

Replication of experimental results in software engineering

Andrew Brooks, John Daly*, James Miller, Marc Roper and Murray Wood
Dept. Computer Science, University of Strathclyde,
Livingstone Tower, Richmond Street, Glasgow G1 1XH, Scotland
(email: andy@cs.strath.ac.uk)

ISERN-96-10

Abstract

Carrying out empirical studies is widely held to be of importance. A view less widely held is that experiments should be replicated externally to verify and validate the original results.

This paper serves two main functions. First, the need for external replications is established. The role of replication in experimental software engineering is discussed. Without the confirming power of external replications, results in experimental software engineering should only be provisionally accepted, if at all. An extension to the framework for experimentation in software engineering by Basili et al [5] is proposed to differentiate between the various kinds of internal and external replication and their powers of confirmation and to allow a better appreciation of the context of a piece of empirical work.

Second, this paper presents a concrete example 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 by Korson [32, 33] 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 toward making the required modifications, and possible deficiencies of subject monitoring. Other sources of variability are also discussed.

It is concluded that external replications, combined with inductive analysis techniques, have an important if not vital part to play in the realization of generalizable results.

*Daly is now with the Fraunhofer Institut (IESE), Sauerwiese n 6, D-67661 Kaiserslautern, Germany

1 Introduction

Carrying out empirical work to understand the problems of software engineering is widely held to be of importance [20, 17]. There are, however, inherent problems in drawing conclusions from single studies, especially those with human subjects. (Though this paper is intended primarily for subject-based experimentalists, computational experimentalists should also find it relevant.) Every stage of an experiment from background reading through to result interpretation through to the writing of the report and its conclusions is prone to error. These are but some of the problems: (i) The background may not be properly researched and the empirical study may be addressing an unimportant issue. (ii) Inappropriate methods may be used. For example, with subject-based experiments, strictly controlled laboratory experiments are usually best supplemented with more qualitative or ethnographic forms of experimentation (as we shall see). Also, insufficient sampling can lead to a situation where a real, even quite sizeable, effect has little chance of being revealed as significant. (iii) Errors of commission or omission may be made or experimental variables may be incorrectly classified. (iv) Statistical procedures may be misapplied. For example, parametric statistics may be applied to non-normal data. (v) Alternative interpretations may not be presented. With subject-based experimentation, it is likely that more than one interpretation can be placed on the data. We agree with Collins [13] who regards an experiment to have been incompetently performed if some alternative explanation for the data has been overlooked. Failure to discuss alternative interpretations of data can prevent a reviewer performing a meaningful meta-analysis of the research area. Brooks and Vezza [8] is an example of a paper providing the reader with alternative interpretations. (vi) Missing details may prevent the reader from forming their own view of the worth of the data; for example, error estimates may not be provided for some or all of the critical measures or raw data may be crudely summarised when it could have been presented in full. (vii) The experimental methods may be poorly reported so that it is impossible to perform an external replication study. For example, instructions and task materials given to subjects may not be fully reported or may otherwise be unobtainable. Unfortunately, numerous empirical studies in the software engineering literature are lacking in this respect. Other authors have criticised poor reporting, for example, Basili et al [5] and MacDonell [38]. It may well be that the required quality and completeness of reporting of subject-based experimentation can be found only in theses for higher degrees or in a series of highly detailed technical reports. (viii) Conclusions may not be justified from the analysis or may simply be incorrectly expressed in terms of the null hypothesis. (ix) There is the problem of confirmation bias where experimenters discredit or reinterpret information that contradicts the hypothesis they hold: but this phenomenon appears to be more complicated than originally thought [18].

Because of the problems above, scientists demand that experimental results are externally

reproducible i.e. that an independent group of other researchers can repeat the experiment and obtain similar results. External replication is an alleged cornerstone of modern scientific disciplines.

Generally speaking we feel that work can often be regarded as research by advocacy, that is the work is managed to support an idea or tool and lacks the rigour characteristic of empirical studies in more established areas of modern engineering, scientific, and social science disciplines. The *yet another system* paper is all too common. Even empirical papers published in respected journals and proceedings cannot escape criticism: see, for example, some of the criticisms by Kitchenham et al [31]. We share Glass's view of the advocacy problem [21]:

Advocacy has been with us for so long that it just doesn't seem to occur to anyone that there's a component missing from our research.

Yet few doubt the need for software engineers to work from principles and guidelines in which the professional community has high confidence, all the more so if the application is safety critical. Few doubt the need to solve the problems of software maintenance and improve software economics. Few doubt the need for solid, empirical foundations to claims about software. As stated in excerpts from a 1990 report by the Computer Science and Technology Board [19],

So-called system maintenance may constitute up to 75 percent of a system's cost over its lifetime ...

and

In the absence of a stronger scientific and engineering foundation, complex software systems are often produced by brute force ...

Leveson [35] has also stressed the role of experimentation,

Building a scientific foundation of software engineering requires both building mathematical models and theories and performing carefully design experiments.

So it would seem that after a generation of software engineering research (and a generation of educating students), the foundation is not strong and serious software problems persist. It is our view that a fundamental reappraisal is fully justified. Perhaps little of any real import has been learned since the original Garmisch conference [43]. Potts [47] has cast similar, but not as harsh, doubts about the practical difference software engineering research has made over the last 25 years. Perhaps even the established principles and guidelines regarding structured design (so well written about by Page-Jones [44] whose textbook has sold over 100,000 copies) should be reappraised. Should we be surprised if even these first-order principles and guidelines are found wanting? Much is to be gained, therefore, by critical examination of previous empirical studies,

by externally replicating the experiments of others and by performing, verifying, and validating new experiments with modern software technologies.

A general discussion of the role of replication in the scientific ideology and in practice is presented as a basis for a discussion of the role of replication in experimental computer science. Various kinds of internal and external replication are differentiated. Internal replications are undertaken by the original experimenters and may be published as one paper or as a series: external replications are undertaken by independent researchers who seek to check and improve on the findings of other researchers. It should be noted that this article is not concerned with the notion of replication applied to an individual experimental design. (For example, some experiments have built-in replication with more than one programming team tackling the same project, so as the scope of the examination increases across these teams, the more convincing an individual study's conclusions become.) The notion that the different kinds of replication carry different powers of confirmation is developed. Without the confirming power of external replications, it is argued that many principles and guidelines in software engineering arising from work in experimental computer science should be treated with caution and should only be provisionally accepted, if at all. Internal replications carry less confirming power. The reader should be left in no doubt of the importance of performing external replications.

An extension to the framework for experimentation in software engineering by Basili et al [5] is proposed to more fully differentiate between the various kinds of replication and their powers of confirmation: this should allow researchers and referees alike to better appreciate the context of a piece of empirical work.

