

Supporting a software inspector throughout inspection

I. Tervonen

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

Abstract

It is largely accepted that 65 to 90 % of operational defects are detected by inspections. Individual inspection is the most effective type, the others are public inspection and group inspection. These three types of inspections determine the support required to reach economically the maximum score of defect-detection. In this paper we first introduce the three types of inspections, then consider appropriate support for them, illustrate the support provided by a CASE tool, and review related work.

1 Introduction

Inspections and reviews are typically used as quality assurance techniques early in the development process, well before formal artifacts such as source code. The full potential of a review is rarely achieved in any of its current forms. Johnson and Tjahjono [5] present three significant roadblocks to fully effective review: (1) labour intensity (an entire man-year of effort is needed for a team of four reviewers to review a 20 KLOC program, Russell [10]), (2) better compatibility with incremental development methods (reviews are deployed as "hurdles" to be jumped a small number of times at strategic points during development), and (3) prescriptivity (e.g. a meeting must last a maximum of 2 hours, each line of code must be paraphrased, lines of code must be read at rate of 150 lines per hour. Strict prescription nevertheless appears to suggest that organizations must adapt to a review method, rather than that the review method must adapt to the needs and characteristics of organizations).

In this paper we focus on the above shortcomings, i.e. how to make inspections more economical, how to tie inspections to the development

196 Software Quality Management

method, and how to make inspections more tailorable. We define the inspection process with three types of inspections, individual inspection, public inspection and group inspection. Thus the economical aspect means that the individual inspectors should easily understand the software descriptions (baselines) to be inspected, the participants of public inspection must flexibly communicate with each others, and the participants of group inspection require summary reports of open issues to be considered in the meeting. The use of equal CASE tool in development and inspection makes the process more consistent, and the use of tailored rules and checklists makes the inspections more adaptable.

In this paper we first introduce three types of inspection, then consider appropriate support for them, illustrate the support provided by a CASE tool, and finally review related work.

2 Inspection and its evaluation

Gilb and Graham [3] following Fagan [2] define an inspection with steps such as entry, planning, kickoff meeting, individual checking, logging meeting, edit, follow up, exit and release. It is largely accepted that inspection is an inseparable part of the development process when each baseline is inspected. This is depicted by means of the inner circle in Figure 1. The cycle of development-inspection can be terminated either in the redevelopment stage (rework does not require inspection) or in the inspection evaluation stage (inspection does not generate rework).

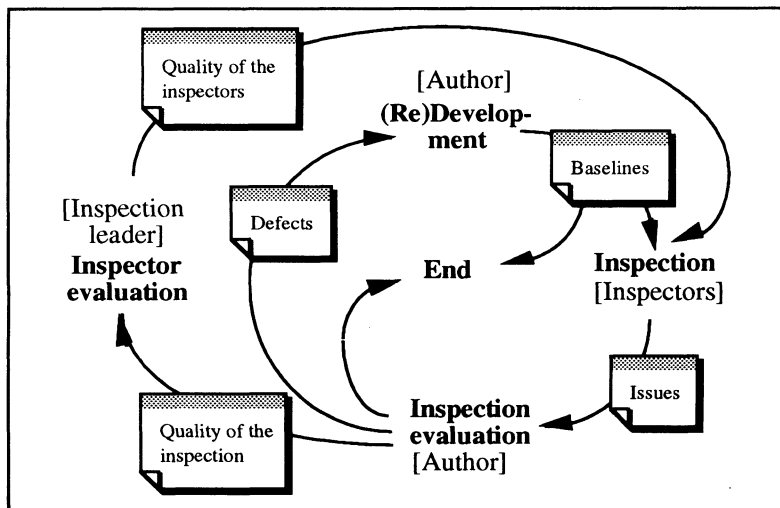


Figure 1: Inspection and its evaluation

As depicted in Figure 1, we can find also another circle (the outer circle) which is focused on the evaluation of the inspection process and further on the

evaluation of the inspectors. This circle is an improvement cycle and makes an inspector more effective to detect defects.

