Software Quality Journal **7**, (1998) 11–20

Painless improvements to the review process

JUHA IISAKKA and ILKKA TERVONEN

Department of Information Processing Science, University of Oulu, FIN-90570, Oulu, Finland

Received July 1997

Although the review process is defined in an extremely formal way, each company normally has its own way of tailoring it, because staff tend to see formal reviews as too heavy a method for practical software development. These tailored versions differ in type and in the grade of formality. This paper introduces some easy ways to improve the review process. The goal is to find steps which are simple enough for the management and staff to accept and efficient enough to improve the review process.

Keywords: review process, review types, improvement

1. Introduction

It is largely accepted that review, especially formal inspection, is the most effective means of finding defects, and thus plays an important role in the software development process. Although the review process is defined in an extremely formal way [1, 2], each company normally has its own way of tailoring it. These tailored versions differ in type and in the grade of formality. Companies attempt to tailor their process even though experts warn them of the dangers of turning away from the formal method. This tailoring is understandable however, as practitioners see formal review as being too strict a method. It also justifies our approach, because practitioners need alternative review methods that are defined at an appropriate level of formality for use in practice.

The purpose of this paper is to introduce minor steps for improving the review process. The goal is to make proposals that are motivated and painless to implement and use, but in order to achieve this there are three criteria the steps should meet.

First, management must accept the improvement ideas. As the staff are the most important asset in software companies, one cannot waste this resource because good employees' time means money. Management may accept expensive monetary investigations if it is believed that these will improve the staff's working, but at the same time, management will not approve anything if it suspects that it would detract from the primary goal of the company (i.e. the amount of deliverable code).

Second, the staff must also accept any improvement step. Of course, they will obey the management and implement the step, but use of new improvements might be little more than symbolic in practice.

Third, the improvement should be continuous and remain in use. The company (both management and staff) should quickly recognize its benefits. The best situation is when an

improvement in the software development process motivates a search for further improvements.

On the other hand, there are limitations on the implementation of our proposals. The development process in a company should be documented and formalized. If the staff includes many 'gurus' who code individually, without help and teamwork, there will be no mental background for reviews. In addition, the participants will have to accept criticism, as no one should be afraid when others spot errors in their work. The participants should regard zero-defect documentation as a personal advantage and be happy only when there are others to help them.

This paper is based on interviews carried out in Finnish software companies and on some classroom experiences at university. Since the interviews were confidential we are not able to give any facts about the companies involved, but this should not be a problem, as we regard these 'improvement steps' as universal for software companies. We first discuss the topic 'why reviews are important in software development'. Then, in Section 3, we consider the problems of reviews in general and give some suggestions. Section 4 presents a set of new, painless types of review, and finally the results are summarized and discussed.

2. Why is software review so important?

Weller [3] argues that reviews can instrument the software development process. We can easily believe this when we recognize the close connection between the review/inspection