To provide a concrete example of an external replication, 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 by Korson [32, 33] 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.

When our interest in performing an external replication had been established, we spent several weeks reviewing a number of doctoral dissertations in the computing field where the emphasis was on experimental investigations. Korson's experiment was chosen as our first attempt at performing an external replication on several counts:

1. We were unsure of the empirical foundation of the guideline that modular software is better and Korson's work claimed to show a large positive effect for modularity. Lord Rutherford has been quoted [39] as saying

If your experiment needs statistics, you ought to have done a better experiment.

Korson's experiment seemed to be one of these better experiments as the raw data and histogram plots alone are enough to convince the reader of the result without resorting to the use of inferential statistics. It was unnecessary to perform a formal, statistical power analysis to check that there was a reasonable probability of detecting a significant finding in our replication. (Fenton [17] cites relatively early work which claims to show no strong evidence to support the widely held beliefs about the benefits of modularity.)

2. Korson's thesis appeared to contain sufficiently detailed descriptions of experimental materials and procedures to allow a replication attempt. The appendices to his thesis contained complete code listings and lists of instructions given to subjects. The body of his thesis provided criteria for subject participation and a description of the experimental design and processes.

3. 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

4. On a first reading, Korson's work qualified as well performed empirical work (unlike much empirical work in software engineering which can be dismissed on a first reading). Also, we could find no criticisms of it in the literature: Banker et al [3] summarise Korson's findings without criticism.

5. If Korson's work bears up to detailed scrutiny and is 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).

So, despite all the problems of empirical work noted earlier and the resulting wariness one should have over devoting the resources necessary to performing an external replication, we took the view that if any study was to be externally replicated, Korson's work was a good place to start.

The results of our external replication, however, were markedly different from those of the original. Though our results were in the same direction, we found no significant difference between the average times taken to maintain modular and monolithic code whereas Korson had found a ρ value of .001. So what has caused this inconsistency? Should yet another experiment be performed?

To break the paradox of the experimenters' regress (see Section 2.4), 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 toward making the required modifications, and possible deficiencies of subject monitoring. Other sources of variability are also discussed. This paper extends the analysis presented in an initial report of the external replication given by Daly et al [16].

Drawing from our experience of having performed an external replication, guidance is also given on both the scale of any recipe-improving when attempting a replication and on the level of reported detail required, as a list of desiderata, to help others to perform a meaningful replication.

It is usual to arrive at an appreciation of empirical work after reading the published reports. We have found, however, that performing an external replication provides a deep appreciation of the strengths and weaknesses of a piece of work.

It is concluded that external replications, combined with inductive analysis techniques, have an important if not vital part to play in the realisation of generalizable results in software engineering.

2 Replication

2.1 The ideology

Subjecting theory to experimental test is a crucial scientific activity. Popper [45], however, explains that researchers must be sure of their results before reporting them, stating

We do not take even our own observations quite seriously, or accept them as scientific observation, until we have repeated and tested them.

Coupled with this advice, modern scientific ideology now also demands that experimental results are replicable by an external agency. For example, as Lewis et al [36] rightly claim

The use of precise, repeatable experiments is the hallmark of a mature scientific or engineering discipline.

Goldstein and Goldstein [22] take this one step further, stating

We now take for granted that any observation, any determination of a 'fact', even if made by a reputable and competent scientist, might be doubted. It may be necessary to repeat an observation to confirm or reject it. Science is thus limited to what we might call 'public' facts. Anybody must be able to check them; experimental observations must be *repeatable*.

Not only must the researcher make his work repeatable, however, some even regard it as being beholden on the scientific community to execute replications just to verify the experimental results, as we ourselves now do. For example, Huxley [28] has stated,

In scientific inquiry it becomes a matter of duty to expose a supposed law to every possible kind of verification ...

Huxley [28] also notes,

And in science, as in common life, our confidence in a law is in exact proportion to the absence of variation in the result of our experimental verifications.

So the greater the number of experimental verifications the better, at least until such time additional verifications carry no further power of confirmation. We agree with Curtis [15], when he says

... results are far more impressive when they emerge from a program of research rather than from one-shot studies.

But Curtis appears to be referring to internal replications and does mention the greater confirming power of external replications.

Broad and Wade [7], in their description of the scientific ideology, consider replication to be the third check in verifying scientific claims, the first two being the peer review system that awards research grants and the journal refereeing that takes place before publication. They also describe the ideal of reporting experiments as follows,

A scientist who claims a new discovery must do so in such a way that others can verify the claim. Thus in describing an experiment a researcher will list the type of equipment used and the procedure followed, much like a chef's recipe. The more important the new discovery, the sooner researchers will try to replicate it in their own laboratories.

Replication is also concerned with the way the original hypothesis is expressed. As Smith [53] has stated,

Replication does two things: first, it tests the linguistic formulation of the hypothesis; second, it tests the sufficiency of the explicit conditions for the occurrence of the phenomena.

One need not go as far as performing a replication to discover problems with the linguistic formulation of a hypothesis. For example, Henry and Humphrey [25] state in their conclusions that the "experiment supports the hypothesis that subjects produce more maintainable code

with an object-oriented language than with a procedure-oriented language” when in fact their subjects were asked to make modifications to an object-oriented system and a functionally equivalent procedure-oriented system which had been originally developed by a single graduate student. (A correct statement of the hypothesis is given earlier in their paper.)

Regarding the sufficiency of the explicit conditions, an example would be where criteria for subject participation in a software engineering experiment may be insufficiently specific and, as a result, the replication yields different results owing to variability between subjects.

Concerning a particular flawed study in psychology which was accepted as being valid for a long time, Broad and Wade [7] wrote,

Why did nobody helping to raise generations of undergraduates ... replicate the study?

Such a question could equally as well be addressed to many educators of computer science students regarding numerous studies whose results are communicated apparently quite uncritically to students.

Also, given the human component and the rich variety of software and hardware technologies, it surely is beholden on the community to perform many, many, such verifications.

Much is said and written about quality control in software development (e.g. Card [10]). It is ironic, to say the least, that the quality control mechanism of replication, especially external replication, is so little practised amongst those doing the science behind the engineering. There is an additional irony: because of the current state of software development practice, N-version programming has been suggested as a fault detection and recovery mechanism (see, for example[30]). We know so little about doing it right, we end up replicating system functionality across several programs. It is surely better to use the tactic of replication much earlier.

A stronger scientific and engineering foundation can only come about if the research community becomes more involved in external replications. Before presenting our external replication, we discuss the idea of replication from a number of viewpoints.

