Empirical Software Engineering, 3, 233–253 (1998)

A Comparison of Tool-Based and Paper-Based Software Inspection

F. MACDONALD
Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, United Kingdom

J. MILLER
Department of Computer Science, University of Strathclyde, Glasgow G1 1XH, United Kingdom

Abstract. Software inspection is an effective method of defect detection. Recent research activity has considered the development of tool support to further increase the efficiency and effectiveness of inspection, resulting in a number of prototype tools being developed. However, no comprehensive evaluations of these tools have been carried out to determine their effectiveness in comparison with traditional paper-based inspection. This issue must be addressed if tool-supported inspection is to become an accepted alternative to, or even replace, paper-based inspection.

This paper describes a controlled experiment comparing the effectiveness of tool-supported software inspection with paper-based inspection, using a new prototype software inspection tool known as ASSIST (Asynchronous/Synchronous Software Inspection Support Tool). 43 students used ASSIST and paper-based inspection to inspect two C++ programs of approximately 150 lines. The subjects performed both individual inspection and a group collection meeting, representing a typical inspection process. It was found that subjects performed equally well with tool-based inspection as with paper-based, measured in terms of the number of defects found, the number of false positives reported, and meeting gains and losses.

Keywords: Software inspection, CASE, tool support, controlled experiment

1. Introduction

Software inspection, originally described by Michael Fagan over twenty years ago (Fagan, 1976), is well-known as an effective defect finding technique. Experience reports extolling the virtues of inspection are easily found. For example, Gilb and Graham (1993) present a number of success stories from a variety of projects. Despite the benefits, inspection remains an expensive process, mainly due to the number of people involved (usually at least three), and the amount of time which the team must spend both individually and as a group. Furthermore, inspection is most effective when the process is applied rigorously. If rigour is lacking, feedback from the process cannot be used for improvement. Yet rigour can be difficult to achieve due to incomplete or insufficiently detailed descriptions of the process (Russell, 1991), such that actual practice varies depending on the interpretation. The issue is compounded by the variety of processes which have been proposed.

Computer support has been suggested as a means to reduce costs and improve rigour, resulting in the development of a number of prototype tools, from both academic and industrial sources. A comprehensive review of these can be found in (Macdonald et al., 1995) and (Macdonald et al., 1996). These tools allow the inspection team to browse and annotate the product on-line, and may support discussion during meetings. Although existing systems present innovative approaches to supporting inspection, in general they suffer from a num-

ber of shortcomings. Primarily, they support only a single, usually proprietary, inspection process. They also only support inspection of plain text documents, while today's software development environments contain many different document types. The move to an electronic medium also provides an opportunity to explore new defect detection techniques, yet such work remains sparse. Finally, although collection and analysis of inspection data is desirable, existing tools provide little such functionality.

Given the limitations of existing tool support, the authors have been working to design and implement a second generation inspection support tool which embodies the important lessons learned from first generation tools, as well as tackling perceived weaknesses. This system is known as ASSIST (Asynchronous/Synchronous Software Inspection Support Tool). ASSIST has also been developed with the goal of comparing tool-based inspection with paper-based. In the next section, the evaluations of existing tools are described. Some issues pertaining to tool support for software inspection are then discussed, followed by an overview of ASSIST.

1.1. Evaluations of Existing Support Tools

While there have been a number of attempts at implementing tool support for software inspection, the quality of evaluation of each tool varies enormously. For example, in the case of ICICLE (Brothers et al., 1992), the only published evaluation comes in the form of 'lessons learned'. In the case of Scrutiny, in addition to lessons learned (Gintell et al.,1995), the authors also claim that tool-based inspection is as effective as paper-based, but there is no quantifiable evidence to support this claim (Gintell et al., 1993).

Knight and Myers (1993) describe two experiments involving their InspeQ inspection tool, designed to support their phased inspection method. The first simply provided information on the feasibility of their method, and on the usability of the associated toolset. The second experiment involved inspection of C code, but provided no comparison with paper-based inspection. Mashayekhi (1995) reports on the implementation of three prototype tools, with the aim of investigating distribution and asynchrony in software engineering. Again, no comparisons with paper-based inspection are made, except in the case of group meetings, where comparable meeting losses are found using both the tool and paper-based methods.

Finally, CSRS (Collaborative Software Review System) has been used to compare the cost effectiveness of group-based review with that of individual-based review (Tjahjono, 1995). Again, since both methods are tool-based, there is no indication of the relative merits of tool-based and paper-based inspection.

The above represent the best evaluations which can be found in the literature. A number of other tools have also been reported, all with very little evaluation. And although the evaluations described above attempt to measure, in various ways, the effectiveness of tool support, the fundamental question "Is tool-based software inspection as effective as paper-based inspection?" remains unanswered. Tool-supported inspection will only become an accepted practice if it can be demonstrated that it does not detract from the main goal of software inspections: finding defects.

1.2. Issues in Tool Support for Software Inspection

When moving from paper-based to even the simplest tool-based inspection, there are a number of advantages which can be identified. To begin with, all documents used in the inspection can be presented in electronic form, a natural consequence of electronic document preparation. The tool can therefore automatically ensure that the most up-to-date version of the document is used, especially if integrated with a version control system. It is also possible to provide cross-referencing facilities for documents, from simple search facilities to sophisticated code browsers. Use of electronic documents also avoids the cost of printing multiple copies of documents for each inspector, along with the associated environmental factors.

The crux of inspection is finding defects in the product. A fundamental feature of any inspection support tool, therefore, is flexible storage and manipulation of these defect lists. Inspection support tools allow defects to be linked to the position in which they occur in the document, allowing them to be accessed easily, obviating the need to add line numbers to documents and minimising errors due to inaccurate noting of the position of the defect. Problems of unintelligible handwriting are also removed. When a synchronous group meeting is held, the burden on the scribe is reduced as defects can be easily shared between participants and quickly copied into the list which forms the output of the meeting, precluding any misunderstanding when transcribing the defect. Finally, a voting mechanism can be used to help speed the discussion process.