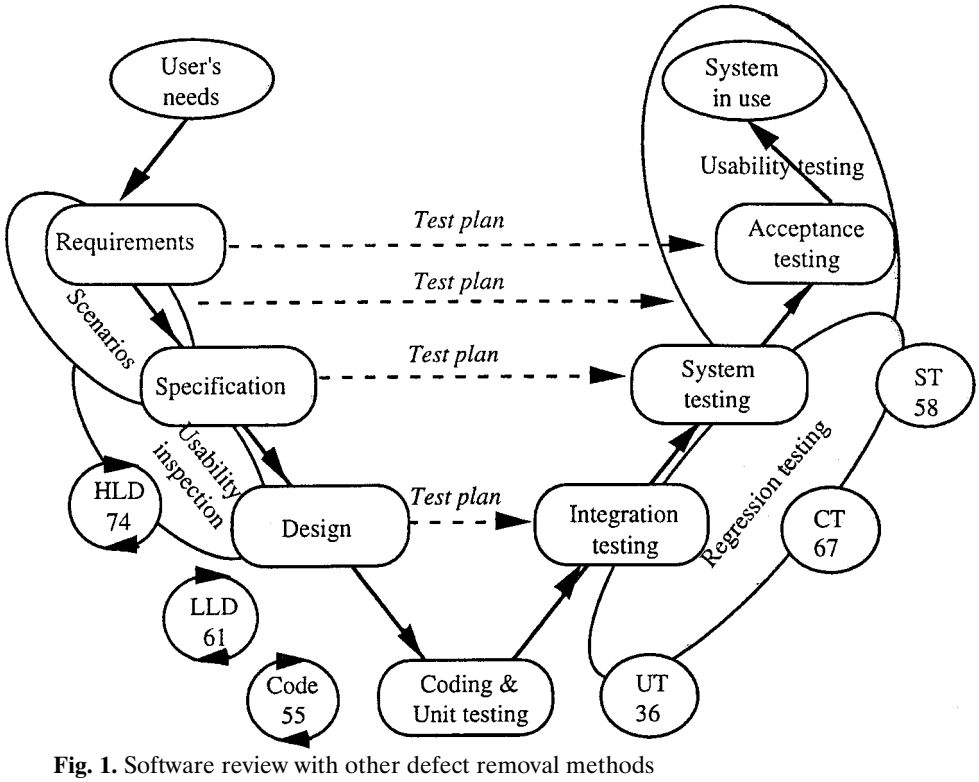

Fig. 1. Software review with other defect removal methods

and development phases as presented in the V-process model (cf. Fig. 1), for example. Review data can be used by the development staff in real time to make decisions about which process is best to follow, and the data also provide a better understanding of the dynamics of the development process. Review data collected from earlier development activities, requirements and designs for coding can provide measurable indicators of progress during these activities.

The V-process model forms an appropriate presentation technique for the further justification of review. The right side of the model explains the testing activities and testing phases, and the numbers inside the circles indicate the percentages of errors detected during unit testing, component testing and system testing. (Percentages from errors that should have been found in those phases [4].) Regression testing is based on the reuse of test scripts for different versions of the software product, while usability testing focuses on evaluation of the graphical user interface (GUI).

The small circles on the left side of the V-process model represent the role of review. Metrics literature [4–7] introduces reviews for high-level design (HLD), low-level design (LLD) and code phases and justifies the importance of defect removal at these phases with figures of 74%, 61% and 55% [4]. In addition, we also recognize scenarios and usability inspection, although their role is not so well known or so widely accepted as HLD, LLD and code reviews. Scenarios provide a way of defining the activities that will occur in the general use of a product, and are used to define the requirements and to model the use made of it, and thus act as a basis of further usability reviews and usability testing. Here scenarios may be used in test planning and to ensure that certain interface features are evaluated. Porter *et al.* [8], in their recent comparison of three methods for uncovering faults (issues) in software requirements specifications (*ad hoc*, checklists and scenarios), recognize scenarios as the most efficient for detecting faults. Unfortunately, we do not have any empirical data on the percentage efficiency of requirement reviews. Jones [9] estimated the value of a review method for requirements defect removal to be 'fair' in a context in which corresponding values for design and code defects were 'excellent'.

Mack and Nielsen [10] characterize usability inspection as a generic term for several methods, including at least the following eight; heuristic evaluation, guideline reviews, pluralistic walk-throughs, consistency inspections, inspections of standards, cognitive walk-throughs, formal usability inspections and feature inspections. If we place emphasis on users' participation, then pluralistic and cognitive walk-throughs are the most essential. When usability specialists are preferred to users, then heuristic evaluation is the method of choice.

3. Some general problems and suggestions

Review is traditionally defined with steps such as entry, planning, kick-off meeting, preparation, logging meeting, edit, follow up, exit and release. In this section we consider some problems discovered with the present review[1] process. First, we discuss the problem of insufficient education, and then consider the problem of code inspection, in which we focus on the interfaces of modules, in particular. Third, we discuss the problem of ranking reviews as secondary activities. Finally, we consider the general meeting problem in the case of the kick-off meeting. Under each topic we also present some suggestions for resolving the problems detected.

[1]We use the review here as a general term including inspection, formal review and walk-through, for example.

3.1 *Insufficient education*

Personnel tend to know quite well the terms connected with review, such as *inspection, review, scribe, logging, kick-off, meeting, issue,* but have no clear idea about their contents. One learns to review by following and participating in reviews. Of course, some employees may have had courses in inspection, but most (especially newcomers) will have no such education.

Typically, a company which has accepted reviews as a part of its development process standard will also possess guidelines and specifications for reviews. This material will be ignored without continuous training in review and control by management, and thus by and by new reviews separate from the standard of the company. In addition, all departments have evolutionarily and fortuitously created their own way of reviewing, which will differ from the original review method. We can discover three problems which influence this corrupting of review principles in particular.