2.2 Frequency of actual replication studies

In schools, colleges, and universities, replication studies are performed daily: but such studies are usually scaled-down versions, are performed by students in the act of learning, and have no confirming power. As Collins [13] notes,

As more becomes known about an area however, the confirmatory power of similar-looking experiments becomes less. This is why the experiments performed every day in schools and universities as part of the scientific training of students have no confirming power; they are not *tests* of the results they are supposed to reveal.

Those employed in research rarely perform replication studies. Again, as Collins [13] notes,

For the vast majority of science, replicability is an axiom rather than a matter of practice.

Broad and Wade [7] also draw attention to the lack of replication work by stating:

How much erroneous ... science might be turned up if replication were regularly practiced, if self-policing were a more than imaginary mechanism?

Broad and Wade [7] reckon that the Simpson-Traction replication¹ was

... probably one of the very few occasions in the history of science in which the philosopher's ideal of replicability has been attained.

Since Broad and Wade's remark was made, there has been the saga of cold fusion. Many laboratories around the world tried to repeat the cold fusion experiment by Pons and Fleischmann — see Close [12] or Amato [1]. Ordinarily, no scientist would have dreamt trying to replicate a poorly reported experiment. The lure of cheap, relatively pollution free energy in abundance, was an exceptional motivation.

Regarding the frequency of external replication work of software engineering experiments, we have been unable to find examples. Two recent reference works [41, 40] do not discuss external replication nor cite examples (though [40] provides a good discussion of independent verification and validation of software by an independent group different from the software development team). Reviews in the area of software testing and maintenance where empirical work is relatively commonplace make no mention of external replication work: Roper's [49] selected annotated bibliography of software testing and an investigation of the characteristics of empirical software maintenance studies between 1980 and 1989 by Sharpe et al [51].

A particular study which should have been externally replicated soon after its results were published was by Shneiderman et al [52] whose experiments called into question the utility of flowcharts. According to Scanlan[50], the work by Shneiderman et al led to the decline of flowcharts as a way to represent algorithms. Scanlan, however, identified fatal flaws with the methods used by Shneiderman et al; for example, time was not a measurable dependent variable (the subjects were all given as much time as they required). Scanlan designed a new experiment using time as a dependent measure and claimed “my experiment shows that significantly less time is required to comprehend algorithms represented as flowcharts”. Such work on flow-charts is typical of software engineering research where empirical work is either poor or if done well is not followed up by external replications to confirm the generalizations.

¹According to Broad and Wade, in 1961, Simpson had Traction watched while Traction unsuccessfully tried to repeat a biochemistry experiment concerned with protein synthesis.

As stated earlier, this article is not concerned with the notion of replication applied to an individual experimental design. Internal replications carried out as part of a program of research are relatively commonplace. Basili et al [5] provide details and examples of both these forms of replication. Some internal replications are not even reported as they were undertaken to improve the sensitivity of the results and the published report is written as if there was just a single experiment.

It is the rarity of external replication in software engineering research that is the major concern.

2.3 Defining replication

Care must be taken, however, to clarify what is meant by replication. The Universe is forever changing. Human observers and subjects are unique (Brooks [9] and Curtis [15] report on empirically discovered programming ability differences ranging from 4 to 1 to 25 to 1). The number of measurements that can be made to describe the experimental setting is without end. (An art of experimental science is in making neither errors of commission or omission.) Accuracy of observations can always be improved upon until the Uncertainty Principle becomes important. Strictly speaking, it is more correct to talk of partial replication and the goal of performing as near exact replication as possible. Exact replication is unattainable.

According to Broad and Wade, exact replication is an impractical undertaking because the recipe of methods is incompletely reported, because to do so is very resource intensive, and because credit in science is won by performing original work. They do, however, draw attention to the important activity of improving upon experiments. They state,

Scientists repeat the experiments of their rivals and colleagues, by and large, as ambitious cooks repeat recipes — for the purpose of *improving* them... All will be adaptations or improvements or extensions. It is in this recipe-improvement process that an experiment is corroborated.

Baroudi and Orlikowski [4] qualify this and note

Where a study fails to reject a null hypothesis due to low power, conclusions about the phenomenon are not possible. Replications of the study, with greater power, may resolve the indeterminacy.

(These authors provide a useful guide of the issues surrounding statistical power in inferential statistics.)

2.4 Experimenters' Regress

As noted by Collins[13], a paradox exists for those who want to use replication as a test of the truth of scientific claims:

The problem is that, since experimentation is a matter of skillful practice, it can never be clear whether a second experiment has been done sufficiently well to count as a check on the results of the first. Some further test is needed to test the quality of the experiment — and so forth.

Even though we failed to replicate Korson's results, we will show how an inductive analysis of our data established possible interpretations of the differences between both sets of results. The inductive analysis, by yielding an understanding of the processes involved, resolved the paradox of the experimenters' regress: we found no need to perform further experiments to resolve the conflicting results.

3 Framework

As stated earlier, this article is not concerned with the notion of replication applied to an individual experimental design. What we mean by internal replication is where the original experimenters have carried out a series of evolutionary experiments which may be published as one paper or as a series. Basili et al[5] cite examples of internal replications. Internal replications have some confirmatory power but it is less than that achieved by external replications.

By external replication we mean published experiments which are repeated by experimenters who are independent of those who originally carried out the empirical work. Exact replication is unattainable, so it is important to consider and categorize the differences.

First, researchers must consider the experimental method. Should a similar or alternative method be used? A basic finding replicated over several different methods carries greater weight. As Brewer and Hunter[6] have stated,

The employment of multiple research methods adds to the strength of the evidence.

It may be more important, however, to confirm first the original results. Should the method be improved by, for example, debriefing subjects after the formal experiment is over? Such debriefings, as we shall see for our Korson replication, can provide insights into the processes involved. This type of improvement does not compromise the integrity of the replication.

Second, researchers must consider the task. Should a similar or alternative task be used? A basic finding replicated over several different tasks carries greater weight. As Curtis [15] has stated,

When a basic finding ... can be replicated over several different tasks ... it becomes more convincing.

It may be more important, however, to confirm first the original results. Should the task be improved by, for example, making it more realistic?

Third, researchers must consider the subjects. Should a similar or alternative group of subjects be used. A basic finding replicated over several different categories of subjects carries greater weight. It may be more important, however, to confirm first the original results. Should the group of subjects be improved by, for example, by using more subjects or more stringent criteria for participation?