On the other hand, the move to a computer-based process does create several drawbacks. The time required to train an inspector in the use of the tool may become an issue. If the tool is too complex, it may detract from the effectiveness of the inspection, even in experienced inspectors. Also, many people are far slower at typing than handwriting, slowing down the process of noting down defects, especially if the tool is heavily mouse-based.

There may also be two problems concerning the move from paper to screen. One concern is the limited amount of screen space. Examination of the product, a source document and a checklist requires three windows to be on-screen simultaneously. However, most common displays are not capable of simultaneously showing three such windows of sufficient size to be useful. The screen may also be cluttered with other windows necessary for operation of the tool, such as has been described in Scrutiny (Gintell et al., 1995). Contrast this with paper-based inspection, where inspectors are free to find as large a workspace as required and to spread all the documents around in a manner comfortable to their working method. The second problem concerns reading text from a screen. There have been a number of studies comparing reading from screen versus reading from paper, and Dillon (1992) provides a good review of these. Evidence points to a 20–30% reduction in speed when reading from screen compared to reading from paper, while reading accuracy may suffer for visually- or cognitively-demanding tasks. On the other hand, comprehension appears not to be affected, and may even be improved. Ergonomic issues may also have a part to play in reading text from a screen. These include the fixed orientation of the screen, differing width to height ratios, refresh rates, image polarity, display quality and so on. Reading from screen has traditionally been thought to be more tiring than reading from paper, while there appears to be a natural tendency for users to prefer paper. Several caveats apply when considering

how these types of study apply to tool-based inspection, including their length, the type of text used, and the task being performed. Lastly, it should be noted that the quality of an individual display will affect the usability of such a tool, and that the ideal presentation will vary depending on the individual user, implying that the ability to customise the tool is an advantage.

This summarises the advantages and disadvantages of applying the simplest tool support to inspection. While more advanced tools may provide other benefits, such as allowing distributed inspection, the main purpose of inspection is always finding defects. If it can be proven that the simplest level of support does not alter the efficiency of inspection, other, more advanced, support can be explored secure in the knowledge that the overall concept is not fundamentally flawed. In this case, an appropriate measure of efficiency is the number of defects detected in a constant period of time. More formally, the null hypothesis, H_0, can be stated as:

> There is no significant difference in performance between individuals performing tool-based inspection and those performing paper-based inspection, measured by the total number of defects found during a given time period.

The alternative hypothesis, H_1 is simply:

> There is a significant difference in performance between individuals performing tool-based inspection and those performing paper-based inspection, measured by the total number of defects found during a given time period.

Similar hypotheses can also be formed when considering the performance of inspection teams as a whole.

Practically all existing support tools provide the level of support required to test these hypotheses. Given that favourable comparison with paper-based inspection is the means by which tool-supported inspection will become acceptable, it is surprising that there is little investigation of this nature. To test these hypotheses, and to tackle perceived deficiencies in existing tools, a new prototype inspection support tool has been developed, and is described in the next section.

1.3. An Overview of ASSIST

Our initial research concentrated on tackling the lack of flexibility of existing tools in terms of the number of inspection processes which they support. This research has produced an inspection process modelling language known as IPDL (Inspection Process Definition Language) (Macdonald, 1997), which is capable of describing any current and future inspection processes, along with the documents and personnel involved in those processes. ASSIST is capable of executing any process written in IPDL, ensuring that the process is followed precisely and that the inspection participants are provided with the correct materials and tools at each stage of the inspection. ASSIST is unique in providing such flexible support, which allows it to perform inspections with any number of people on any number and type of documents.

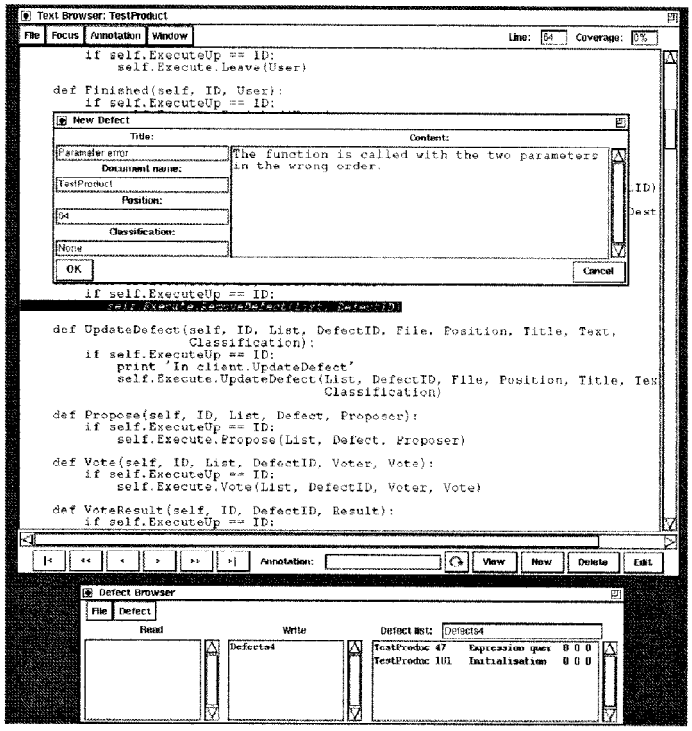


Figure 1. Using ASSIST to inspect some C++ code.

The version of ASSIST used provided four main facilities when performing inspections: the **execute window** and three types of browser. The execute window provides each participant with a list of the people involved in the inspection and their status, along with a list of the documents available. Double-clicking on a document name opens the appropriate browser for that document.