3 Three types of inspection

In this paper we focus on two of the inspection stages [3], namely individual checking and logging meeting (cf. Figure 2). We enhance the traditional inspection process with the collaborative inspection approach so that we replace a part of the face-to-face meeting with on-line meetings. This decreases the mutual time required and makes the conciliation of participants' timetables easier. We now define the inspection process with three types of inspections, individual inspection, public inspection and group inspection

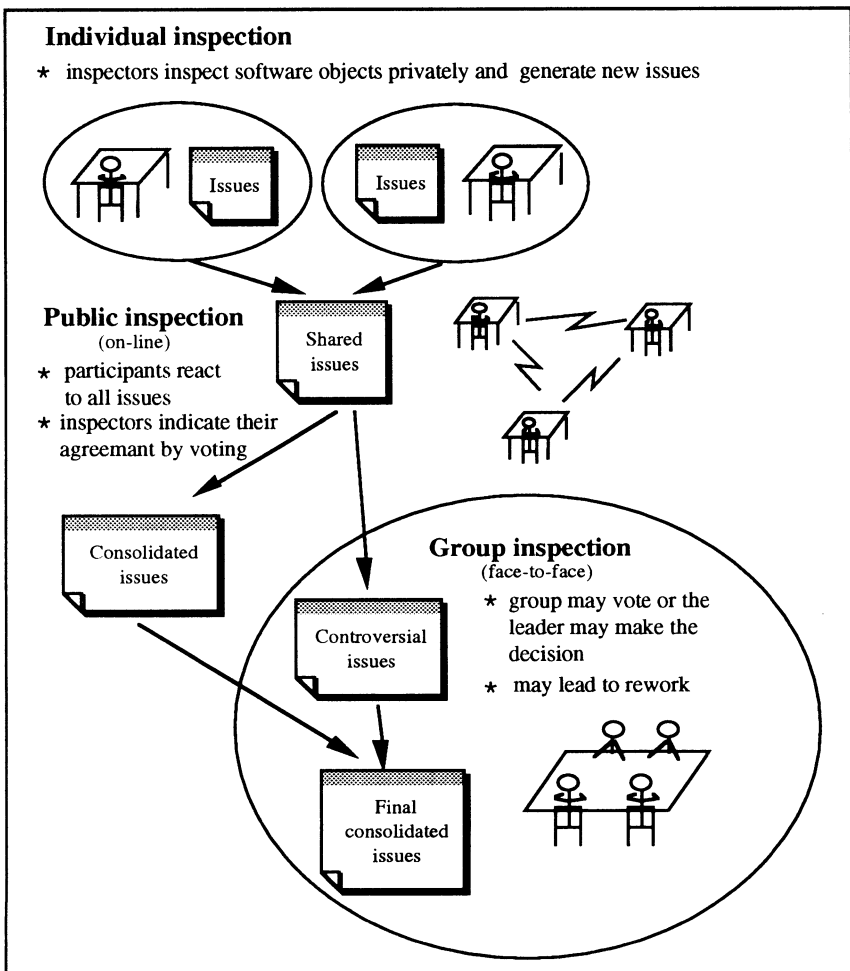


Figure 2: Three types of inspection

198 Software Quality Management

Typically, individual checkers (inspectors) work alone on the product document (baseline) using the source documents, and the rules, procedures and checklists provided. The time spent in individual checking is a key and critical part of the inspection process. Without individual checking [3], you will end up with merely a group review which will probably only find 10% of the defects which could have been found by applying inspection rigorously. In order to make the individual checking effective, roles are assigned, with associated checklists, which help to focus the individual checking process in different directions for each individual checker. The inspection leader monitors the individual inspection process, and if new issues arise, the process proceeds to a public inspection (on-line work), in which the participants react to all the issues and the inspectors can also create new issues and links between them. Finally, the inspectors indicate their agreement about the existing actions by voting. Then, the inspection leader creates a consolidated representation of the state of the inspection, and if controversy is evident a group meeting may be called. This entails a group inspection (face-to-face meeting) in which the group may vote about unresolved issues which can lead to reworking, or else the inspection leader may decide on reworking or simply conclude the inspection.

