SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 22(2), 173–182 (FEBRUARY 1992)

Experience with Fagan's Inspection Method

E. P. DOOLAN
Shell Research B. V., P. O. Box 60, 2280AB Rijswijk (Z-H), The Netherlands

SUMMARY

Fagan's inspection method was used by a software development group to validate requirements specifications for software functions. The experiences of that group are described in this paper. In general, they have proved to be favorable. Because the costs of fixing errors in software were known, the payback for every hour invested in inspection was shown to be a factor 30. There are also other benefits that are much more difficult to quantify directly but whose effect is significant in terms of the overall quality of the software.

Some pointers are given at the end of this paper for those who want to introduce Fagan's inspection method into their own development environment.

KEY WORDS Software quality assurance Fagan's inspection method Software review Defect detection Software requirements specifications

INTRODUCTION

In this article we describe the experiences a software development group had with Fagan's inspection process as a means of verifying and validating software requirements specifications for functions within a large production software package. The package in question exceeds 2 million lines of Fortran, contains more than 250 separate high-level geophysical functions and runs on a wide range of hardware configurations.

The package is continually being updated. A new release of the product is made available to the user community twice a year. In a typical release at least one-quarter of the total code has been changed in some way, be it as a result of fault fixing, the addition of new processing functions or the disposal of obsolete ones. The releases are shipped together with comprehensive user documentation and installation instructions, and are prepared for running on an operating company's specific hardware configuration.

The enormous flexibility of the package results from a software platform that minimizes the duplication of functionality by making the available functionality serve multiple purposes. The development, support and maintenance of this platform and the processing functions it contains is carried out by the Seismic Software Support Group (SSSG) at Shell Research. This group contains some 25 people with professional skills in software engineering and geophysics.

To maintain a competitive edge in the seismic-processing world, a strong emphasis must be placed at all times on the quality of the package. The SSSG is continually

0038–0644/92/020173–10$05.00

Received 24 September 1990
Revised 2 May 1991

seeking methods to ensure a higher standard for its product, and it is on one aspect of this search for continual improvement that this paper concentrates.

SOFTWARE QUALITY ASSURANCE

Early in 1987 it was decided to bring the co-ordination and management of all software-quality activities under the purview of one independent quality-assurance group, whose task was defined to include the guiding, monitoring and promoting of activities related to quality assurance and control within SSSG. One of the first activities of this group was to put a price on the non-conformance [1] of the SSSG'S software development and maintenance activities. The objective of this exercise was to locate, using, for example, the simple Pareto principle, the stage of the software life cycle in which an improvement would most benefit the quality of the package as a whole.

NON-CONFORMANCE ANALYSIS

The two most 'expensive' items on the list of costs attributable to non-conformance were fault-report fixing and project definition.

Fault reports are descriptions of anything that a user of the software finds unsatisfactory. The subjects of these reports range from errors in the software (code or documentation) to suggestions for improvement and requests for enhancements. For the package in question, between 300 and 400 fault reports a year are received. An analysis of these report: shows that nearly half were suggestions for improvement or requests for enhancements to the existing software. Further analysis of these 'faults' shows that many were obviously items that should have been dealt with in the software from the outset and could have been, had more care been taken with the specification of the original product.

The average cost for the SSSG to fix a fault of any type in the package in question is of the order of one man-week (approximately $3500). This figure includes not only the manpower costs but also the costs of computer time, testing time, configuration management, fault-report database updates, user notification and the generation of release information. Taking the number of fault reports that can be classified as suggestions or requests for enhancements and that should by all accounts have been taken care of by the original software and simply multiplying it by the average cost of fixing an arbitrary fault results in annual non-conformance cost of 3½ man-years.

