38

Experience in Software Inspection Techniques

David A Brunskill and Alan Samuel

School of Computing, Staffordshire University, P.O. Box 334, Beaconside, Stafford, Staffs ST18 0DG, UK.

Abstract

It is generally accepted that every programmer checks his or her code before submitting it to testing. This process has been formalised by means of 'reviews', 'walkthroughs' and 'inspections'. Inspections were used in industry by the author, following Fagan, but were not well-metricated: others have shown that the quantifiable benefits of inspections may be considerable. A final-year project at Staffordshire University has produced results for software inspection compared both to computer-based testing alone and to computer-based testing following inspection in which metrics were collected throughout the process, allowing comparisons of effectiveness and efficiency.

Keyword codes: D.2.4, D.2.5, D.2.8
Keywords: Program verification, Testing and Debugging, Metrics.

1. INTRODUCTION

Every programmer checks his or her code before submitting it to testing, and always has done. However, it is a well-established psychological fact [1], [2] that the reading of the program by a second, independent person is a useful technique for discovering errors that would otherwise go unnoticed. It appears that the independent checker does not need to be more experienced in the language, or better-informed in terms of likely sources of error; simply coming to the program fresh appears to lend a perspective denied the originator.

Basili [3] showed, in a controlled study, that code reading was more effective than either functional testing, or structural testing, although the participants believed the contrary. In practice, informal "cross-checking" of each other's work has been common in many development environments, as programmers recognised that a second view of a program could quickly highlight errors that would otherwise be missed, or not be discovered until formal testing, when a considerable amount of rework would result.

In his seminal work on software testing, Myers [2], following Fagan [4], recognised the significance of code inspection in the process of developing correct programs, and gave some of the earliest and most practical guidelines for conducting the process. More recently, Gilb and Graham have gathered together the threads of universal best practice in this area [5], and proposed a comprehensive methodology for the process of software inspection, including the use of metrics.

M. Lee et al. (eds.), *Software Quality and Productivity*

2. METHODS FOR PROGRAM VERIFICATION

This paper does not discount software testing as a necessary technique in the process of producing correct software; however, the role of testing *per se* is considered separately in Section 5. For now, however, we wish to concentrate on methods other than computer-based testing techniques.

2.1 Desk-checking

The process of program evaluation by means of peer assessment is well-established in general. The way in which this cross-checking of work is conducted, however, varies from establishment to establishment.

In some situations, it may be that such verification is never formally stated as a requirement of the software development process, but has merely evolved over time. In the author's experience, this has tended to arise quite naturally among teams which have worked together successfully on a number of projects, with relatively few personnel changes. In such an environment, staff tend to be relaxed, confident of their own abilities, and unthreatened by criticism.

Alternatively, program verification by desk-checking may be an institutionalised, formal process, in which programs produced by junior programmers are evaluated by more senior personnel - often the designers - and/or by the project leader. Such a process tends to evolve quite naturally into one of the methods considered below.

2.2 Reviews and Walkthroughs

As noted by Yourdon [6] and others, the distinction between a Formal Technical Review and a structured walkthrough is not always clear: the phrases are frequently used interchangeably in practice.

However, current best practice suggests that the word 'Review' should be reserved for a consideration of project development at milestone points: statistical data may also be produced at such a time for QA purposes (see, for example, [7]). The personnel involved will typically be a chairman, the author of the work under review, a QA representative, plus one other independent reviewer, who has some expertise in the area under consideration. Reviews may apply to any stage of the lifecycle, and their primary purpose is to provide information to management, to answer the question, "Does this product do what it is supposed to do?" [8].

A structured walkthrough applies normally to the analysis and/or design stages of software development, and records data on issues such as completeness, correctness and integrity of the entity-relationship diagrams, data dictionary, and process specifications; and on structure charts, module specifications, etc. A *code walkthrough* is usually equivalent to a code inspection, as originally defined by Michael Fagan at IBM [4].