An experimentation framework for software engineering has been established by Basili et al [5] but this framework must be explicitly extended to incorporate external replication and its various forms where method, task, and subjects can each be either *similar*, *alternative*, or *improved*. (We believe it unnecessary at this stage to work with more detailed categorizations.) These method, task, and subjects categories could also be applied to internal replications. Of course, if too many alternatives are used or if the scale of any recipe-improving is too substantial, it becomes debatable whether the study counts as a replication. Initially, the power of confirmation will be high with external replication studies but there will come a point when a result is so well established (or not) that the replication ceases to have research value and the experiment should be moved from the research laboratory into the teaching laboratory.

In terms of method, task, and subjects, we categorize our Korson replication as an example of (improved, similar, similar). The method has been categorized as improved because we debriefed our subjects.

4 Korson replication

4.1 Review of Korson's work

Korson [32, 33] 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). Both programs were written in Turbo Pascal. The changes required could be classified as perfective maintenance as defined by Lientz and Swanson [37] i.e. changes

made to enhance performance, cost effectiveness, efficiency, and maintainability of a program. Korson reckoned that the time taken to make the perfective maintenance changes would be significantly faster for the modular version.

Before 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.

The modification process of the experiment proper 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 4.4.3), 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.

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.

4.2 Critique

On a first reading, Korson's work could be classified as well performed empirical work. A number of criticisms, however, can be made, some of which only arose because we attempted an external replication which necessitated a close inspection of the work reported in [32] and [33].

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.

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 [32] 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 [32, 33] 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 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 realized if notes made by subjects on program listings had been analysed, if subjects had been debriefed on 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 one of the most natural ways 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 external 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.

4.3 Pilot study

4.3.1 Introduction

An initial pilot study was conducted with four members of our research group EFoCS (Empirical Foundations of Computer Science) 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.

4.3.2 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 straightforward 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 hot key customisation code.

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 problems for some subjects when checking their output against expected results.

The solutions adopted for the last three of these problems are discussed in the section on experimental design differences (Section 4.4.1).

4.3.3 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 significant 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 cannot 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 happening - 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 4.2 and some can be clearly viewed as providing additional criticisms of Korson's study. In the discussion section later (Section 5.3), we find similarity between some of the comments above and some of those made by subjects of the experiment proper.

4.4 The external replication study

4.4.1 Experimental design differences

A discussion of the general problem of deciding if any changes should be made to the experimental recipe is presented later in Section 6, along with a description of those suggestions for improvement which we chose not to implement.

Described below are several changes that were made to the experimental instructions in an attempt to improve their readability and highlight particular points. A number of actions that we performed but which were not documented by Korson are also described here. These changes and actions were minor recipe-improving ones whose only impact should be to reduce sources of variability and help understand the processes as well as the products of subjects' behaviour: as such we felt justified in making them.

As noted earlier, a problem was encountered with American-specific terminology e.g. lay-away. 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.² (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 Alt keys provided three distinct ways of carrying out a program test.

The participants were given a listing of the input data file and a listing of the modified output 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 output listings were circled 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 per monitor no comments can be made about differences. In our study, 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

²It is unclear from our facsimile copy of Korson's thesis whether he had also done this.

reference conflicts were caused by this. Also, Korson allowed his subjects to keep the Turbo Pascal reference manuals as payment for their participation. We, on the other hand, gave our undergraduate subjects credit towards the costs of 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. Between the pretest and the experiment we provided our subjects with a break and light refreshments.

Korson did not debrief subjects at the end of the experiment. We chose to do so by asking subjects to complete a debriefing questionnaire which included asking about brief personal details (see Appendix B). This was done to help obtain an understanding of the processes of subjects' behaviour.

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. (Korson made no reference to seat layout.) Subjects were not told about the nature of the experiment, but were verbally instructed that different versions of the program existed and that, subsequently, certain subjects were likely to finish quicker than others: the intention was to reduce subjects concern about their performance. (Again, Korson made no reference to any verbal instructions given to subjects.)

4.4.2 Replication methodology

The replicated experiment required at least 16 subjects, the number used by Korson. We had 23 volunteers. 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 described as a heterogeneous group in terms of age, qualifications, experience and perhaps even ability, all subjects met the criteria set by Korson: "fluency in Pascal, knowledge of the IBM-PC, and an amount of programming experience" [32].

To our surprise, especially so given the comment of one of the pilot subjects who felt that the experimental task required no significant intellectual capacity, six subjects, three from the modular group and three from the monolithic group, failed to complete the experiment. One of these six failed to start the experiment proper, having walked out early. The other five progressed as far as the logic phase.

All of Korson's subjects in his Experiment 1 finished in the first sitting and so we had not planned a second sitting for those who unexpectedly failed to complete. So subjects who did

	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.

not complete were not asked to return later to complete the experiment. Fortunately we had lost equal numbers of subjects from both groups. (We viewed these incompletions as being symptomatic of a floor effect which occurs when an experimental task is too difficult.)

The procedure each subject followed was:

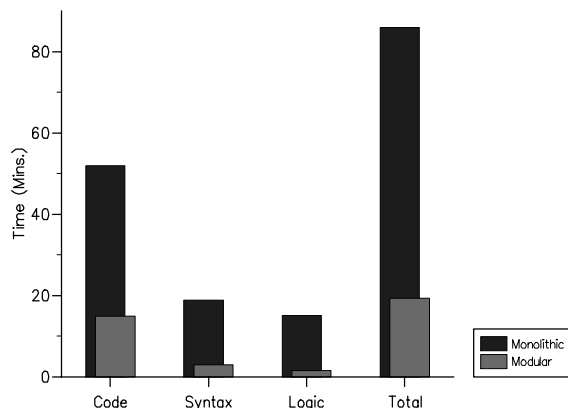
1. pretest to familiarize subjects with the environment; for example experimental procedure, editor, etc.
2. short break (10 - 15 minutes) for refreshments and to allow subjects to clear their thoughts,
3. experiment,
4. finally, as noted above, each subject was asked to complete a debriefing questionnaire which included asking about brief personal details (see Appendix B).

Both pretest and experiment were of the same format that Korson used and the reader is directed to his thesis [32] for full details of the instructions, documentation, modification specifications, and full source code listings.