Project definition entails the definition and validation of the scope and requirements of the project. The non-conformance costs associated with this aspect of a software project were even greater than those associated with fault reports. In the three year period between 1984 and 1986 two projects had to be shelved late in their development cycles clearly because of inadequate requirements specifications. The total manpower invested in these two projects was more than 10 man-years. Furthermore, there were many smaller projects for which the development time could clearly have been reduced if more emphasis had been placed on the drawing up of the original requirements. The annual non-conformance cost due to improper project definition and requirements analysis amounted to at least 4 man-years.

Taking these two major sources of non-conformance costs together yields an annual chronic waste [1] of some 7½ man-years or, equivalently, more than one-quarter of the SSSG'S total manpower budget at that time.

SOFTWARE REQUIREMENTS SPECIFICATIONS

Both sources of chronic waste identified above arise because not enough effort was devoted to the requirements specification phase of projects. Indeed, until 1987 the SSSG paid almost no attention to this stage of the software life cycle. Accordingly, any requirements analyses that had been carried out were more the exception than the rule.

In the light of the possible gain that could be achieved, the software quality-assurance group decided to start placing more emphasis on the software requirements specifications (SRS). Accordingly, guidelines were drawn up for how a software requirements specification should look, using the ANSI/IEEE standard 830 [2] as a model. Furthermore, the specifications were to be expressed in ordinary English, as opposed to any formal language. There were two reasons for this.

First, a document written in English was generally thought to be better received in the geophysics world. The people in this world are processors (users of the resulting software) and researchers. Although they could be assumed to have a background in physics or mathematics, very few felt at home with a formal-language specification. It was considered imperative to have the opinion of these people on the proposed product if the maximum benefit was to be reaped from the specification phase.

Secondly, the choice of a natural language was almost forced upon us by the fact that none of the formal languages available were really capable of handling anything more than the simplest of problems, and the geophysical functions being built by the SSSG were anything but simple. Having decided to use a natural language to convey specifications, the guidelines for writing such specifications were extended with sections on how to avoid some of the pitfalls inherent in this decision.

Since reviewing software items was an accepted practice in the SSSG, it followed naturally that this would also be done for software requirements documents. A survey was made of all known review methods, and their various pros and cons were weighed against one another (see, for example, Reference 3). After due consideration, it was decided to adopt Fagan's inspection as the SSSG'S review method. No other experience was available within the Shell Group at that time, but the idea behind it seemed appealing and the results noted in the literature were impressive.

FAGAN'S INSPECTION—AN OVERVIEW OF THE METHOD

Fagan's inspection [4–6] is a review process developed by Michael Fagan at IBM in the 1970s. Fagan was a professional quality-control engineer who derived his basic ideas on statistical quality control from two of the gurus of quality management, Deming and Juran. The inspection process is akin to walkthroughs, although it differs significantly in some aspects. It can very briefly be described as follows.

An inspection is organized by a moderator, who may be appointed by the software quality-assurance group. The moderator receives a copy of the document to be inspected and checks that it satisfies a number of predetermined criteria (entry criteria). He or she then puts together an inspection panel of no more than five people (including the author of the document to be inspected).

The inspectors are then invited to attend a short (20–30 minutes) 'kick-off' meeting. At this meeting the objective of the inspection is defined, the subject-

matter briefly explained and any other relevant details are discussed. The inspection material is distributed and roles may be assigned to some or all members of the panel, with requests to pay specific attention to some aspect of the documents to be inspected.

After the kick-off meeting, each member of the inspection panel studies the document that has been submitted for inspection (the low-level document) in conjunction with the input document to this low-level document (the high-level document), as well as any standards and checklists that the moderator has considered relevant, with the objective of identifying defects and omissions in the low-level document. The high-level document for the software requirements specifications inspected by the SSSG, for example, is a three-page document that defines the scope of the project. All documents used in an inspection are themselves also subject to inspection.