2.3 Software Inspections

Inspections are now primarily seen as a part of the software quality improvement process, in which defect detection occurs routinely, and as a result of which the development process is improved. There is, additionally, much evidence in the literature (e.g., [9], [10], [11]) as well as anecdotally, of considerable cost-benefit.

As originally proposed by Fagan, design and code inspections were seen as a formal, efficient and economical way of finding errors; an inspection team was assumed to comprise members with specific roles to play, viz:

- moderator: a competent technical member of staff specially trained in inspection techniques who should have a galvanising and synergistic effect on the team members;
- designer: the person who produced the piece of work under inspection;
- coder/implementer: the person responsible for translating the design into code;
- tester: the programmer responsible for writing and executing test cases.

It has been recognised during the intervening years that all personnel involved in inspections must be committed to the use of this method. This implies that managers require at least one day's training to familiarise them with the technique, moderators require three days' training, and participants need at least a half-day introduction.

The most impressive results, in terms of defect reduction and cost-benefit, have been realised where senior management are committed to this method of working, as an integral part of the process of quality improvement.

3. AN INDUSTRIAL PERSPECTIVE

3.1 Background

During 1990 and 1991, the author led a software development team whose brief was to design an electricity prepayment metering system, making use of contactless smart cards. This involved considerations of data security, communications, and the human-computer interface, as well as database management, and inspections were introduced at the design stage in order to improve the quality of the resulting system.

3.2 Introduction of Inspections

None of the team members involved in inspections were specially trained in the techniques, but all were impressed with the results claimed. A member of the Quality department was present at the inspections in addition to the suggested team members, and a senior designer from another project was co-opted as moderator. No meeting lasted more than 2 hours.

As each module was designed, its author made a formal request for inspection, and the relevant documents were circulated in advance to all participating staff. Subsequently, the inspection meeting was convened, and the design checked, under the direction of the originator. Errors were categorised as 'Missing', 'Wrong' or 'Extra', and were also categorised by problem type. The log was then circulated to all participants, and a follow-up meeting rechecked the design, after the errors originally identified had been dealt with; if necessary, a third iteration of the process occurred.

3.3 Results

It is clear, with hindsight, that the major weakness in the process as described above was the lack of senior management commitment. It was the software developers who initiated the

process, believing it would improve quality and reduce overall development time, and in the event the use of inspections was vindicated. The final product was delivered on schedule and worked correctly. Errors did occur subsequently in operation, but the delivery of correctly-working software on the originally-scheduled date was in contrast to experience both with projects which had preceded this, and others which were under development at about the same time.

Fagan identifies six operations as essential in the inspection process: Planning, Overview, Preparation, Inspection, Rework and Follow-up [9]. In the work as described above, these stages were not strictly adhered to: in particular, there were unclear entry and exit criteria; the process did not follow best practice; and the preparation was haphazard. Further, no metrics were formally collected for the inspections, along the lines of, for example, defects found/page, defects found/hour, time for correction, or even time spent in inspections, and time for software testing. Nevertheless, the experience was enough to persuade the personnel involved that the process had produced real quality-improvement results, and that it should be retained and cultivated for future projects.

4. AN ACADEMIC PERSPECTIVE

4.1 Background

A final-year student at Staffordshire University, Alan Samuel, compared human-based testing techniques with computer-based testing techniques, on three small program units, written in C, COBOL, and FORTRAN. The inspection and testing processes were monitored by collecting metrics from start to finish, allowing calculation of effectiveness, efficiency, and cost saving.

The experiment was prompted in part because, as noted by Bisant and Lyle [12], inspections were developed in the industrial sector, rather than in an academic environment, and most of the data available to date has been produced in industry. A further motivation was the desire to attempt to evalute the extent to which the use of inspections could be incorporated in the undergraduate curriculum.

4.2 Methods Used

The software inspection technique was the process as originally defined, and as developed and amplified by Jones and others [13], [5]. The computer-based testing required some consideration: since all the programs were small (about 300 lines of code) and of only moderate complexity, it was decided to use structural testing based on loop-modified path coverage. Thus, a set of test inputs was generated whose number was quite manageable, but which were sufficient to provide test effectiveness at the TER3 = 1 level.