The **list browser** allows the user to manipulate lists of items, typically document annotations describing defects. Lists can be either read only or read-write. Each item within a list consists of a title, the name of the document which the item refers to, a position within that document, a classification and a free-form textual description. Items can be added, removed, edited and copied between lists. Additionally, during a group meeting the list browser allows participants to propose items from their personal lists to the whole group, allowing them to be discussed and voted on. If the item is accepted, it is copied to a master list of defects, representing the output of the group meeting. The scribe may also edit proposed items to reflect refinements suggested at the meeting.

The **text browser** allows documents to be viewed and annotated via the list browser. The browser is based on the concept of a current focus, i.e. a line of text which is currently under scrutiny. The current focus can be annotated, or existing annotations read. During a

group meeting, the focus for the whole group is controlled by the reader: when the reader moves the focus to a new line, the other browsers are automatically updated. The browser indicates the current line number and the percentage of the document inspected. The view of the document can be split horizontally, allowing two separate areas of the document to be viewed simultaneously. Finally, a find facility is available, allowing the user to search for specific words or phrases in the document. This facility works in a manner similar to that found in Netscape. Figure 1 shows the list browser and text browser of ASSIST being used to inspect some C++ code.

Finally, the **simple browser** only allows text documents to be viewed. Facilities such as annotation and line numbering are not available, although the split window and find facilities are still available. This browser is used for all supporting documents, such as checklists and specifications, which are themselves not being inspected.

Apart from the process modelling facilities, this implementation of ASSIST provides the basic level of functionality required to compare tool-based inspection with paper-based, a level which is similar to that provided by the tools described in Section 1.1. It was therefore decided to investigate this question before further implementation was undertaken. Not only would this give information on the comparative effectiveness, it would provide useful feedback on the usability of ASSIST and help shape our future research, in terms of advanced facilities to support defect detection. The next section describes a controlled experiment to test the hypotheses stated in Section 1.2.

2. Experiment Design

The testing of our hypotheses required two groups of subjects to inspect a single document, one using a tool-based approach and the other using a paper-based approach. To ensure that any effect was not simply due to one group of subjects being of higher ability, the subjects must also inspect a second document, this time using the alternative approach. The inspection process used consisted of two stages: an individual detection phase, where each subject inspected the document for faults, and a group collection meeting, where individual lists were consolidated into a single master list for the group. This experiment was done for code inspections and the results are applicable to this type of inspection. The term “document” used in the description of what material was inspected as part of this experiment refers to a “piece of code” as explained in Section 2.3.

2.1. Subjects

The experiment was carried out during late 1996 as part of a team-based third year undergraduate Software Engineering course run by the Computer Science department of Strathclyde University. The students already had a firm grounding in many aspects of Computer Science. In particular, they had been taught programming in Scheme, C++ and Eiffel, and had also completed a course in the fundamentals of Software Engineering. Student motivation was high, since the practical aspects of the experiment formed part of their continual assessment for this course, the final mark of which contributes to their overall degree class.

A total of 43 subjects participated in the class, split into two approximately equal sections. Section 1 had 22 subjects and Section 2 had 21 subjects. The split was achieved by ordering subjects according to their mark in a C++ programming class (C++ was chosen as the type of code to be used in the experiment). Adjacent subjects were then blocked into sets of four, with two randomly chosen subjects assigned to one section, with the remaining two subjects assigned to the other. Within the two sections the subjects were organised into groups of three (and a single group of four). This was done in such a way as to create equal ability groups, based on their C++ programming marks. Section 1 therefore consisted of six groups of three subjects and one group of four subjects, while Section 2 contained seven groups of three subjects.

2.2. Statistical Power

The existence of the phenomenon being empirically investigated does not guarantee production of a statistically significant result. Statistical power analysis is a method of increasing the probability that an effect is found in the empirical study. A high power level indicates that a statistical test has a high probability of producing a statistically significant result, i.e. if an effect exists it is highly likely that it will be found, and a Type II error is unlikely to be committed. Similarly, if an effect does not exist, the researcher has a solid statistical argument for accepting the null hypothesis, which is not the case if the study has low power.

The power of the statistical test becomes a particularly important factor when H_0 is not rejected, i.e. the effect being tested for is not found. The lower the power of the test, the less likely H_0 is accepted correctly. Consequently, when H_0 is not rejected and the statistical test is of low power, the results are ambiguous and the only conclusion that can be drawn is that the effect examined has not been demonstrated by the study. Studies with a high power, on the other hand, offer the advantage of an interpretation of the results when there is insignificance. There exists strong support for the decision not to reject the null hypothesis, something a low powered, statistically insignificant study cannot give.

In common with many software engineering experiments, this experiment has had limited control over the components of statistical power. The hypothesis requires a two-tailed decision, and the experimental plan was to use parametric testing (ANOVA). The sample size (or more correctly the harmonic mean) is defined by the size of the class, and is 21.5. The exact value of required statistical power is a topic of debate amongst statisticians, but most experts agree that a power rating of between 0.7 and 0.8 is required for a correct experimental design. If we desire a statistical power in this range, this implies that the experiment will have to assume a large effect size (approximately 0.8), and hence this defines the sensitivity of the experiment, and the minimal sensitivity where it is 'statistically safe' to accept the null hypothesis.

During the training period, all the exercises where closely monitored, allowing an estimate of the statistical variance of both populations. Combining the two variances produces a result of 0.16, allowing us to calculate the minimum normalised difference (average number of defects found divided by the maximum number of defects available to be found) for which the experimental design can be considered statistically safe—0.13. This equates to 1.5 defects, given that both experimental programs have 12 defects. Therefore, the experimental

design safely allows us to accept the null hypothesis for normalised differences above (approximately) 13%. Obviously the null hypothesis could still be accepted for differences less than 13%, but for these effect sizes the experiment has an increased risk of committing a Type II error. Indeed, by best experimental practice this risk is deemed unacceptable in this situation. Hence, at these reduced effect sizes the experiment is unable to reliably prove that no effect exists. See (Miller et al., 1997) for a fuller discussion of the role of statistical power in experimental design.