The inspection panel then comes together again at a specified date for the defect-logging meeting. At this meeting, which should last no longer than two hours, every defect discovered (both on one's own and during the meeting itself) is logged. A causal-analysis meeting is then organized in which the inspectors discuss the cause of (some of) the uncovered defects and propose possible ways to prevent these types of errors from occurring in future documents. (The causal-analysis meeting is a later addition to the inspection process and was not an explicit integral part of the process originally developed by Fagan.)

Thereafter the moderator checks that, for each defect discovered, the appropriate author has taken some remedial action. (It does not necessarily have to be an update to the document.) The moderator is also responsible for gathering statistics so that all the techniques of statistical quality-control can be applied to the whole inspection process.

FAGAN'S INSPECTION IN THE SSSG

Fagan's inspection was first tried by the SSSG in March 1988. As of August 1989, 11 software requirements specifications totalling some 500 pages had been inspected.

The inspection panels consisted of between four and six people, including the moderator, who also served as an inspector (in theory he or she does not have to take on this role). The panel members are chosen from the geophysical research group, the experimental seismic processing group (responsible for evaluating and testing new software), a production processing group making full-time use of the package, and the SSSG itself. This mix ensured that the most important aspects of the SRS would automatically y be judged by those most qualified to judge them; the moderator occasionally assigned other specific aspects of the SRS to certain inspectors.

Inspectors studied the specifications on their own at an average rate of five to six pages per hour. The average number of defects logged per page was between two and three, usually fairly evenly divided between major and minor defects (see below for the definitions of 'major' and 'minor'). Seven to eight pages of the SRS were covered on average per hour of the defect-logging meeting, which was kept under two hours. One defect was logged every three minutes at these meetings. Of the total time spent on an inspection (including kick-off meeting, administration and self-study), an average of 2·2 h were spent on each page of the SRS.

Although the average number of pages covered in an hour of a defect-logging meeting falls within the limits as stated in the literature (between 5 and 20), the number of defects logged in these meetings is nearly a tenth of what can be achieved. One reason for our low defect-logging rate is that some discussion about the defects was allowed during the meeting; Fagan's official rules of inspection prohibit discussion of any kind. The argument against discussion presumes that the more defects logged per minute, the more cost-effective the inspection is. However, our experience has shown that discussions can actually have a stimulating effect, since many additional defects were discovered during such discussions. There are other reasons as well why such discussions took place in the defect-logging meetings. Deficiencies in existing standards meant that some *ad hoc* solution had to be adopted before the meeting could proceed. Furthermore, because the high-level document itself had not been inspected, serious issues often arose over the scope of the project as formulated there.

To clarify this last point, it should be noted that the original guidelines drawn up by the SSSG for writing an SRS closely followed ref. 2. There, the scope of the requirements specification is an integral part of the SRS itself. Because of this, the scope was not at first inspected as a document in its own right. Consequently, the serious issues that arose precisely with this scope, the inspection-panel members increasingly felt that the high-level document defining the scope of the SRS should be issued as a separate document and inspected first. Since this item is quite short (three pages), its inspection can be quick.

This issue illustrates one of the effects of doing an inspection. As people become aware of the tremendous benefits of the inspection process, there is an increasing desire to apply it to other software items, such as user documentation and code. In this way, inspection breeds inspection.

As has been noted in the literature, standards come to life through inspections. They no longer remain on the shelf, rarely consulted; instead, they become serious working documents. One effect of this renewed application of standards, of course, is that many defects are discovered in the standards themselves. Also a need for more and better standards and guidelines arises to curb the tendency to discuss issues that amount to a matter of taste. Valuable inspection time should not be taken up with these types of issue, and the easiest way to achieve this is to agree on general rules by which these questions can be quickly resolved. The resultant solutions, of course, are automatically inspected every time they are used and therefore evolve into something that represents the best current thinking on the subject-matter.