4.4.3 Statistical results

The timing data collected during the experiment is presented in Figure 1 (the results in parentheses are Korson’s mean times) 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 believed 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. So we adopted

Korson Results – Information Hiding



EFoCS Results – Information Hiding

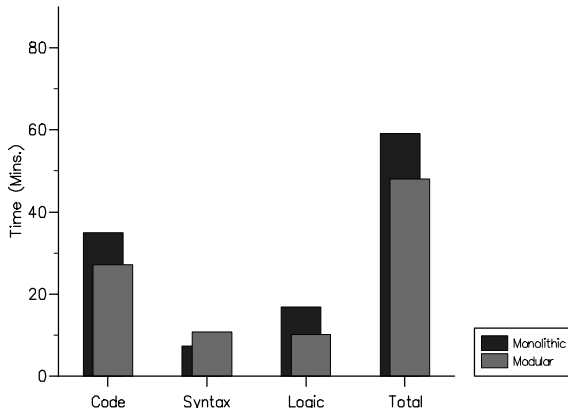


Figure 2: Bar charts displaying mean times.

his null hypothesis that the form of information hiding employed has no effect on maintainability. Surprisingly, however, we did not reject the null hypothesis: mean syntax time actually took longer for the modular version. Also, 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 about 4 times as long as his modular subjects yet our monolithic subjects took about 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 [32, 33] 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 for both our groups were similar and both distributions passed skewness and kurtosis tests of normality: so our use of the t-test was justified (see below).

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 shown in 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 5.3).

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.³ 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 well removed from Korson’s significance level and even the minimal level of 0.05 usually required to reject the null hypothesis. So what is the source of the inconsistency? Should another experiment be performed? To try to break the paradox of the experimenters’ regress, an inductive analysis was undertaken to investigate the reasons for this difference, the results of which are discussed in Section 5.2.

When the null hypothesis is not rejected it is usually necessary to perform a statistical power calculation to determine whether insignificant results were due to poor power levels or whether the phenomenon is really insignificant [4]. Such an analysis was unnecessary here because our inductive analysis yielded a number of interpretations of subjects’ behaviour which render meaningless any interpretation based solely on the null hypothesis.

Version	EFoCS		Korson	
	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

5 Inductive analysis

Traditional statistical approaches require identification of independent, dependent, and random variables with no errors of omission or commission: a single mistake can jeopardise an entire experiment. The assumptions underlying parametric statistics are often never tested (for example, the assumption of normality): a single incorrect assumption can jeopardise an entire experiment. Also, statistical approaches alone often do not give the reader any real insights into the data which can be disadvantageous when trying to understand subject behaviour. Correlation studies between variables of concern might lead to the discovery of significant correlations, but this does not imply cause and effect and individual subject behaviour at different datum points might be the result of entirely different processes.

To overcome these problems, Brooks and Vezza [8] argue for an experimental paradigm where an inductive analysis supplements any statistical analysis. In this paradigm, data gathering remains as unrestricted as possible to allow the building of a database of actual descriptions

³Daly et al[16] incorrectly quote Korson’s ρ value as .0001.

and performance results. Techniques borrowed from machine learning are then applied to this database to generate rules or decision trees. The inductive analysis can help explain the processes underlying individual results as the facts and rules relate to datum points and not averages. An inductive analysis might wrongly be misconstrued as being similar to the shotgun approach to correlation. The differences are that individual datum, not average trends, are being explained and there is no requirement to generalize the results to populations: the emphasis is on trying to understand what took place.

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” [8]. The rule induction system used was IRIS [2] which is based on [48, 23]. IRIS allows each database variable in turn to be the dependent variable. New variables, noted after the experiment, may be introduced to the database, as long as variable values are obtainable.

Usually, induction data has to be described at different levels of detail while observing the effect on the induction rules. Appendix A gives the first level of data before logical groupings took place. In the final phases of the induction process many numeric variables were assigned logical values in an attempt to induce less fragmented rules and some logical groupings were made more crude. Other strategies used to try and induce less fragmented rules included splitting the induction database into two (one for monolithic subjects and one for modular subjects).

We have found that the act of constructing a database brings the experimenter closer to the data and allows a degree of inductive analysis to take place which supplements that achieved by using machine learning to systematically search for patterns. Also, parts of induced rules rather than whole induced rules may suggest patterns worthy of closer examination.

Before discussing the patterns revealed by the inductive analysis (see Section 5.2), the variables used are summarised in Table 2. A more detailed description of variables, where required, is provided below, along with the final logical groupings used.

The analysis extends that given by Daly et al [16] by taking more account of monitors’ observations of the experiment: the impact on the analysis has been of refinement and reinforcement.

5.1 Induction database

Variables (1) to (6), representing the timing data, were each categorized into three logical groups by plotting a histogram of the data and choosing the two best split points by inspection. Subjects were categorized by position, variable (11), 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` (what caused the most

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

difficulty) was graded 1 - pretest ; 2 - finding the correct places to modify; 3 - remaining mixture of individual answers. Variable (15) **code** (understanding the code) was graded 0 - no understanding; 1 - only the relevant parts; 2 - fairly well; 3 - well or very well. Variable (16) **learn** (learned anything) 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** (any other comments) 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. 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. The final database used is shown in Table 3. (A separate induction database was built for the pretest data but no meaningful patterns were suggested.)

subject	Induction Variables																		
	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

5.2 Induced rules and interpretations

Rules were produced for each variable 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 as described below. (The first pattern was discovered indirectly when one rule weakly suggested an ability effect which prompted an investigation of the relationship between pretest timings and timings for the experiment proper.)

1. A suggested relationship was found between the total times taken for the experiment and the pretest. All nine of the monolithic subjects appeared in the top twelve places when ranked by pretest timings (see data in Figures 1 and 3).
2. Of six subjects who had missed a high number of changes in the think phase of the experiment, four had the highest syntax times.

The first pattern can be interpreted as an ability effect. Also, 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

	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.

(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 Korson’s series of experiments). So the suggested ability effect is one source of variability between the two experiments. The ability effect interpretation is the *bête noir* of performance studies with human subjects: performance behaviour and experimental outcomes can wholly be attributed to the ability of each subject, but such an interpretation fails to reveal anything about the treatment effects which must have had some influence.

Only subjects B, Q, and O deviate from this ability interpretation in the sense of a performance mismatch between pretest and experiment. Comments made during debriefing helped explain the mismatches. Subject B, who had the longest pretest 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 pretest instructions, but that he had learned from the pretest. This subject also mentioned that the comment preceding the four procedures where the required modifications were concentrated was a big help. Subject Q, who took the longest with the monolithic code despite being the second fastest on the pretest, commented that he had not read the instructions fully at the start and had missed important code for outputting a record. This subject had also accidentally deleted a line during the experiment and a monitor advised him of his problem. The monitor was later interviewed about this intervention. There was no doubt that the line had been accidentally deleted and it was simply coincidence that this had been observed by the monitor. When the accidental deletion was corrected the program passed the test straightaway. So we felt justified in including this datum in our analysis: to have discarded it would have meant discarding the longest time for a monolithic subject. We considered applying a correction factor to the recorded time but the monitor could only crudely estimate that several minutes had elapsed between the accidental deletion and its notification to the subject — too crude an estimate to work with. Had the subject been allowed to continue unawares, then the recorded time would have had little to do with the experimental hypothesis

as the source of this difficulty was program independent.