2.3. Materials

Having previous experience of running defect detection experiments (Miller et al., 1998), it was decided that the most appropriate material to inspect would be C++ code. A number of factors influenced this decision. Initially, source code was chosen as the appropriate material due to ease with which defects in code can be defined. This is in contrast with, say, English language specifications, which provide many problems of ambiguity. It is also easy for inspection of such material to degenerate into arguments over English style and usage. Intelligent seeding of defects in code avoids these problems and provides a well-defined target against which performance can be judged. Subject experience was also taken into account. Inspection must be performed by personnel with experience in the type of document being inspected. It was therefore important to choose material in a form which the subjects had experience in. This also avoids teaching a new notation or language which subjects may spend much of their time trying to understand and become familiar with, instead of finding defects. Since the subjects were competent in C++, material in that language was chosen for the experiment. The decision was further ratified by the availability of high quality material from a local replication of Kamsties and Lott's defect detection experiment (Kamsties and Lott, 1995).

For the training materials, a selection of programs originally used in Kamsties and Lott's experiment were used, since each program had an appropriate specification, a list of library functions used by the program and a comprehensive fault list. These programs were originally written in non-ANSI C and were translated into standard C++ for our experiment. The programs were also edited to remove some faults and add others. The programs used were: `count.cc` (58 lines, 8 faults), `tokens.cc` (128 lines, 11 faults) and `series.cc` (146 lines, 15 faults). A further example, `simple_sort.cc` (41 lines, 4 faults) was created for use in the tool tutorial.

Since the Kamsties and Lott material had already been used in the same class last year, the two programs to be used for the experiment (and, hence, the assessment), were specifically written afresh. One program (`analyse.cc`, 147 lines, 12 faults) was based on the idea of a simple statistical analysis program given in (Deitel and Deitel, 1994). The second program (`graph.cc`, 143 lines, 12 faults) was written from a specification for a Fortran graph plotting program, originally found in (Basili and Selby, 1987). For each program, a specification written in a similar style to that of the Kamsties and Lott material was also prepared, along with appropriate lists of library functions.

There is no clear consensus on the optimal inspection rate. For example, both Barnard and Price (1994) and Russell (1991) quote a recommendation of 150 lines of code per hour.

On the other hand, Gilb and Graham (1993), recommend inspecting between 0.5 and 1.5 pages per hour, translating to between 30 and 90 lines of code. All conclude that lower rates improve defect detection. Each practical session lasted two hours, giving an inspection rate of around 70 lines per hour. This figure represents a compromise since subjects were not professional inspectors and could not be expected to perform at the highest recommended rates. At the same time, there was enough time pressure to make the task realistic. Two hours is also a standard inspection meeting length.

The actual inspection task was to use the program specification and list of library functions to inspect the source code for functionality defects, making use of a checklist. Use of a checklist is standard inspection procedure, and subjects were supplied with the checklist described below. Students were specifically discouraged from finding defects relating to other qualities, such as efficiency. Each program was seeded with errors of functionality and with checklist violations. For the two experimental programs, defects in one program were matched, in terms of type and perceived difficulty, with defects in the other program, in an effort to match the overall difficulty of the programs. All programs used compiled with no errors or warnings using `CC` under SunOS 4.1.3.

The checklist used was derived from a C code checklist by Marick (1992), a C++ checklist by Baldwin (1992) (derived from the aforementioned C checklist), the C++ checklist from (Humphrey, 1995) and a generic code checklist from (Ebenau and Strauss, 1994). From the C and C++ checklists we removed items which we considered to be irrelevant (for example, none of our programs made extensive use of macros), along with esoteric items, such as those dealing with threaded programming and signals. From the generic checklist we removed items not relevant to C++. Duplicates were then removed and the remaining items grouped into a number of categories. An additional category was added concerning differences between the specification and behaviour of the program. Finally, we performed another edit on the checklist to reduce the number of categories to ten, allowing the checklist to fit on two sides of paper. We felt that a short checklist covering the major points to consider would be more effective than a much longer, more detailed checklist which the subjects would struggle to cover in the time allowed. This follows practice recommended by Gilb and Graham (1993).

2.4. Instrumentation

For paper-based inspection, each student was given an individual defect report form containing blank entries to be filled with each defect found. For group meetings, the scribe was given a similar form to prepare the master list. During tool-based inspection, ASSIST was used to keep both individual lists and the master list. Each practical session was limited to a maximum of 2 hours. Almost all participants made use of the full two hours.

For each subject, data collected were the total number of correct defects found and the number of false positives submitted (i.e. defects which subjects incorrectly identify), and similarly for each group. Also calculated were meeting loss (number of defects found by at least one individual in the group, but not reported by the group as a whole), and meeting gain (number of defects reported by the group, but not reported by any individual) for each

group. Finally, for each defect in each program, the frequency of occurrence was obtained, both in tool-based and paper-based inspection.

2.5. Experiment Execution

The practical element of the course ran over a period of ten weeks. The first six weeks were devoted to providing the subjects with training in software inspection and using ASSIST, as well as refreshing their C++ knowledge. These practical sessions were interspersed with lectures introducing each new topic where appropriate. After inspection of each program was complete, the subjects were presented with a list of faults in that program. The remaining four weeks were used to run the actual experiment. Each practical session was run twice, once for each section of the class, thus ensuring their separation when using different methods on the same program. Both practicals occurred consecutively on the same afternoon of each week.