Even if the panels inspect higher-level documents, improve standards and introduce more guidelines, their defect-logging rates for software requirements specifications are not expected to increase beyond one per minute. The synergetic effect of bringing together a group of people who have studied the material thoroughly is currently too great to disallow all discussion, even though the prohibition is stipulated in the rules of Fagan's inspection. After all, the real argument against discussion is that it is not a profitable use of the inspectors' time, but that has not been our experience to date. We find discussion to be useful, but it should be tightly controlled by the moderator.

COST-BENEFIT ANALYSIS

The effectiveness of the SSSG'S inspections can be gauged by estimating the pay-back (in terms of costs saved) for every hour invested in inspecting. To do this, we must first classify the defects found by the inspectors.

THE CLASSIFICATION OF DEFECTS

Currently, we use three classes of defects. These are minor, major and super-major. The categories can be described as follows:

1. *Minor defects.* These range from something like a spelling mistake (which may be truly trivial) through sentence-construction errors (thereby possibly leaving the meaning open to misinterpretation) to clarification issues. These last defects arise when the specification raised some question in an inspector's mind that was left unanswered in the text. Such issues can be major omissions on the author's part (in which case they would be classified as major errors), or they may simply need the addition of a clarifying sentence indicating the 'lie of the land'. In the latter case these errors would be classified as 'minor'.
2. *Major defects.* These can be identified very simply: had the software been produced according to the uncorrected specification, the 'major defect' would have resulted in a fault report. Many of the major defects uncovered in requirements specifications for the software produced by the SSSG have to do with the user interface and the choice of default values. Filling one or two places on the inspection panel with an experienced processor, whose prime function is to check that the proposed user interface is comprehensible, accessible, logical and as user-friendly as possible, guarantees that a number of major errors are always detected in this area. A second source of major errors is incorrect or missing functionality or some vagueness in the specification that would have led to serious misinterpretations.
3. *Super-major errors.* These are errors so serious that they would have made the software virtually useless or, at any rate, would have caused it to deviate from existing standards and/or processing techniques to such an extent that chaos would have ensued. If left undiscovered in the original specification, to repair such an error in the final product would require a complete or, (at the very least) substantial redesign and rewrite. One example of such a problem is a specification requiring an input file that was almost impossible to create in a standard seismic data-processing sequence.

COSTS OF FIXING DEFECTS IN RELEASED SOFTWARE

To obtain some idea of the value of inspection, we assume that, had the requirements specifications not been written down and reviewed with Fagan's method, the detected defects would have appeared in the final software product. It is generally accepted that the cost of fixing problems in released software can be as much as 80 times more than that of fixing them at the specification stage. So there is clear benefit to be derived by getting the software requirements specifications down on paper and reviewing them by almost any method.

We have defined a major defect as something that would have resulted in the

generation of a fault report, had the software been written to the original specification. And the total average cost of fixing an error described in a fault report is known to amount to one man-week for the package produced by the SSSG. This figure takes into account all the costs involved, including testing, documentation update, configuration management, machine resources, etc.

RETURN ON INVESTMENT

For the sake of argument, we assume that every 25 minor defects would have resulted in the generation of one extra fault report. This is not unreasonable since a minor textual error could easily lead to a misinterpretation of the information presented. We also arbitrarily assume that a super-major defect is equivalent to 10 major defects. (We would expect in general that most super-majors would be detected before the software was released, even without inspection, so that the costs incurred would be less than if the software was released as specified; hence the relatively low figure of 10.)

Basing the development and maintenance effort saved by inspection on the classification of defects into minor, major and super-major and the costs of later repairing the three types of defect as outlined above, then we can show that for every hour invested in inspection nearly 30 are recouped. In other words, the time it would have taken to repair the detected defects, had they appeared in the released software, is 30 times greater than the total time invested in the inspection. Taking into account that the SRSS produced by the SSSG correspond to software products requiring between two and three man-years' development effort, we can alternatively state that inspection results in an average saving of 16½ man-months of software development and maintenance time over the course of the software's lifetime (this is simply the average time it would have taken to repair the defects detected by inspection, had they appeared in the software). Table I shows the return on investment per inspection.