4 Supporting an inspector

We derived the demands for support from three shortcomings in present software inspections. We now discuss two of these, namely the economical and tailorability aspects. The economical aspect means that the individual inspectors should easily understand the software descriptions (baselines) to be inspected, the participants of public inspection must flexibly communicate with each other, and the participants of group inspection require summary reports of open issues to be considered in the meeting. The tailorability aspect is based on the use of tailored rules and checklists which make the inspections more adaptable.

4.1 Economical aspect in individual inspection

There are some demanding characteristics in inspections, which further delineate the support inspectors require. Inspection is included into the software development process, which causes that inspectors should understand the design method, notation and representation forms of the design environment used by a software designer. The connections between a designer and inspectors are illustrated in Figure 3. The common background is based on the rules and the checklists derived from them. In an inspection situation we emphasize the recorded design rationale as an explanation of an alternative description and design decision.

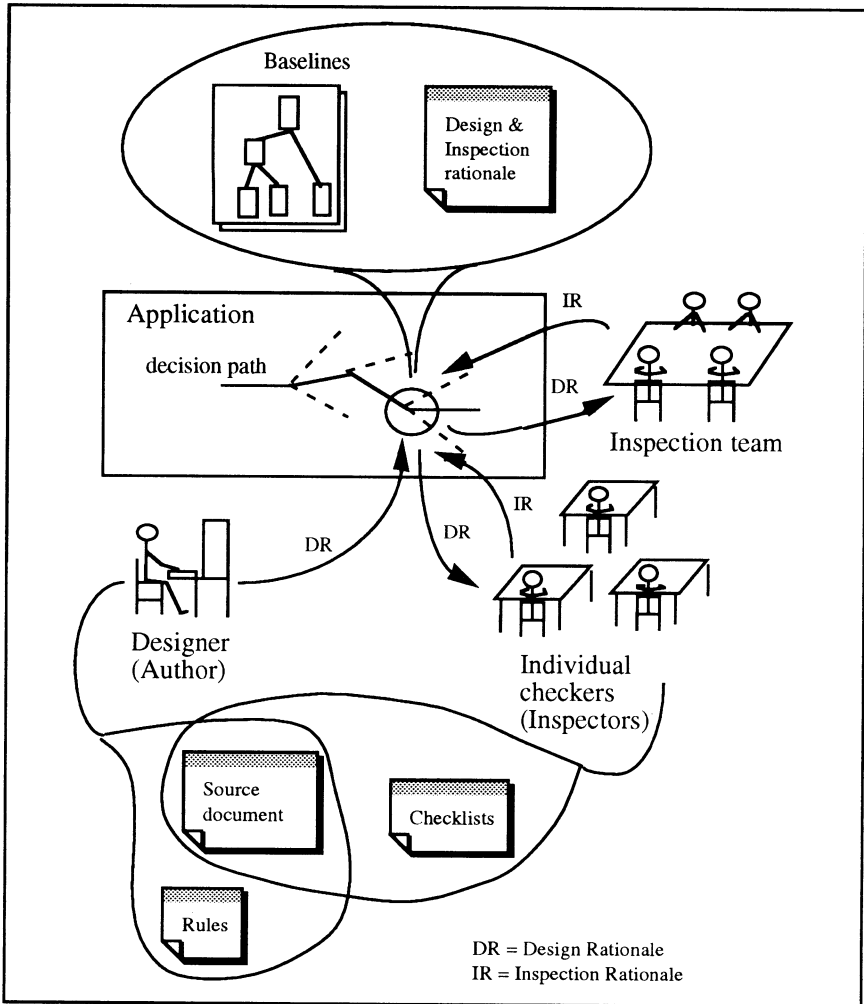


Figure 3: Sharing information between designers and inspectors

The main document which is being inspected is referred to as the product document (baseline of an application) which could be a design, a requirements specification, a code listing, a test plan, and so on. The source document is the primary document which the product document is compared with. In addition, every work process has some rules and checklists. Rules help increase the objectivity of the inspection process, they are directives for authors, but in the hand of checkers they are only used as if they were checklists. The only real purpose of checklists is to permit and encourage the interpretation of rules which would probably not be made by some checkers, but which will lead to uncovering major issues.