2.6. Threats to Validity

Threats to Internal Validity

Any empirical study may be distorted by influences which affect dependent variables without the researcher's knowledge and this possibility should be minimised. The following such threats were considered:

- Selection effects may occur due to variations in the natural performance of individual subjects. In our experiment, this was minimised by creating equal ability groups.
- Maturation (learning) effects concern improvement in the performance of subjects during the experiment. Our data was analysed for this and no effect was found. Section 3 describes this analysis in more detail.
- Instrumentation effects may occur due to differences in the experimental materials used. To help counteract the main source of this effect in our experiment, both groups of subjects inspect both programs.
- Presentation effects may occur since both sets of subjects inspect the programs in the same order. It is believed that if such an effect exists, it is symmetric between both sets of subjects, and that the effect presents less risk than the plagiarism effect possible when the order of presentation is reversed for one set of subjects.
- Plagiarism was a concern in our experiment since the group phase of each experimental run took place one week after the individual session, hence providing an opportunity for undesired collaboration among subjects. This was mitigated by retaining all paper materials between phases. With the same purpose in mind, data from the tool regarding the individual phases was extracted immediately after each session. Furthermore, access to the tool and any on-line material was also denied. Finally, any plagiarism effect would

be noticeable by any group presenting an above average meeting gain. No such groups were detected.

Threats to External Validity

Threats to external validity can limit the ability to generalise the results of the experiment to a wider population, in this case actual software engineering practice. The following were considered:

- The student subjects involved in the experiment may not be representative of software engineering professionals. This was unavoidable since our choice of subjects was limited by available resources.
- The programs used may not be representative of the length and complexity of those found in an industrial setting. The programs used were chosen for their length, allowing them to be inspected within the time available. However, the amount of time given to inspect each program was representative of industrial practice quoted in popular inspection literature.
- The inspection process used may not correspond to that used in industry, in terms of process steps and number of participants. For example, the process used did not involve the author presenting an overview of the product, and a rework phase was not used. However, the detection/collection approach used in our experiment is a standard process (Gilb and Graham, 1993).

These threats are typical of many empirical studies, e.g. (Porter et al., 1995; Kamsties and Lott, 1995). They can be reduced by internal and external replication of the experiment, with other subjects, programs and processes.

3. Results and Analysis

3.1. Defect Detection

Table 1 presents a summary of the data and the analysis of variance of the individual phases of both inspections. For each program, the method used by each section of subjects is shown, along with the number of subjects in that section and the mean number of defects found in the program. The standard deviations, standard errors and F ratios and probabilities are also shown. For `analyse.cc`, it is obvious that there is very little difference in performance, and this is confirmed by the analysis of variance. For `graph.cc`, the section using paper-based inspection appear to outperform that using the tool, although this difference is not significant. In both cases the null hypothesis concerning individuals must be accepted.

Table 2 presents a summary of the data and the analysis of variance of the group phases of both inspections. These results follow the same pattern as for individual: `analyse.cc`

Table 1. Analysis of variance of individual defect scores.

Program	`analyse.cc`		`graph.cc`	
Section	1	2	1	2
Method	Tool	Paper	Paper	Tool
Subjects	22	21	22	21
Mean	7.68	7.76	6.64	6.00
St. Dev.	1.55	1.92	1.43	2.05
St. Error	0.33	0.42	0.30	0.45
F Ratio	0.02		1.40	
F Prob.	0.88		0.24	

Table 2. Analysis of variance of group defect scores.

Program	`analyse.cc`		`graph.cc`	
Section	1	2	1	2
Method	Tool	Paper	Paper	Tool
Groups	7	7	7	7
Mean	10.86	10.71	9.57	8.86
St. Dev.	0.69	0.95	1.27	1.07
St. Error	0.26	0.36	0.48	0.40
F Ratio	0.10		1.29	
F Prob.	0.75		0.28	

provides very similar results between methods, while `graph.cc` provides a larger difference, but which is not statistically significant. Again, the null hypothesis as applied to groups must be accepted.

Under further investigation, the data from the individual phase of the `graph.cc` inspection failed the Levene test for homogeneity of variances. However, the robustness of the F test is well documented. For example, Boneau (1960) has studied the effects of violating the assumptions which underlie the t test, and generalised these results to the F test. Provided samples sizes are sufficient (around 15) and virtually equal, and a two-tailed test is used, non-homogeneity of variances should not cause difficulties. A similar conclusion is presented in (Edwards, 1967). As a safeguard, the Kruskall-Wallis non-parametric test was applied to all four sets of data, and gave results similar to those for the parametric tests, with no significance.

The data was analysed for any effect stemming from the order in which the methods were used and for any difference caused by the two programs. The results are shown in Table 3. There proved to be no significant difference between subjects who used the tool first and those who used paper first. However, the results indicated a significant difference in the difficulty of `analyse.cc` compared with `graph.cc`. This was also supported by one of the post-experiment questionnaires (described in more detail in Section 3.4). The greater difference in performance with `graph.cc` may imply that the tool becomes less efficient as the material under inspection becomes more complex, and this should be investigated

Table 3. Analysis of variance of method order and program.

Effect	F Ratio	F Prob.
Order	0.34	0.56
Program	33.78	$\ll$0.01
Order$\times$Program	2.20	0.15

Table 4. Summary defect detection data for `analyse.cc` (individuals).

Defect No.	Tool	Manual
1	13	17
2	22	21
3	2	3
4	16	12
5	7	7
6	21	20
7	21	17
8	4	8
9	17	18
10	13	11
11	18	18
12	14	11

further. Finally, the data was analysed for any effect from the order in which the methods were used combined with the two different programs. No significant result was found.