The participants in the inspection process were also students, many of whom were reading for higher degrees following a period of industrial experience. In all, 12 different personnel were involved in the code inspections, their backgrounds ranging from system administration to mechanical engineering. None of these volunteers had any prior experience of software inspection, and so had to be introduced to the method, its aims, techniques, and outcomes. The testing was performed by Samuel, who proceeded in each case by drawing the program flowgraph, determining the McCabe complexity of the program, and then deriving the basis test cases. The overall strategy involved each program being first inspected, as desribed above, and metrics noted for time taken, number of majors/hour etc. The program was also compiled, debugged, and tested independently, and metrics noted for this process. Finally, each program

was tested *after* corrections due to inspection, and similar data collected for this combined process.

The C and Fortran programs were specifically written for this experiment, making use of standard programming methods and techniques as taught at Staffordshire University. They were not evaluated in any other way prior to the experiment taking place, and so represented typical 'first-cut' programs. The COBOL program was a module taken from a much larger order-processing system which was running live at that time in business, but which suffered extremely high maintenance costs, and typical downtime of 3 - 5 hours per week. This module was of particular interest for these reasons: however, because the 'copy' and data files were unavailable, it was not possible to compile the module, and so for testing, only static analysis was possible, and results for effectiveness and time saved may be estimated only by inference.

4.3 Results

The results may be summarised as shown in Tables 1, 2 and 3 below.

Table 1: Time taken for code verification.

	C Program (hrs)	Fortran Program (hrs)	COBOL Program (hrs)
inspection alone	13.6	10.9	10.4
test alone	16.5	9.5	n/a
inspection + test	5.5	14.9	n/a

Table 2: C program Statistics.

	Efficiency (Majors/hr)	Effectiveness* (Majors)	Time saved (hrs)
inspection alone	0.73	60%	2.9
test alone	0.34	35%	-
inspection + test	0.54	60%	11.0

Table 3: Fortran program statistics.

	Efficiency (Majors/hr)	Effectiveness* (Majors)	Time saved (hrs)
inspection alone	0.55	60%	4.0
test alone	0.63	30%	5.4
inspection + test	0.40	60%	-

*Effectiveness calculated as follows: assume inspection discovers 60% of all errors (historically acceptable). Then testing effectiveness is determined as the proportion of known *and assumed* errors discovered by this method.

For the COBOL program, a total of 24 issues were raised during inspection, at a rate of 0.5 issues/minute. A total of 5 majors was discovered. The total time taken to correct the major defectives was 10.4 hours, i.e., an average rate of 2.1 per major. If the assumption is made that the time to fix a typical error on the live system is 5 hours (representing a single corrective maintenance effort), then the total maintenance time required to correct the faults will be 25 hours, representing a notional time saving due to code inspection of 14.6 hours.

5 SUMMARY AND CONCLUSIONS

Clearly, this experiment is limited in its scope and no very general conclusions may be drawn from such an exercise. A number of points are relevant, however, in the context of the activity.

5.1 Some Metrics for this Experiment

Inspection proceeded at a rate of approximately 1 page per hour; this compares favourably with Fagan's recommended rates of 90 noncommentary source code statements per hour as a rule-of-thumb, and 125 noncommentary source code statements per hour at a maximum. The errors uncovered implied an error rate immediately after coding of 1 per 8 lines for Fortran, 1 per 30 lines for C, and 1 per 40 lines for COBOL. These figures are higher than one would expect for the Fortran program, but not untypical for the others (e.g., 1 error per 18 lines of C code is not uncommon [14]).

It should also be noted that

- the participants were untrained in software inspection
- the programs considered were small
- it was not possible to compile and dynamically test the COBOL module.