200 Software Quality Management

The sharing of information between designers and inspectors is implemented by means of design and inspection rationale, as depicted in Figure 3. The designer produces design rationale in software design phases. Inspectors must be familiar with the used development method and environment to understand alternative descriptions, design decisions and justifications. The designer and the inspectors have in large a similar background for working. The designer uses source document and rules (e.g. method dependent) to produce the product document (baseline), whereas the inspectors evaluate the baseline from source document and checklist viewpoint. Both the individual inspectors and the inspection team can produce new justification information (inspection rationale).

4.2 Tailorability aspect

Rules and checklists form a common background for the participants in software development. Checklists are needed for the checking process alone, and are not normally used in the production of the product document. A checklist is a tool for instructing the checkers what to search for, describing clever defect search techniques and encouraging continuous improvement by storing good checklist questions for future use [3].

In addition to the presented memory aid, rules and checklists can also be used as a store of accumulated knowledge. We can organize their management using the quality factor, criterion, metric structure of SQM approach [9], as illustrated in Figure 4.

We have modified the base structure of the SQM approach so that the goal corresponds to the factor and the rule to the criterion. These items serve the designer's work, whereas the checklists and metrics are for inspectors. As depicted in Figure 4, Expandability is refined with Readability and Upgradeability criteria which form the rules for the designer. In object-oriented tailoring of rules the Readability means "Aiming at understandable class structures, method interfaces and methods, for example. The inspectors evaluate e.g. the superclass-subclass structure of the baseline and can use the NMO (number of methods overridden), NMI (number of methods inherited) and NMA (number of methods added) metrics [7], for example. We can evaluate the superclass-subclass structure with these metrics so that, a large number of overridden methods indicates a design problem, the number of methods inherited from superclass indicates the strength of the subclassing by specialization, and subclasses should define (add) new methods extending the behaviour of the superclass.