Subject O, who had a good pretest performance but a relatively poor performance on the experiment proper, commented on a difficulty having arisen in the experiment because of having read data into the wrong variable and that he hated monolithic code. This subject had also written procedures, an action which could have taken extra time.

The second pattern concerns four of the six subjects who had missed a high number of changes in the think phase of the experiment. These four (subjects F,G,H, and N) had the highest syntax times and on checking the saved versions of their programs it became clear that they all had made edits correcting semantics during the syntax phase, specifically against instructions. (A monitor had made a written note that Subject F appeared to also be using the edit phase for thinking.) Significantly, 3 out of these 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. This clearly indicates a problem with the experiment. The work of one phase appears to be overlapping into the work of another so creating inaccurate timings of think, syntax, and logic phases. (One monitor had also made a written note that Subject P realized he would be making many syntax errors if he simply typed in what he had written down during the think phase and that Subject P had interrupted his edit phase to ask the monitor what he should do. The monitor advised him to do what was required to make the program work.) Possible reasons for this phase problem include: (a) the experimental environment of think, edit, syntax and logic phases was simply too artificial, and (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.

5.3 Further analysis and 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 final working 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, so the remaining subjects should have had relatively low logic times: all they should have been doing was running through the testing procedure. In reality this was not so. 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. One explanation of this unexpected variability is the procedure the subjects followed to test

the program. As in Korson's experiment, to allow subjects to target their testing, 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 timing results of each way were as follows, (i) 1m 15s, (ii) 55s, and (iii) 2m 30s.⁴ 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. 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)? The answer is this would bring these subjects' logic times into line with all Korson's modular subjects (bar one) but that there would be still be no significant difference between modular and monolithic subjects with $\rho < .259$ (two-tailed, independent t-test with $df = 15$, $t = -1.172$). The difference between the total time means widens only a little 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 nominal ρ level of 0.05 usually required. (The time stretch factor of 1.38 matches the size stretch factor between modular and monolithic programs, but this is very probably coincidental given all the sources of variability revealed by the inductive analysis.)

The remaining analysis and discussion of this section draws on many of the individual comments made by subjects, notes made by the monitors, and follow up inspections of the programs written: this discussion can be viewed as an inductive analysis arising from a cross-referencing of material which goes beyond that achieved by using the machine learning techniques.

Another source of variability was the approach the subjects took toward making the modifications. It was observed by monitors during the experiment that several subjects appeared to be attempting to follow the execution flow of the program, rather than just concentrating on finding the modifications to be made. One monitor had made a written note that modular subjects E and F appeared to be doing too much searching around. These differences in approach may be related to subjects' cognitive style: a field dependent style involves less analytical and more global processing of stimulus material. McKenna [42], however, has argued that

... measures of field dependence do not meet the criteria for a cognitive style at the conceptual level and at the empirical level there are substantial correlations with standard ability tests.

As a result of the different approaches taken, subjects' understanding of the code ranged from

⁴As performed by a person competent with both the testing procedure and working program.

none at all to very well. Subjects I and J, the two fastest finishers in the monolithic version, both explained in the subsequent debriefing questionnaire that they had no understanding of the code and both subjects commented 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 4.3). 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. Subject G specifically commented that following the flow of program control had caused the most difficulty. So there is evidence that the four slowest modular subjects (E, F, G, and H) had all tried to understand the code more than was strictly necessary and there is evidence that two of the four fastest modular subjects (B and D) had noticed the comment preceding the four procedures where the required modifications were concentrated and had found it to be of help.

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. One monitor had made a written note that Subject E used an eraser to correct his pencil markings — yet another source of variability in the timing data.

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 phases. One of the subjects who did not complete the actual experiment, however, commented that 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; for example compiling, running and experiencing the program first and then making changes. The phased approach was artificial to him. Subjects K, M, and P made similar comments. Subjects A, B, and Q commented that the instructions were long and difficult to read.

So by unravelling the processes we have revealed several possible reasons for the variability in our subject behaviour and we have broken the paradox of the experimenters' regress: we can explain the different results obtained in our replication study.

6 Guidance

6.1 Introduction

Guidelines for the reporting of computational experiments with mathematical software are well established. See Crowder et al [14], Jackson et al [29], Lee et al [34], and Hoaglin and Andrews [26]. Even so, Hauck and Anderson [24] found little or no improvement in the reporting of simulation experiments by statisticians nearly a decade after the recommendations of Hoaglin and Andrews were published and, more recently, Hooker [27] has commented that

Not even the minimal standards of reproducibility are observed by most authors.

Ironically, bearing in mind the work of Shneiderman et al which was dismissive of the utility of flowcharts, Lee et al [34] suggest use of *flowcharts* as one way of describing the detail of algorithms used in computational experiments with mathematical software.

In general terms, much of the guidance provided in these papers carries over to the reporting of software engineering experiments. Below, we give advice regarding the scale of recipe-improving when conducting an external replication and advice regarding the reporting of subject-based experiments.

6.2 Scale of recipe improving

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 perform a near exact external replication to confirm first the original results i.e. only minor recipe-improving changes could be tolerated. Minor recipe-improving changes that were implemented have already been described (Section 4.4.1). We did, however, make the improvement of debriefing subjects but this did not interfere with the application of the original method. We list here 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.) **Recommendation:** when conducting an external replication, carefully consider the purity of the replication.

6.3 Level of report detail

Once an experiment has been performed, analysed and the time comes to writing the findings, the researcher must provide as much detail surrounding the empirical work as possible to allow others to perform external replications. The volume of such material dictates that it is likely that it can only be documented in a thesis for a higher degree or with the aid of a series of highly detailed technical reports or laboratory kit made available on request. We have recently been made aware of a laboratory kit for software inspections [46] now being used for external replication work. This is the way forward.

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 external replication study and which disguise some possible sources of variability between the experiments. We present these details as a list of desiderata.

First, Korson did not present any information regarding the number of monitors he used: we found a ratio of one monitor to 4 or 5 subjects was barely satisfactory and had we known that Korson had worked with a better monitor-subject ratio we would have taken steps to provide additional monitors. **Recommendation:** explicitly state the monitor-subject ratio.

Second, Korson made no mention of verbal instructions issued to subjects. Was additional information given to them emphasising particular written instructions or was nothing said? We make an issue of this because we gave information verbally to students that different versions of the program existed almost without thinking. **Recommendation:** clarify any verbal instructions given to subjects.