(1) The management of the company may be committed to reviews and may demand them without motivating the project staff. This may result in the risk of the staff regarding them as a waste of time. This does not mean that the staff consider reviews useless, but that compulsory non-motivated reviews appear to have become symbolic occasions.
(2) Persons in charge and other more experienced employees in a software development project have a strong personality and status in the 'mental hierarchy' of the project, and give a very convincing example to inexperienced employees. When the most experienced participant in the review says: 'I am the last person to read material beforehand. I can just as well listen when the author explains his work!' all review soon become walk-throughs.
(3) The working conditions in the project have their own influence on reviews. If it is difficult to get reviewers together (when they are working in different locations) or there is a lack of competent reviewers, the situation can affect the value of reviews.

We propose that the staff should receive regular education in formal inspection methods. Gilb and Graham [2] argue that employees are content with formal inspection if they have proper experience in it. Though we do not agree entirely with this claim, we believe that the employees consciously manipulate their reviews – not haphazardly – if they have a deep knowledge of formal inspection.

3.2 *Focus on module interfaces*

The enormous amount of material involved is a common problem in reviews, especially in the case of maintenance-oriented software development. Normally the author has written or changed tens of lines of code, but this new code may belong to the older code in a highly unmodular way (perhaps only a line has been added in a module). The readers have to check whether the new code has an influence on the older code, and it is generally accepted that this is very laborious to do with procedural programs, and even worse with object-oriented programs.

At the same time the author will have made a unit test for his/her material which checks the interface of the code (interface between modules). We propose that the code reviews

should focus on the material of the unit test. By reviewing how well this test covers the interface, reviewers are able to verify what is the influence of the module on the other modules.

When reviewers are working with modules that act on a reviewed module, they are naturally interested in the effects that the module has on their own modules (e.g. the interface between modules and a common memory). The black-box review focused on unit test material is more understandable and better motivated for readers than the formal inspection of enormous modules.

Software development universally shifts the focus away from the code reviews. As an investment, it is much more profitable to aim at efficient specification or design reviews than code reviews.

3.3 *Reviews as 'secondary activities'*

The author him/herself is usually the scribe. Two things explain this situation. First, it is laborious to be a scribe. One may need a whole day to write down the record of a logging meeting. The scribe makes notes at the meeting by hand (there is seldom any PC in the room) and the official record has to be written up from the handwritten notes on a computer terminal or PC.

Secondly, it looks as if reviewing belongs to the secondary tasks in software development. One does not code anything useful when one is reviewing,[2] and consequently it is embarrassing to ask someone to be a scribe for one's product, knowing what a laborious job it is.

There is a danger, however, that the author and the reviewers may have different ideas on the material being reviewed. When the author has not understood a certain point in the job, for instance, he/she may misunderstand the issues raised in the review, and then write them into the record incorrectly.

We propose that companies should have tools to help the work of the scribe, because it would then be easier to ask someone else to act in this capacity. It is not necessary to write the whole records when there is a tool that will do part of the work.

Reviewing as a secondary action is also one reason why the author often presents his/her own work at review meetings. According to Gilb and Graham [2] the author ought never to be a presenter, as there is always a danger of unconsciously omitting unpleasant points in the work. The people we interviewed commented that it is worth noticing when the presenter (who is not an author) begins to waver, as this usually means a problem in documentation.

3.4 *An alternative kick-off meeting*

Gilb and Graham [2] present a version of the kick-off meeting in which the inspection leader divides the material to be inspected and the inspection roles among the inspectors. The kick-off meeting ensures that participants have all the necessary material.

It may be hard to gather participants even for logging meetings, however, and still more difficult for a kick-off. At the same time, participants may be motivated to review the

[2]There are still companies with the WHISCY spirit (Why Isn't Sam Coding Yet???[3]). This means that only coding is real work and all other duties such as planning, designing, reviewing, testing or learning are secondary occupations.

material anyway without any need for team spirit. The moderator (inspection leader) and the author are able to perform the kick-off meeting duties alone. After the author has decided that the material is ready for review, he/she will contact the moderator, and they can together choose the appropriate participants, gather the required material (e.g. source document, product document and checklists) and fix the roles of the participants [see 2]. Next, the moderator meets all the reviewers separately and discusses aspects concerning the review.

When the author is experienced in his/her job and knows the domain, he/she can write a statement on the material to be reviewed explaining where he/she has had the worst problems and what are the other critical parts or aspects of the material. Accordingly, the reviewers can focus on these potentially erroneous parts of the material. This is useful when time before the review is limited. If the author is inexperienced, however, or the domain area is new to the staff, the statement can only disconcert the reviewers.