THE UNQUANTIFIABLE BENEFITS

Along with the monetary benefits mentioned above, the inspection process also has quite an array of benefits that do not lend themselves to quantification in financial terms. They are no less important for that reason, however, and in fact can contribute substantially to the quality of the software in general. The most widely relevant of these defects are listed below.

(a) The package developed and supported by the SSSG is 15 years old. To make optimal use of it while cutting down on maintenance and programming costs, continued upgrading is necessary. Inspections are found to be a very good mechanism for highlighting and prioritizing candidate areas for enhancement in this respect.
(b) The inspection process promotes team work, and is a good means of developing team spirit in a large group.
(c) Inspection is an excellent means for transferring technology and can accordingly serve as a back-up mechanism, should key people be suddenly removed from the project.

Table I

SRS inspected	1	2	3	4	5	6	7	8	9	10	11
Number of pages in document	3	58	23	32	59	23	41	126	22	15	55
Number of inspectors	5	7	6	6	6	5	6	6	7	5	6
Total shelf study time in hours	10	236	34	20	21	11	39	39	13	49	30
Pages studied per inspector per hour	2	2	4	10	17	11	6	19	12	1	11
Defect-logging time in minutes per page	30	8	23	5	6	1	5	8	2	8	4	8	8
Major defects logged	19	85	66	25	39	55	31	41	15	125	30
Super-major defects	—	2	1	—	—	—	2	1	—	8	3
Minor defects logged		46	50	24	41	50	96	91	27	127	91
Minutes per defect logged	3	3	4	3	4	3	2	2	4	3	3
Defects logged per page	9	2	5	2	1	5	3	1	2	17	2
Pages covered per defect-logging hour	2	8	3	11	11	4	8	30	8	1	8
Defects effectively detected per study hour	3	1	3	3	4	10	3	3	3	5	4
Total hours invested	22	306	100	47	64	49	80	77	38	131	85
Total hours invested per page	7	5	4	2	1	2	2	1	2	9	2
Return on investment in hours per hour invested	35	14	31	22	26	46	27	29	17	64	30

(d) The inspection process provides on-the-job training for employees who need to become familiar with:

(i) *Standards.* They are actively applied in inspections and become documents that represent the best working knowledge in the area in question.

(ii) *Technical material* Inspections provide an excellent way of introducing people to a new technical subject. They can still perform as inspectors, if they concentrate on a subject with which they are already familiar, but along the way they learn new technical details.

(iii) *Culture.* By having new people attend inspections (either as inspectors or observers), they are quickly introduced to the group's working methods and ways of doing things. To this end, observers should also attend the causal-analysis sessions.

(iv) *Inspection.* One of the best ways of learning about inspection is actually doing it.

(e) Inspectors identify the root causes of detected defects (in the causal-analysis meetings) and can propose modifications to the software development process that will prevent similar defects from occurring in the future.

SOME POINTERS FOR STARTING UP

A number of pointers are listed below for those who want to adopt Fagan's inspection as a review technique. These pointers have been culled from our experiences with the method and the lessons we have learned along the way. In particular, the wide range of returns on investment shown in Table I has its origin in points (b) and (h) below.

(a) Starting a process such as Fagan's inspection is difficult. It is important that

the first project(s) be seen as a success by everybody. They should therefore be chosen with extreme care and should not be too large. Also, the people involved in these initial projects should be highly motivated and inclined to make both the projects and the process a success.

(b) Starting off is also difficult for another reason: none of the input material for the inspection sessions will have been inspected, and consequently it will probably be riddled with errors and defects. This tends to make the initial inspection sessions difficult, because time has to be spent trying to resolve the problems that the input documents should have resolved; there can easily be a tendency to get bogged down. A good moderator is essential if these problems are to be overcome.