Third, 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. **Recommendation:** for experiments involving subjects progressing through various phases, clarify whether subjects were required to genuinely complete phases and, if so, how this was controlled.

Fourth, Korson gave his subjects three distinct ways of testing the program for correctness, yet did not publish the likely timings for these different methods: providing variability in the method of testing may have been a source of variability in both his and our results. **Recommendation:** expert timings should be reported, especially so if there is an established variation in the way subjects may perform. (The reporting of expert timings also acts as a check on the existence of ceiling effects which occur when an experimental task is too easy.)

Fifth, it is usual to address outliers in a data set, but Korson does not present any additional information to explain why one of his subjects took an exceptionally long time to complete the monolithic version: he relies solely on the interpretation bound up in the null hypothesis. Providing additional information might have benefited researchers interested in performing an external replication. **Recommendation:** alternative explanations of data, especially outlier data, should be sought and reported.

Sixth, Korson did not publish his pretest results. Had he done so, this would have allowed comparison with our subjects' times and provided another measure of the ability of our subjects relative to the original subjects. **Recommendation:** when pretest data is collected, it should be published to assist those who are attempting an external replication.

Seventh, 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 some subjects whose experience or ability was not of the required level may have passed the selection

criteria for our replication study. (Recall that six of our subjects failed to complete.) Whilst Korson qualified his professional programmer classification as someone with the equivalent of at least one year's programming experience as a full time programmer, he did not provide such additional qualification for his advanced computer science students. Any computing student who had completed a Pascal programming class could have passed the criteria set down by Korson: fluency in Pascal, knowledge of the IBM-PC, and an amount of programming experience. This is an example of poor linguistic formulation because of the degree of subjectivity in determining what is meant by the words: fluency, knowledge, and amount. **Recommendation:** more objective subject selection criteria should be stated when specifying parameters such as age, qualifications, position, programming experience, previous experience of performing maintenance tasks, experience of software environments (languages, editors, compilers or interpreters), and experience of platform employed for the experiment. The recency of any experience may also be an important parameter. (The danger is that the criteria become too exacting and gathering enough participants for the study becomes impossible: at the same time, the subject pool can become so specialised that external validity is compromised because of the real variability amongst professional software engineers.)

Finally, criticism can be made of the fact that though seven of Korson's 16 subjects were classified as being professional programmers, the distinction is not maintained in neither the design, the analysis, or the interpretation of results: subjects were randomly assigned. Had all the professional programmers been assigned to the modular group? It is not known. **Recommendation:** where recognizable differences exist between subjects, data is presented to allow an analysis taking account of these differences.

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 to explain the marked difference between our results and those of Korson. Korson went much further than many researchers in reporting experimental procedures and materials and he must be commended for that. He published his code for the experiments (both the pretest and the experimental code) and the instructions for both the pretest and experiment. He published individual subject timings rather than just averages, along with the statistical tests and their results. Unfortunately, a few omitted details have prevented the fullest possible interpretations of the external replication. so any major technical report or thesis presenting empirical work should report even seemingly minor details as noted above to help others to attempt a meaningful external replication.

Other researchers will no doubt wish to add to this list of desiderata as more external replications are performed.

7 Conclusions

The need for external replication of software engineering experiments has been justified and we have presented results from our first external replication which was of a study by Korson into the effects of modularity on program maintainability.

Our results were markedly different to those of Korson: though our findings were in the same direction as Korson's, we did not find the large and highly significant effect that Korson found i.e. we did not replicate his findings.

Korson chose to rely on a quantitative analysis of timings i.e. solely on outputs. In contrast, our approach was to combine traditional methods with an inductive analysis: this enabled an understanding of the behaviour of our subjects (i.e. the processes involved) and yielded a number of interpretations to explain why the results were different. The inductive analysis broke the paradox of the experimenters' regress. In this sense neither ourselves or Korson were wrong though Korson's analysis was weak by not taking account of the processes and we have made a number of criticisms of his experiment. We conclude that it is not yet possible to generalize about the scale of the benefits of modularity: independent verification and validation of a new generation of modularity experiments is desirable.

We found evidence for an ability effect, a 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 accidentally deleting a line caused problems for one subject. Two subjects even began to proceduralise the monolithic code. Some subjects commented on the artificiality of the experiment: one in particular wanted to use a prototyping approach and was unhappy at having to fall-in with the constraints of the experiment. We conclude it preferable to use more scenario-based or naturalistic forms of subject-based experimentation.

Of 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, we had evidence that the four slowest finishers on the modular version had taken some trouble to try and understand the program. The former behaviour we call pragmatic maintenance — focussing in on only those bits of code required and no more. So we also conclude it vital to observe a number of maintenance activities to understand the role of such pragmatic maintenance and to determine if it is a good or a bad thing.

There is no getting away from explaining the human component in software engineering experiments that have human subjects.

We regard this paper as being complementary to Hooker's[27] paper on the need for an empirical science of algorithms and the paper by Basili et al[5] on experimentation in software engineering through our extension to their framework.

We recommend that educational programmes place greater emphasis on laboratory work with students designing, executing, analysing, and reporting experiments, many of which should be external replications of classic work to identify discipline milestones.

We conclude that external replications, combined with inductive analysis techniques, have an important if not vital part to play in the realization of generalizable results. Software engineering experiments must be externally replicated at research centres around the world. (In the human factors literature, Chapanis[11] has made a similar plea for replication as a way of achieving generalizations.)

Acknowledgments

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

References

- [1] I Amato. Pons and Fleischmann Redux? *Science*, 260:895, May 1993.
- [2] G Arisholm. IRIS integrated rule induction system. *Dept. of Computer Science, University of Strathclyde, Scotland*, 1987.
- [3] R D Banker, S M Datar, C F Kemerer, and D Zweig. Software complexity and maintenance costs. *Communications of the ACM*, 36(11):81–94, 1993.
- [4] Jack J. Baroudi and Wanda J. Orlikowski. The problem of statistical power in MIS research. *MIS Quarterly*, 13:87–106, March 1989.
- [5] V R Basili, R W Selby, and D H Hutchens. Experimentation in software engineering. *IEEE Transactions in Software Engineering*, 12(7):733–743, 1986.
- [6] J Brewer and A Hunter. *Multimethod Research A Synthesis of Styles*. Sage Publications, 1989. Sage Library of Social Research 175.
- [7] William Broad and Nicholas Wade. *Betrayers of the Truth*, page 17 and 81. Oxford University Press, 1986.
- [8] A Brooks and P Veza. Inductive analysis applied to the evaluation of a CAI tutorial. *Interacting with Computers, The Interdisciplinary Journal of Human-Computer Interaction 1*, pages 159–170, 1989.
- [9] Rueven E. Brooks. Studying programmer behavior experimentally: The problems of proper methodology. *Communications of the ACM*, 23(4):207–213, April 1980.
- [10] D N Card. Software quality engineering. *Information and Software Technology*, 32(1):3–10, 1990.
- [11] A Chapanis. Some Generalisations about Generalisation. *Human Factors*, 30(3):253–267, 1988.