4. New types of review

Preparation for the review (individual review) and the logging meeting form the major phases in the conventional review process. In our earlier paper [11] we divided the logging meeting into two types, public review and group review. In public review all reviewers are aware of all issues and capable of reacting to them. It is possible for all to discuss whether the issue under review is an error or not. It is not necessary to keep a review meeting if it is possible to discuss errors otherwise. Instead, a group review needs a face-to-face meeting. Face-to-face meetings typically provide a natural way of negotiating and collecting opinions. The problem is that these meetings also consume time (participants must move to and from the meeting place) and are difficult to arrange. A shift to more flexible implementations of meetings is thus an understandable alternative, although these raise some new problems such as insufficient understanding of the material and communication difficulties. The understanding problem may be solved by means of additional information presented in the form of a design rationale. Insufficient communication can be reduced by means of collaborative tool environments, which allow distribution of the material and collection of the pros and cons attached to alternatives, and also support decision making by providing summary reports of the justifications given by participants [see examples 11].

In this paper we further characterize these review types and introduce four new types, the *virtual logging meeting, limited logging meeting, pair review and no-logging meeting review.* The new types of review are compared with the logging meeting type in Fig. 2.

4.1 Virtual logging meeting

A virtual logging meeting is a combination of public and group review as introduced in [11]. Logging in public review (inspection) is managed by a moderator and may be implemented at the same time but in a different place or at a different time and in a different place, for example. This physical and temporal distribution of the review reduces the mutual time required and makes reconciliation of participants' timetables easier. The logging and recording of agreed, unique issues is one responsibility of the review leader. In a public review, support is required for flexible communication between participants. This is an area of innovative research in computer-supported co-operative work (CSCW) and organizational memory.

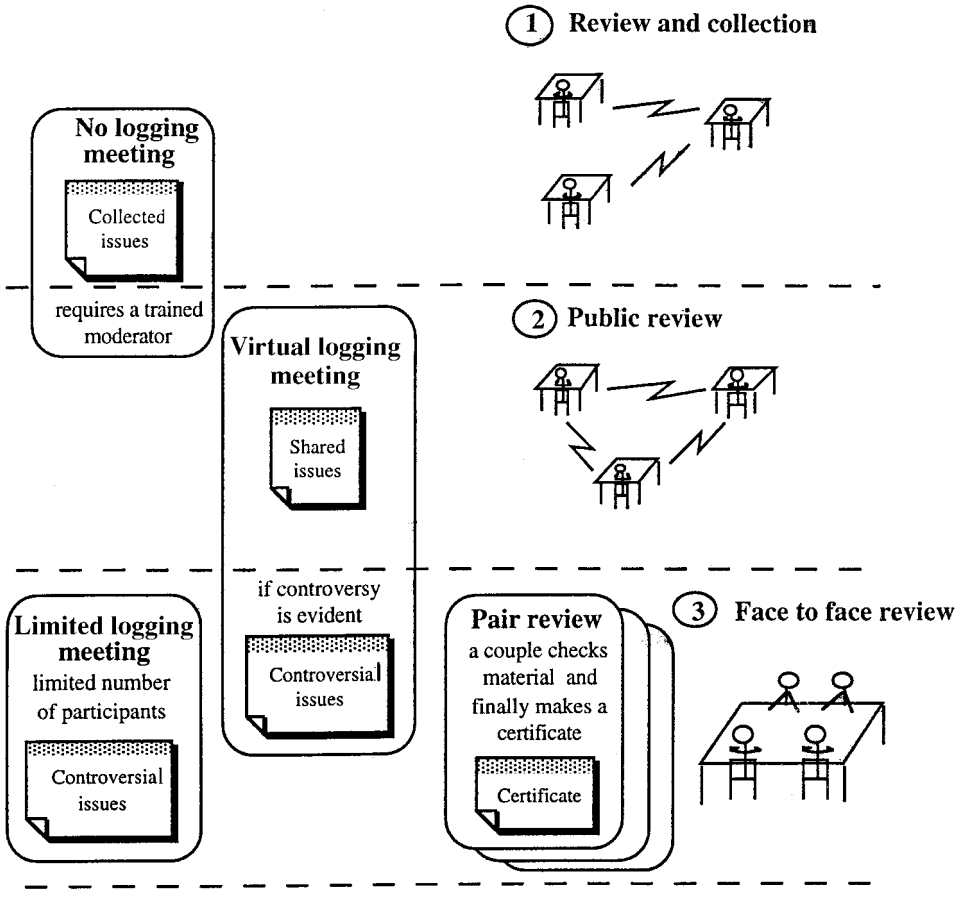

Fig. 2. The three different types of review meeting

If controversy is evident or discussion is required, a group review (face-to-face meeting) may be called, mainly to continue the discussion of open issues. i.e. topics on which the participants disagree. The review leader steers the discussion of controversial issues, the group discusses and negotiates unresolved issues, and the opinions are recorded. As this phase focuses on controversial issues (i.e. only a small part of the prepared material), the general guidelines for a review meeting (e.g. review rate) are not relevant.