Table 4 summarises the frequency with which each defect was found in `analyse.cc` during the individual phase. In most cases there is no great difference between the scores achieved with the tool compared to those using paper. Considering the three defects with the largest differences (1, 7, 8), there is no clear indication why such differences exists. Defect 1 is an array indexing error, defect 7 concerns missing functionality and defect 8 is a failure to initialise an array. In theory, defect 8 should be found by all subjects, since it is explicitly covered by an item in a checklist. The difference may indicate that it is more difficult to use the online checklist (lack of screen space). On the other hand the low scores point to an an overall lack of checklist use.

Table 5 shows the frequency of detection for the group phase of the `analyse.cc` inspection. The largest difference is for defect 3 with only one tool-based group finding it, compared to three paper-based groups. The reason for this becomes apparent by considering the individual results. Only two tool users found this defect, each belonging to distinct groups. One group reported the defect, while the other did not. In contrast, three paper-based subjects found the defect, each of whom belonged to a different group. All three groups managed to report this defect. Hence, the overlap between group members' defect lists can have a significant effect on the group score. Given a different makeup of groups, this difference could have been reduced to zero.

Table 5. Summary defect detection data for `analyse.cc` (groups).

Defect No.	Tool	Manual
1	7	7
2	7	7
3	1	3
4	7	6
5	6	6
6	7	7
7	7	7
8	6	6
9	7	7
10	7	6
11	7	7
12	7	6

Table 6. Summary defect detection data for `graph.cc` (individuals).

Defect No.	Tool	Manual
1	20	20
2	19	21
3	13	20
4	4	4
5	16	20
6	7	10
7	13	15
8	1	3
9	4	7
10	7	7
11	19	15
12	3	4

Table 6 summarises the defect detection frequency for the individual phase of the inspection of `graph.cc`. The largest difference appears for the third defect, which was found by 20 of the paper-based inspectors, yet only 13 of the tool users. This alone represents 35% of the overall difference between tool and paper. This defect concerns a missing function call, which means the program does not print some output specified by the specification. In fact, this is a very easy defect to detect using the search facility of the tool. By entering the function name as the target of the search, the inspector can find calls to that function, almost guaranteeing that it will be found. However, although the mechanics of the find facility were explained, the use of the tool to detect such defects was not explicitly taught to the subjects. A difference of 4 in favour of paper-based inspection occurs for defect 5 (variable names X and Y are transposed), but there is no obvious reason for this difference.

Table 7. Summary defect detection data for `graph.cc` (groups).

Defect No.	Tool	Manual
1	7	7
2	7	7
3	7	7
4	3	2
5	7	7
6	5	6
7	7	7
8	0	4
9	4	6
10	4	3
11	7	6
12	4	5

The same applies to defect 11 (an incorrect calculation), which has a difference of 4 in favour of the tool.

The defect detection data for the group phase of the inspection of `graph.cc` is shown in Table 7. This time defect number 8 has the largest difference between methods. The data for the individual phase already shows that this is the most difficult defect to find. The relatively poor performance of the tool can be explained by the fact that every individual who found the defect belonged to a separate group, giving the paper-based groups a 3 to 1 advantage. Another paper-based group also managed to find the defect at the meeting, giving 4 to 1. Finally, the single tool user who actually found the defect either failed to mention it at the group meeting or was talked out of it, since that group did not report it. A similar trend is apparent with defect 9, with 4 individual tool users mapping into 4 groups, and 7 paper users mapping into 6 groups. Again, these results show that the overlap between individual lists can have a large effect on the group result. Groups with individuals whose defect lists have a greater overlap are disadvantaged, even though individual scores may be very respectable. Different group makeups could have reduced both these differences to 1.

3.2. False Positives

In addition to the number of defects found by each subject, the number of false positives reported was also measured. False positives are items reported by subjects as defects, when in fact no defect exists. It is desirable to investigate whether tool-supported inspection alters the number of false positives reported, since an increase would reduce the effectiveness of the inspection. On the other hand, if use of the tool in some way suppressed false positives, the efficiency of the inspection would be increased, with less time wasted on discussing these issues.

Table 8 presents the analysis of variance for the false positive data from the individual phases of each inspection. While the tool appears to provide an improvement over paper

Table 8. Analysis of variance of individual false positives.

Program	`analyse.cc`		`graph.cc`	
Section	1	2	1	2
Method	Tool	Paper	Paper	Tool
Subjects	22	21	22	21
Mean	3.59	4.19	3.23	3.24
St. Dev.	1.943	1.94	1.9	2.14
St. Error	0.41	0.42	0.4	0.47
F Ratio	1.02		≈0.00	
F Prob.	0.32		0.99	

Table 9. Analysis of variance of group false positives.

Program	`analyse.cc`		`graph.cc`	
Section	1	2	1	2
Method	Tool	Paper	Paper	Tool
Groups	7	7	7	7
Mean	3.57	3.14	2.14	2.43
St. Dev.	1.72	1.46	1.34	2.22
St. Error	0.65	0.55	0.51	0.84
F Ratio	0.25		0.08	
F Prob.	0.62		0.78	

for `analyse.cc`, this difference is not significant. On the other hand, the means for `graph.cc` are almost identical, and the ANOVA test confirms this. The analysis of variance for the false positive data of the group phases is shown in Table 9. For both programs, there is no significant difference between the tool-based and paper-based approaches.

Examination of the false positives revealed no discernible difference in those produced by tool-based inspection as compared with paper-based. The only point of interest was that some of the false positives which occurred were defects which had occurred in training material. Presumably the subjects had memorised these as defect types and almost blindly submitted them as defects without checking if they actually occurred.