Additionally, the time taken for structural testing assumed that the whole process was being done by hand. This perhaps gives a fairer comparison of the efficacy and efficiency of the two methods, but clearly, static and dynamic analysis tools have much to offer in this area: the best of the currently-available tools are capable of discovering quite subtle errors. Additionally, of course, inspection cannot do many of the things that testing alone can do: for example, give a measure of performance or reliability.

5.2 Pedagogical Issues

A subsidiary reason for conducting the project was to establish the extent to which software inspection would be useful as a technique by means of which undergraduate students could improve both their productivity and the quality of their programs. It is clear that inspections would be of real value in this sense, as well as having a bonus effect in preparing students for current best practice in software development when they graduate. There are some administrative and pedagogical issues to be considered, however, in that for the most part, university students are expected to produce individual pieces of work, and collaboration is discouraged. Nevertheless, group projects are undertaken at various stages in the course of a degree, at Staffordshire and other universities, and it would seem appropriate to introduce this method in that area. This would allow the students to apply the techniques of software inspection in accordance with the latest thinking, and in an environment which would allow them to develop the skill without the constraints of commercial practice. As well as the

introduction of software inspections in this way, it is intended to develop the work described in this paper with a future final-year or Master's student: ideally, a metric relating correctness of code prior to testing as a function of (among other things) effort devoted to software inspection would be the result of such work.

As software systems develop in size and complexity, and the number of systems which are in some sense mission-critical increases, the question of software quality assumes an increasing importance, and defect prevention becomes more of an issue than defect detection. The relevance and effectiveness of software inspection in achieving higher-quality software is probably no longer debatable, but difficulties still exist in persuading industry at large of this. The terms 'walkthrough', 'inspection' and 'review' are also widely misused, or misinterpreted, in the commercial environment. However, the data in this paper suggests strongly that a properly-managed combination of inspection and computer-testing offers the most effective and efficient way to produce reliable software. Additionally, the adoption of inspections, as currently formulated, as an integral part of software engineering undergratuate education, would play a useful role in disseminating this expertise more widely in practice, and in altering the perception of the activity of producing high-quality software, as a quantifiable, cost-effective process resulting in continuous improvement.

REFERENCES:

1. WEINBERG, G M, 'The Psychology of Computer Programming', (Van Nostrand Reinhold, 1971).
2. MYERS, G J, 'The Art of Software Testing', (Wiley-Interscience, 1979).
3. BASILI, V, 'Software Metrics', *Programme for the Open University, U.K.*, 1987.
4. FAGAN, M E, 'Design and code inspections to reduce errors in program development', *IBM Systems Journal*, 1976, Vol 3.
5. GILB, T and GRAHAM, G, 'Software Inspection', (Addison-Wesley, 1993).
6. YOURDON, E, 'Structured Walkthroughs', (Prentice-Hall, 1977).
7. ARTHUR, L J, 'Improving Software Quality: An Insider's Guide to TQM', (Wiley, 1993).
8. WEINBERG, G M, and FREEDMAN, D P, 'Reviews, Walkthroughs, and Inspections', *IEEE Transactions on Software Engineering*, Jan 1984, 10-1.
9. FAGAN, M E, 'Advances in Software Inspections', *IEEE Transactions on Software Engineering*, July 1986, 12-7.
10. ACKERMAN, A F, BUCHWALD, L S, and LEWSKI, F H, 'Software Inspections: An Effective Verification Process', *IEEE Software*, May 1989.
11. CHRISTENSON, D A, HUANG, S T, and LAMPEREZ, A J, 'Statistical Quality Control Applied to Code Inspections', *IEEE Jnl Selected Areas in Communications*, Feb 1990, 8-2.
12. BISANT, D B, and LYLE, J R, 'A Two-Person Method to Improve Programming Productivity', *IEEE Transactions on Software Engineering*, Oct 1989, 15-10.
13. JONES, C L, 'A Process-Integrated Approach to Defect Prevention', *IBM Systems Journal*, 1985, 24-2.
14. HATTON, L, 'Automated Incremental Improvement of Software Product Quality: A Case History', *Proc. Software Testing, Analysis and Review Conference (STAR '93), London, Oct 1993.*