4.2 No logging meeting

Borrajo [12] states in his fine article there is no need at all for review meetings. The traditional inspection style of reviewing demands a trained moderator who has a lot of work to do considering one review, whereas the participants should be motivated to find errors without anyone to guide them. The reviewers can send their issues and comments to the author privately and he/she can check them alone and decide whether there is any need for action or not. There is no need for a meeting at which a whole group can together say what is wrong with the material. One risk factor in this approach is the competence of the

reviewers. Borrajo suggests that the improvement proposals given by reviewers should also be evaluated by the authors. This would guarantee efficiency of the reviewers.

Review without a logging meeting is a special case of a virtual logging meeting. In the virtual logging meeting the reviewers do not necessarily meet each other face-to-face. Normally they just send their issues to the common 'notice board' where the other reviewers are able to study them and comment on them. If the comments from other reviewers are irrelevant and the author merely checks the issues, the virtual logging meeting has become a no logging meeting review.

We propose that the staff should consider this opportunity when working in a mature domain area and when software development is just daily routine. When there are problems with knowledge or technology, on the other hand, face-to-face meetings support the discussion of problematic issues and collaborative problem solving.

4.3 *Pair review*

We introduced the peer review as an alternative activity to individual review [11] because reviewers who are not familiar with the specific domain may require extra support. We can borrow these defined characteristics for our peer and pair review discussion. In peer review a software designer participates in the review and guides the review process. He/she explains the background to the design decisions and guides the review to focus on the topics that are most essential. The time spent may be considered from two contrasting viewpoints. Without the support of a designer it requires only the time of one person, but may be inefficient. The use of two people doubles the amount of time, but their cooperation may increase efficiency. In order to make the peer review effective, roles are assigned, with associated checklists, as in an individual review.

The pair review is a special type of peer review. Two employees make a designer–reviewer couple and change roles after a while so that they act as reviewers for each other. Because this is a 'permanent' couple and each inspects the other's work, we can assume that they take it more seriously than when participating in a normal review with many other people.

The pair review is useful for a company because there are always two people familiar with each difficult technical case. It is important that these two employees can collaborate. If there is a clash of personalities, they are not a valid couple.

The review practices of such a pair will evolve step by step during discussions and thus it is meaningless to formulate any strict rules of action. Due to the iterative nature of the process, the recording of 'temporary' issues (defects) found in informal meetings is not required. The issues are typically marked on paper documents and corrected during the next correction cycle. In the final (formal) meeting, however, the pair review needs a form or template of its own on which both the author and the reviewer guarantee that the material has passed the pair review.

4.4 *Limited logging meeting*

Votta [13] states that one wastes $n - 4$ hours of working time when holding a two-hour review meeting with n persons. Typically only the moderator, scribe and two reviewers will be concentrating on the topic issue at any one time, while 30–80% of the other reviewers may be listening and the remainder 'sleeping', discussing other topics or checking other

places in the material.

That is one reason why we propose that there may normally be no need for a full logging meeting with all participants. The other reason is that it can be hard to find a suitable meeting time for all participants, as staff may be taking part in many simultaneous projects and working in different locations. Due to project deadlines, however, the meeting must take place as soon as possible. It is easier to find a common time for a smaller group than for a whole review team.

Reviewers ought, in any case, to write issues down before the meeting as this helps the scribe, and as the issues themselves may be very complicated. In this way they can be sent to the author by electronic mail to be collected and brought to the meeting. Thus only the most important and technically competent employees need to participate in the meeting, check all the issues and decide what to do.

The limited logging meeting is a specialization of the virtual logging meeting in the same way as the no logging meeting is, but in this case the common notice board is replaced by a smaller group of reviewers. Alongside the virtual logging meeting, the limited logging meeting can also be a specialization of the pair review. When only a couple of people – the author and another person (e.g. the chief designer) – check the issues together, they can work iteratively focusing more on issues than on the document to be reviewed.