3.3. Meeting Gains and Losses

The final set of data to be analysed concerns meeting gains and losses. Synchronous meetings are frequently cited as necessary because it is believed that factors such as group synergy improve the output of the meeting, manifested in defects being found at the meeting which have not been found during the individual phase. On the other hand, defects may be lost when a participant fails to raise a defect found during the individual phase. We hypothesised there would be no difference between tool and paper-based methods for both gains and losses. Table 10 shows the gains and losses for all group meetings in both

Table 10. Meeting gains and losses for both programs. Groups 1–7 perfomed tool-based inspection on `analyse.cc` followed by paper-based inspection on `graph.cc`. Groups 8–14 used the methods in reverse.

Program	`analyse.cc`		`graph.cc`	
Group	Gains	Losses	Gains	Losses
1	0	0	0	0
2	1	0	2	1
3	0	0	0	1
4	0	0	1	1
5	0	0	0	1
6	1	1	0	1
7	0	0	0	0
Total	2	1	3	5
8	0	0	1	0
9	0	0	0	2
10	0	0	0	1
11	0	0	0	2
12	0	0	1	0
13	0	0	0	0
14	0	1	1	0
Total	0	1	3	5

inspections. It is clear even without statistical analysis there is no significant difference between methods.

3.4. Debriefing Questionnaires

During the course, subjects were asked to complete four questionnaires. The first two were given after the full practice using ASSIST, one after the individual inspection (Questionnaire 1) and one after the group meeting (Questionnaire 2). These questionnaires focussed on eliciting qualitative feedback on ASSIST. Two further questionnaires were then presented, one after inspection of the first experimental program was complete (Questionnaire 3), the other after inspection of the second experimental program (Questionnaire 4). These questionnaires concentrated on such topics as the overall difficulty of the task and the relative merits of paper-based and tool-based inspection. This section presents some results from Questionnaire 4.

The question “Overall did you feel you performed better during individual inspection using manual (paper-based) inspection or ASSIST, or were you equally effective with both methods?” was asked. 39% of respondents claimed to have performed better using paper-based inspection, 39% had no preference, while 22% claimed to have performed better with ASSIST. The low preference for ASSIST probably stems from the familiarity of paper, which people are comfortable with. It is possible that extended training with ASSIST would

increase its user acceptance. When the same question was asked with regard to the group meeting, the number of people preferring paper-based inspection dropped to 19.5%, the number with no preference increased to 61% and the number preferring ASSIST dropped to 19.5%. This change is probably due to the perception of the group meeting being an easier task than the individual phase. Collating defect lists is presumably easier than finding the defects in the first place, therefore the method used to perform the collation task is less important. There is no clear correspondence between preferences for individual and preferences for group: some people who expressed a preference for paper-based individual inspection then went on to select ASSIST for the group meeting, others always preferred paper-based or always preferred ASSIST, yet others moved only one category (i.e. from paper-based to no preference or tool-based to no preference).

The qualitative statements indicating preference make interesting reading. People who indicated a preference for paper-based inspection generally liked the tactile nature of paper, allowing them to scribble notes on the code itself. Others simply preferred reading code on paper instead of on-screen. A number of people found it awkward moving between the code, specification and checklist windows of ASSIST. While the scribe's burden is reduced in ASSIST, one student commented that it was *too* easy to propose defects and put them into the master defect list, and therefore the phrasing of the defect was not usually considered as much as when the scribe had to manually write it down.

People who preferred ASSIST pointed out the following advantages. It was easy for the group as a whole to see exactly where individuals' errors were, and it was also considered easier to compile the master defect list, giving more time for the group to search for further defects. Others found it easier to traverse the code, and a number of people preferred the defect creation/editing facilities. The voting method for resolving defects was also considered to be useful.

People who expressed no preference also made interesting points. For example: "[It] was easy to look through code when it was on paper, but ASSIST has its advantages such as searching through the document for keywords...[During group meetings] paper-based inspection provoked more discussion, [but] ASSIST made it easier for [the] reader".

Finally, the question "In terms of complexity, how did `graph.cc` compare with `analyse.cc`?" was asked. The five possible responses were: "much more complex", "slightly more complex", "of similar complexity", "slightly less complex" and "much less complex". 49% of subjects believed `graph.cc` to be "much more complex", while 44% believed it to be "slightly more complex", and only 7% considered it to be "of similar complexity" to `analyse.cc`. These comments support our earlier statistical analysis concerning the difference between programs.

4. Conclusions

Software inspection is an effective defect finding technique and has been widely used. Despite its benefits, it is still deemed to be expensive. Inspection support tools have been proposed as a means to reduce the cost of inspection, yet evaluations of existing tools have been sparse. The question as to whether tool-supported inspection can be as effective as paper-based inspection has yet to be answered with any certainty.

This paper has described a controlled experiment designed to investigate the relative performance of these methods, using a two-stage inspection process. Our null hypotheses stated there would be no significant difference between methods, measured in terms of the number of defects found both by each subject and by each group. The results from our experiment show that a straightforward computer-based approach to inspection does not degrade the effectiveness of the inspection in any way. There is no significant difference in the number of defects found. The number of false positives reported and the amount of meeting gains and losses were also measured, and no significant difference was found. Although the experiment made use of student subjects and inspected short pieces of code, the inspection process used was realistic and the rate of inspection was typical of industry. Statistical power in our experiment was sufficient to allows us to reliably accept the null hypothesis for normalised differences above 13%.

Having compared simple tool-based inspection with paper-based inspection, the authors now intend to investigate active defect detection support. It is believed that advanced computer-based support for inspection can provide significant gains in effectiveness. Such support may be specific to certain document types, such as static analysers for code, or applicable to any document, such as new browsing methods. Our current research lies in implementing and evaluating such support.

Acknowledgments

The authors would like to thank Marc Roper and Murray Wood for their assistance in running the experiment and their comments on this paper. We also thank David Lloyd, Tony Povoas and Ian Gordon for their technical support. Our thanks go to all students who participated in the experiment, who also provided very useful feedback on the usability of ASSIST, as well as finding one or two bugs. Finally, we are grateful to Christopher Lott, Eric Kamsties and Gary Perlman for providing the training material used in the experiment, and for allowing us to reproduce this material.