- [12] F Close. *TOO HOT TO HANDLE The Story of the Race for Cold Fusion*. W H Allen Publishing, 1990.
- [13] H M Collins. *CHANGING ORDER Replication and Induction in Scientific Practice*, pages 19,35,43. SAGE Publications, 1985.
- [14] H Crowder, R S Dembo, and J M Mulvey. On reporting computational experiments with mathematical software. *ACM Transactions on Mathematical Software*, 5(2):193–203, 1979.
- [15] B Curtis. Measurement and experimentation in software engineering. *Proceedings of the IEEE*, 68(9):1144–1157, 1980.
- [16] J Daly, A Brooks, J Miller, M Roper, and M Wood. Verification of results in software maintenance through external replication. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 50–57, September 1994.
- [17] N Fenton. Software measurement: A necessary scientific basis. *IEEE Transactions in Software Engineering*, 20(3):199–206, 1994.
- [18] J Friedrich. Primary Error Detection and Minimization (PEDMIN Strategies on Social Cognition: A Reinterpretation of the Confirmation Bias Phenomenon. *Psychological Review*, 100(2):298–319, 1993.
- [19] Excerpts from a report by the Computer Science and Technology Board. Scaling up: A research agenda for software engineering. *Communications of the ACM*, 33(3):281–293, 1990.
- [20] W W Gibbs. Software’s chronic crisis. *Scientific American*, 271(3):72–81, September 1994.
- [21] R L Glass. *SOFTWARE CONFLICT Essays on the Art and Science of Software Engineering*. Yourdon Press, 1991.
- [22] Martin Goldstein and Inge F Goldstein. *HOW WE KNOW An Exploration of the Scientific Process*, page 207. Plenum Press, New York and London, 1978.
- [23] 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.
- [24] W W Hauck and S Anderson. A survey regarding the reporting of simulation studies. *The American Statistician*, 38(3):214–216, 1984.
- [25] Sallie M Henry and Matthew Humphrey. A controlled experiment to evaluate maintainability of object-oriented software. In *Proceedings of the IEEE Conference on Software Maintenance*, pages 258–265, 1990.
- [26] D C Hoaglin and D F Andrews. The reporting of computation-based results in statistics. *The American Statistician*, 29(3):122–126, 1975.
- [27] J N Hooker. Needed: An empirical science of algorithms. *Operations Research*, 42(2):201–212, 1994.
- [28] T H Huxley. We are all scientists. In H Shapley, S Rapport, and H Wright, editors, *THE NEW TREASURY OF SCIENCE*, page 14. Collins, London and Glasgow, 1965.

- [29] R H F Jackson, P T Boggs, S G Nasch, and S Powell. Guidelines for reporting results of computational experiments. report of the ad hoc committee. *Mathematical programming*, 49:413–425, 1991.
- [30] J P J Kelly, T I McVittie, and W I Yamamoto. Implementing design diversity to achieve fault tolerance. *IEEE Software*, 8(4):61–71, 1991.
- [31] B A Kitchenham, S G Linkman, and D T Law. Critical review of quantitative assessment. *Software Engineering Journal*, 9(2):43–53, 1994.
- [32] 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.
- [33] 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.
- [34] C Lee, J Bard, M Pinedo, and W E Wilhelm. Guidelines for reporting computational results in *IIE TRANSACTIONS*. *IIE Transactions*, 25(6):121–123, 1993.
- [35] N G Leveson. High-pressure steam engines and computer software. *Computer*, 27(10):65–73, 1994.
- [36] John Lewis, Sallie Henry, Dennis Kafura, and Robert Schulman. An empirical study of the object-oriented paradigm and software reuse. In *OOPSLA*, pages 184–196, 1991.
- [37] B Lientz and E Swanson. *Software Maintenance Management*. Addison-Wesley, 1st edition, 1980.
- [38] S G MacDonnell. Rigor in software complexity measurement experimentation. *Journal of Systems and Software*, 16:141–149, 1991.
- [39] A L Mackay. *A dictionary of scientific quotations*. A Hilger, 1991.
- [40] J. Marciniak, editor. *Encyclopedia of Software Engineering*, volume 1 and 2. John Wiley and Sons, Inc., 1994.
- [41] J A McDermid, editor. *Software Engineer's Reference Book*. Butterworth-Heinemann Ltd, 1994.
- [42] F P McKenna. Measures of field dependence: Cognitive style or cognitive ability. *Journal of Personality and Social Psychology*, 47(3):593–603, 1984.
- [43] P Naur and B Randell, editors. *Software Engineering, Report of a conference sponsored by the NATO Science Committee*. NATO, 1968. Held at Garmisch, Germany, 7th-11th October 1968.
- [44] Meilir Page-Jones. *Practical Guide to Structured Systems Design*. Prentice-Hall International, second edition, 1988.
- [45] K R Popper. *The Logic of Scientific Discovery*. Hutchinson of London, revised edition, 1968.
- [46] A A Porter, L G Votta, and V R Basili. Comparing detection methods for software requirements inspections: A replicated experiment. Technical report, Department of Computer Science, University of Maryland, 1994.

	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

- [47] C Potts. Software-engineering research revisited. *IEEE Software*, 10(5):19–28, 1993.
- [48] J R Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [49] Marc Roper. Software testing: A selected annotated bibliography. *Software testing, verification and reliability*, 2:113–132, 1992.
- [50] David A. Scanlan. Structured flowcharts outperform pseudocode: An experimental comparison. *IEEE Software*, 6(5):28–36, September 1989.
- [51] S Sharpe, D A Haworth, and Hale D. Characteristics of empirical software maintenance studies: 1980-1989. *Journal of Software Maintenance: Research and Practice*, 3:1–15, 1991.
- [52] B. Shneiderman, R. Mayer, D. McKay, and P. Heller. Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM*, 20(6):373–381, 1977.
- [53] G P Smith. The problems of reduction and replication in the practice of the scientific method. *Annals of the New York Academy of Sciences*, 406:1–4, 1983.

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 in to similar replies wherever possible: this is the only way to represent the data in a form IRIS

may use. (Some questionnaire responses were sufficiently dissimilar that they were categorized as a group of individual answers. Many of these responses are dealt with in Section 5.)

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?