5. Conclusion

For a number of reasons, each company normally has its own way of tailoring the review process. These tailored versions differ in type and in grade of formality. We have introduced here some easy ways to improve the review process, presenting steps which are simple enough for the management and the staff to accept and efficient enough to improve the review process.

Our proposals are divided into two groups. The first includes various aspects of the review process. The software development staff should have a proper knowledge of the formal review process even if they follow their own version of it. It would be reasonable to focus on the interfaces of modules, and the review process should be ranked in the first category of activities. In addition, a shared kick-off meeting is justified in some cases.

In the second group of proposals we focus on new types of review. One should have the appropriate version for one's own purposes. Even the review meeting may not be necessary if the software development process is familiar enough to the staff, and the pure, formal review method may simply not fit with certain kinds of software process. A *pair review* consists of only two persons, who are very familiar with the material being reviewed. A *limited logging* meeting gathers a limited group together from among the total set of reviewers. It is not necessary for every reviewer to come to a logging meeting since a smaller group is able to check the issues. Besides, it may be hard to get all reviewers to come to a meeting because of timetables. Consequently the logging meeting can be held in a *virtual* form, with the reviewers using a communication channel such as an intranet. When there is no need for group acceptance of errors and the author can him/herself checks the status of the issues, the logging meeting becomes *unnecessary.*

In contrast to the statement of Humphrey [14] that code reviews are more efficient in error searching than code testing, we propose black-box reviews, where one reviews the test plan and not the code. It would be worthwhile checking this opportunity.

Most software companies apparently still locate at the first level in the CMM hierarchy, and may find it too difficult to implement the formal review method since they have difficulties in formalizing their software development process in general. Nevertheless, one can be sure that no company will forsake reviews once it has had good experiences with **efficient** ones. It is for this reason that the companies should implement reviews with care.

References

1. M. E. Fagan. Design and code inspections to reduce errors in program development, *IBM Systems Journal,* **15** (1976) 182–211.
2. T. Gilb and D. Graham. *Software Inspection* (Addison-Wesley, Reading, MA, 1993).
3. E. F. Weller. Lessons from three years of inspection data, *IEEE Software,* **10** (1993) 38–45.
4. S. H. Kan. *Metrics and Models in Software Quality Engineering* (Addison-Wesley, Reading, MA, 1995).
5. A. F. Ackerman, L. Buchwald and F. H. Lewski. Software inspections: an effective verification process, *IEEE Software,* **6** (1989) 31–36.
6. R. G. Ebenau and S. H. Strauss. *Software Inspection Process* (McGraw-Hill, New York, 1994).
7. R. B. Grady. *Practical Software Metrics for Project Management and Process Improvement* (Prentice-Hall, Englewood Cliffs, NJ, 1993).
8. A. A. Porter, L. G. Votta and V. R. Basili. Comparing detection methods for software requirements inspections: a replicated experiment. *IEEE Transactions on Software Engineering*, **21** (1995) 563–575.
9. C. Jones. *Applied Software Measurement* (McGraw-Hill, New York, 1991).
10. R. L. Mack and J. Nielsen. Executive summary, in J. Nielsen and R.L. Mack (eds) *Usability Inspection Methods,* (John Wiley & Sons, 1994).
11. I. Tervonen and H. Oinas-Kukkonen. Reorganizing the inspection process: problems encountered and resolved. *Software Process – Improvement and Practice*, **2** (1996) 97–110.
12. J. Borrajo Iniesta, A tool and a set of metrics to support technical reviews, in M. Ross, C.A. Brebbia, G. Staples and J. Stapleton (eds) *Software Quality Management II* Vol 2: *Building Quality into Software* Computational Mechanics Publication, Southampton, (1994) pp. 579–594.
13. L. G. Votta. Does every inspection need a meeting? *Proceedings of 1st ACM SIGSOFT Symposium of Foundations of Software Engineering.* Published in *ACM Software Engineering Notes,* **18** (1993) 107–114.
14. W. S. Humphrey. *A Discipline for Software Engineering* (Addison-Wesley, Reading, MA, 1995).

Juha Iisakka (MsG) is a teacher of software engineering at the University of Oulu. His current research interests include software quality and review as well as advanced software solutions.

Ilkka Tervonen is a professor (acting) of software engineering at the University of Oulu. He received a PhD in software engineering from the University of Oulu. His current research interests include software quality, software inspection, object oriented approach, and organizational memory. He has written numerous articles on software quality and software inspection.