More information on the ASSIST system can be found at

`http://www.cs.strath.ac.uk/research/EFOCS/assist.html`

A version is freely available for research purposes. For more details, please email the first author at

`fraser@cs.strath.ac.uk`.

A replication package containing electronic versions of all material used in the experiment is also available from the first author.

References

Baldwin, John T. 1992. An abbreviated code inspection checklist. URL: `http://www.ics.hawaii.edu/~johnson/FTR/Bib/Baldwin92.html`

Barnard, J., and Price, A. 1994. Managing code inspection information. *IEEE Software* 11(2): 56–69.

Basili, V. R., and Selby, R. W. 1987. Comparing the effectiveness of software testing strategies. *IEEE Transactions on Software Engineering* 13(12): 1278–1296.

Boneau, C. A. 1960. The effects of violations of assumptions underlying the *t* test. *Psychological Bulletin* 57(1): 49–64.

Brothers, L. R., Sembugamoorthy, V., and Irgon, A. E. 1992. Knowledge-based code inspection with ICICLE. In *Innovative Applications of Artificial Intelligence 4: Proceedings of IAAI-92*.

Deitel, H. M., and Deitel, P.J. 1994. *C: How to Program*. Prentice-Hall International, second edition.

Dillon, A. 1992. Reading from paper versus screens: a critical review of the empirical literature. *Ergonomics* 35(10): 1297–1326.

Doolan, E. P. 1992. Experience with Fagan's inspection method. *Software—Practice and Experience* 22(2): 173–182.

Ebenau, R. G., and Strauss, S. H. 1994 *Software Inspection Process*. McGraw-Hill.

Edwards, A. L. 1967. *Statistical Methods*. Holt, Rinehart, and Winston, Inc., second edition.

Fagan, M. E. 1976. Design and code inspections to reduce errors in program development. *IBM Systems Journal* 15(3): 182–211.

Gilb, T., and Graham D. 1993. *Software Inspection*. Addison-Wesley, Wokingham, England.

Gintell, J. W., Arnold, J., Houde, M., Kruszelnicki, J., McKenney, R., and Memmi, G. 1993. Scrutiny: A collaborative inspection and review system. In *Proceedings of the Fourth European Software Engineering Conference*.

Gintell, J. W., Houde, M., and McKenney, R. 1995. Lessons learned by building and using Scrutiny, a collaborative software inspection system. In *Proceedings of the Seventh International Workshop on Computer Aided Software Engineering*.

Humphrey, W. S. 1995. *A Discipline for Software Engineering*. Addison-Wesley.

Kamsties, E., and Lott, C. M. 1995. An empirical evaluation of three defect-detection techniques. Technical Report ISERN-95-02, International Software Engineering Research Network.

Knight, J. C., and Meyers, E. A. 1993. An improved inspection technique. *Communications of the ACM* 36(11): 51–61.

Macdonald, F., and Miller, J. 1997. A software inspection process definition language and prototype support tool. *Software Testing, Verification and Reliability*, 7(2): 99–128.

Macdonald, F., Miller, J., Brooks, A., Roper, M., and Wood, M. 1995. A review of tool support for software inspection. In *Proceedings of the Seventh International Workshop on Computer Aided Software Engineering*, 340–349.

Macdonald, F., Miller, J., Brooks, A., Roper, M., and Wood, M. 1996. Automating the software inspection process. *Automated Software Engineering: An International Journal*, 3(3/4): 193–218.

Marick, B. 1992. A question catalog for code inspections. Available via anonymous FTP from `cs.uiuc.edu` as `/pub/testing/inspect.ps`

Mashayekhi, V. 1995. *Distribution and Asynchrony in Software Engineering*. Ph.D. thesis, University of Minnesota, March 1995.

Miller, J., Daly, J., Wood, M., Brooks, A., and Roper, M. 1997. Statistical power and its subcomponents—missing and misunderstood concepts in empirical software engineering research. *Information and Software Technology*, 39(4): 285–295.

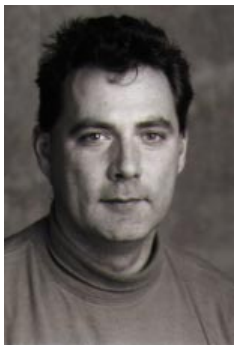
Miller, Roper, M., and Woord M.. 1998. Further experiences with scenarios and checklists. *Journal of Empirical Software Engineering*, 3(1): 37–64.

Porter A. A., Votta, L. G., and Basili, V. R. 1995. Comparing detection methods for software requirements inspections: A replicated experiment. *IEEE Transactions on Software Engineering*, 21(6): 563–575.

Russell, G. W. 1991. Experience with inspections in ultralarge-scale development, *IEEE Software*, 8(1): 25–31.

Tjahjono, D. 1995. Comparing the cost effectiveness of group synchronous review method and individual asynchronous review method using CSRS: Results of pilot study. Technical Report ICS-TR-95-07, University of Hawaii.

Fraser Macdonald received the B.Sc. degree in Computer Science from the University of Strathclyde, Glasgow, Scotland in 1994. He is currently employed there as a teaching assistant while researching for his Ph.D. His research interests include software inspection, object-oriented systems and software visualisation.



James Miller received the B.Sc. and Ph.D. degrees in Computer Science from the University of Strathclyde, Glasgow, Scotland. After a period at the National Engineering Research Initiative Centre researching pattern recognition, he accepted a lectureship at the University of Strathclyde. His current research interests include empirical evaluation, software inspection, object-oriented systems, quality function deployment and the application of TQM techniques to software engineering. His earlier research interests included computer vision, automation, and industrial inspection.