## Acknowledgments

The authors would like to thank all those who took part in the replication and Dave Whittington for technical assistance.


## References


[1] G Arisholm. Iris integrated rule induction system. *Dept. of Computer Science, University of Strathclyde, Scotland*, 1987.

[2] V R Basili, R W Selby, and D H Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, 12(7):733–743, 1986.

[3] A Brooks, J Daly, J Miller, M Roper, and M Wood. Replication's role in experimental computer science. Research report EFoCS-3-94, Department of Computer Science, University of Strathclyde, Glasgow, 1994.

[4] A Brooks and P Vezza. Inductive analysis applied to the evaluation of a cal tutorial. *Interacting with Computers, The Interdisciplinary Journal of Human-Computer Interaction 1*, pages 159–170, 1989.

[5] Rueven E Brooks. Studying programmer behavior experimentally: The problems of proper methodology. *Communications of the ACM*, 23(4):207–213, April 1980.

[6] Bill Curtis. Measurement and experimentation in software engineering. *Proceedings of the IEEE*, 68(9):1144–1157, September 1980.

[7] A Hart. Experience in the use of an inductive system in knowledge engineering. In M A Bramer, editor, *Research and Development in Expert Systems*, pages 117 – 126. Cambridge University Press, 1985.

[8] T D Korson. *An Empirical Study of the Effects of Modularity on Program Modifiability*. PhD thesis, College of Business Administration, Georgia State University, 1986. UMI Order Number 8718376.

[9] T D Korson and V K Vaishnavi. An empirical study of the effects of modularity on program modifiability. In E Soloway and Iyengar S S, editors, *Empirical Studies of Programmers: First Workshop*, pages 168–186. Ablex Publishing Corporation, 1986. A Volume in the Ablex Human/Computer Interaction Series.

[10] B Lientz and E Swanson. *Software Maintenance Management*. Addison-Wesley, 1st edition, 1980.

[11] J R Quinlan. Decision trees and multivalued attributes. *Machine Intelligence Workshop*, 1985.

# A Raw data

Table 4 displays the data in its raw form before any manipulation or logical groupings were introduced. Answers to questions from the questionnaire have, obviously, been logically grouped to similar replies: this is the only way to represent the data in a form IRIS may use.

| | Induction Variables | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| subject | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| A | 15 | 4 | 1 | 2 | 22 | 18 | mod | 237 | 26 | m | rs | y | y | 3 | 3 | 1 | 1 | 0 | n |
| B | 14 | 13 | 7 | 4 | 38 | 25 | mod | 58 | 22 | m | ra | y | y | 1 | 1 | 2 | 5 | 1 | n |
| C | 16 | 6 | 1 | 10 | 33 | 27 | mod | 98 | 21 | m | cs4 | n | y | 2 | 2 | 1 | 1 | 2 | n |
| D | 22 | 3 | 1 | 20 | 46 | 43 | mod | 105 | 22 | m | cs4 | n | y | 3 | 0 | 3 | 5 | 0 | n |
| E | 39 | 14 | 3 | 14 | 70 | 56 | mod | 116 | 20 | m | cs3 | n | y | 1 | 3 | 3 | 1 | 0 | n |
| F | 34 | 7 | 19 | 5 | 65 | 58 | mod | 144 | 33 | m | rs | y | y | 2 | 2 | 3 | 2 | 2 | n |
| G | 31 | 2 | 29 | 1 | 63 | 61 | mod | 69 | 27 | m | ra | n | n | 3 | 1 | 3 | 3 | 5 | n |
| H | 46 | 16 | 25 | 25 | 112 | 96 | mod | 176 | 23 | m | rs | n | y | 3 | 2 | 3 | 2 | 2 | y |
| I | 15 | 15 | 4 | 5 | 39 | 24 | mono | 180 | 20 | m | cs4 | y | y | 2 | 0 | 4 | 2 | 3 | n |
| J | 32 | 30 | 6 | 3 | 71 | 41 | mono | 333 | 19 | f | cs2 | y | n | 3 | 0 | 4 | 2 | 1 | n |
| K | 38 | 14 | 2 | 4 | 58 | 44 | mono | 679 | 20 | m | cs3 | y | y | 2 | 2 | 3 | 2 | 1 | n |
| L | 44 | 21 | 1 | 4 | 70 | 49 | mono | 652 | 22 | m | cs4 | y | n | 2 | 0 | 1 | 4 | 0 | n |
| M | 28 | 30 | 11 | 11 | 80 | 50 | mono | 562 | 22 | m | cs4 | y | n | 2 | 1 | 3 | 4 | 1 | y |
| N | 36 | 47 | 22 | 1 | 106 | 59 | mono | 429 | 33 | m | rs | y | n | 2 | 1 | 1 | 2 | 3 | n |
| O | 29 | 27 | 3 | 29 | 88 | 61 | mono | 445 | 22 | m | cs2 | y | y | 3 | 1 | 3 | 2 | 0 | y |
| P | 55 | 26 | 2 | 36 | 119 | 93 | mono | 661 | 26 | m | ra | n | y | 3 | 0 | 1 | 2 | 0 | y |
| Q | 38 | 22 | 15 | 58 | 133 | 111 | mono | 644 | 33 | m | cs2 | y | n | 3 | 3 | 2 | 2 | 1 | y |

Table 4: Raw data for induction package

# B Debriefing questionnaire

```
COMMENTS SHEET

Thank you for participating. Before you leave, please give a few personal
details and your comments.

Personal details

Name ______________________________
Age ____
Sex ____
Position (eg 2nd year CS) ____________________
Qualifications if you are a graduate (eg BSc Comp Sci 2(i)) ____________

Comments

1. Did you find the task easy?


2. Did you make good use of the editor?


3. What caused you the most difficulty?


4. How well do you understand the code?


5. Have you learned anything? If so, what?


Any other comments?
```