(c) Stick to the inspection process. All the elements in it are essential if it is to achieve its potential. Giving any piece short shrift is one of the main reasons for failing to get the process going.

(d) When the potential of inspection begins to become apparent, pressure arises to inspect more and more. A stepwise approach should be taken to this problem: one new type of document at a time. Even so, the amount of material to be inspected can become very large. As a result, project schedules can become affected. Therefore, it is important to arrange for the training of new moderators as early as possible and to have them moderate a number of inspections under guidance before they get their 'wings'.

(e) It is vitally important to keep as many statistics as possible. Only with statistics can the power of the method be made visible to others, especially management. Be constantly on the look-out for ways to prove the benefits that arise from inspection, for these arise in connection not only with defect detection and prevention but also with on-the-job training and team building.

(f) Getting people from other groups to invest time in your inspection sessions can be difficult. However, it is essential that they do so if maximum benefit is to be derived from the process. Consequently, some effort should be expended in securing the full support of department heads and the panel members' superiors. Also, an account should be set up on which inspectors from other sections, departments, etc., can book their time.

(g) Constantly monitor the whole process and be prepared to make frequent minor updates to it while maintaining all essential elements (see (c) above). The kind of changes envisaged are those that have to do with the administration and organization of the process as a whole and the interrelationship of the various steps with one another. In this way the process can be tuned to suit the environment in which it takes place. Some of these types of improvement will be proposed as a result of the causal-analysis meetings.

(h) Because writing software requirements specifications is a difficult task, it should be assigned to highly qualified people. Otherwise, so much time tends to be spent on inspecting the SRS that, in effect, the inspectors end up rewriting it. This is a very poor use of their time and also leads to their becoming disgruntled and unwilling to participate in further inspections. The inspectors' time must be valued appropriately.

CONCLUSION

We have described our experiences of using Fagan's method to inspect software requirements specifications. We adhered to the inspection process as closely as possible but found that some discussion, tightly controlled by the moderator, greatly increased the synergetic effect of bringing together a group of people who had (individually) thoroughly studied the material in question.

A cost–benefit analysis of the defects uncovered by inspecting software requirements specifications according to the method of Fagan indicates that such a validation process is indeed worthwhile. However, we should not ascribe all the benefits of this process to Fagan's inspection methodology alone. One very clear message emanating from the emphasis placed by the SSSG on software requirements specifications is that the greater visibility and control afforded by merely getting these requirements down on paper already constitutes an enormous benefit.

Fagan's inspection is not only applicable to validating software requirements specifications; it can equally well be used to inspect any item (e.g. scope documents, user documentation, design, code, test plan, test results, etc.) produced during the software lifecycle of a project. In fact, there is no reason why it should be confined to the software world; there are citations in the literature to instances in which the method has been applied successfully to documents as far removed from software as engineering drawings. Any effort to apply it to other areas—management documents, for example—could be very profitable.

ACKNOWLEDGEMENT

The author thanks the members of various departments in Shell who participated in the inspections. Even though they were initially sometimes somewhat reticent, the results reported here would not have been possible to achieve without their enthusiasm, co-operation and willingness to participate in and discuss the process.

REFERENCES

1. J. M. Juran, 'The quality trilogy', *Quality Progress,* August 1986, pp. 18-24.
2. *IEEE Guide to Software Requirements Specifications,* ANSI/IEEE Standard 830, The Institute of Electrical & Electronic Engineers, Inc., 1984.
3. Daniel P. Freedman and Gerald M. Weinberg, *Ethnotechnical Review Handbook,* Ethnotech Inc., 1979.
4. M. E. Fagan, 'Design and code inspections to reduce errors in program development', *IBM System Journal,* **15,** (3), 182–211 (1976).
5. M. E, Fagan, 'Advances in software inspections', *IEEE Trans. Software Engineering,* **12,** (7), 744–751 (1986).
6. T. Gilb, *Software Engineering Design,* to be published.