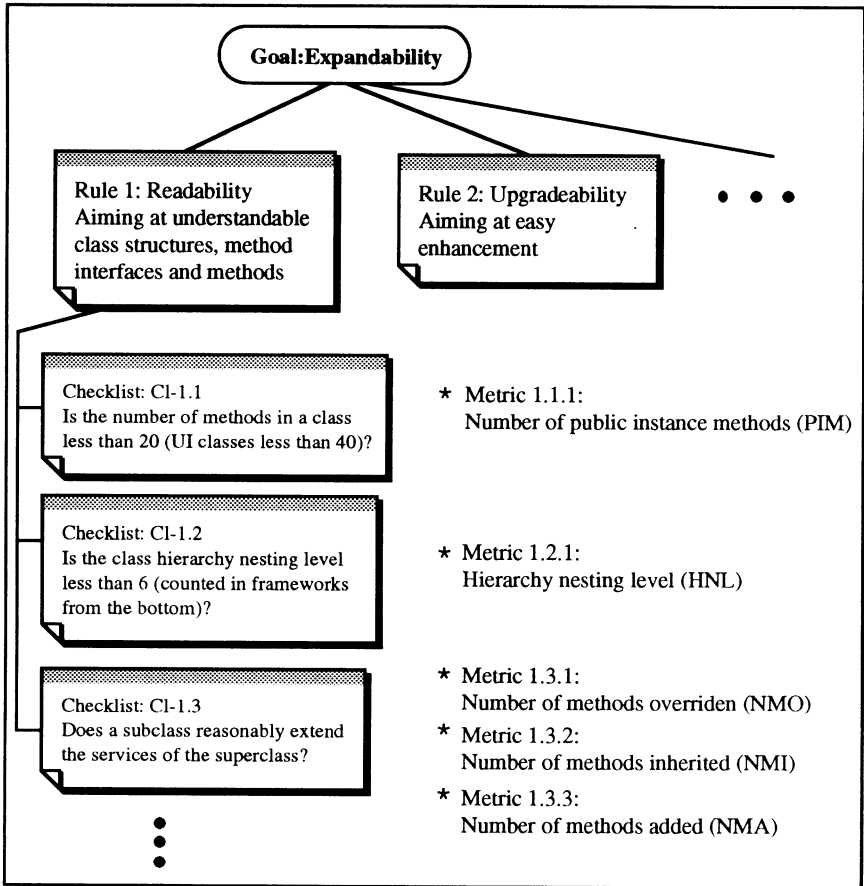


Figure 4: A goal-rule/checklist-metric structure

4.3 Economical aspect in public inspection

In public inspection inspectors require support for flexible communication between participants. This is an area of innovative research in the computer supported cooperative work (CSCW) which asserts that we need to provide computer support for work processes involving more than a single individual. The CSCW technology is classified into four categories [12]: communication mechanism, shared workspace facilities, shared information facilities and group activity support facilities. These principles are useful in public inspection too. The communication mechanism enables inspectors at different locations to see, hear and send messages to each other. The shared workspace facilities enable inspectors to view and work on the same electronic space at the same time. The shared information facilities enable inspectors to view and work on a shared set of information, while the group activity support facilities augment inspection processes.

202 Software Quality Management

When approaching interpersonal computing, we have walked through the old technology of messaging and shared documents which are adequate for purposes such as passing a phone message or scheduling an inspection meeting [1]. The next stage was the groupware which supported richer and more complex social reality. It was based on the "conversational model" which grasped the central nature of conversations in organizations. Restriction in the complexity and naturalness of a given conversational model caused shift to graphical hypertext systems to capture the full richness and depth of the team's interactions on the problem [1]. The new technique supports the creating of an organizational memory based on recorded design and inspection rationales. This technique follows the issue based approach of Rittel [6] in which a particular issue may have various positions, which have certain arguments to justify them.

4.4 Economical aspect in group inspection

In group inspection (face-to-face meeting) the participants demand summary reports of controversial issues to be considered in the meeting. The evaluation of inspection process and further the evaluation of inspectors should be considered in a team, but it could be organized as a brainstorm meeting after a specific group inspection.

5 A prototype of a supporting CASE tool

We recognized the problem of tying inspection flexibly to the development method earlier in this paper. Our solution to this problem is based on the use of design and inspection rationale (cf. Figure 3) and a supporting CASE tool. We experiment the use of design and inspection rationale in the context of the quality-driven assessment method [11]. The method is appropriate for small scale development and Smalltalk development in particular. Of course the inspectors must understand the idea of the method to understand the material to be inspected. The inspectors can also enhance the inspected descriptions with their comments presented in terms of quality issues. Due to object-oriented tailoring, the development history of a software product forms a class hierarchy.

The "QDA tool" prototype is based on the active use of quality factors and criteria (SQM model, cf. [9]) in assessing of design decisions and the recording of design rationales as extension of software descriptions. The major supporting functions are implemented with separate windows (cf. [11]): (1) a Quality window for presenting definitions of quality factors and criteria in general and interrelationships between them, (2) a Featuring window for prioritizing the major quality factors, (3) a Tailoring window for presenting tailored (e.g. object-oriented) quality factors and criteria, (4) a Navigation window, which presents an AND/OR tree as a development history of the software, and (5) a Description window, which is the graphical editor by

which the real alternative software descriptions are produced, and which also supports the extension of descriptions with design rationale (cf. Figure 5).

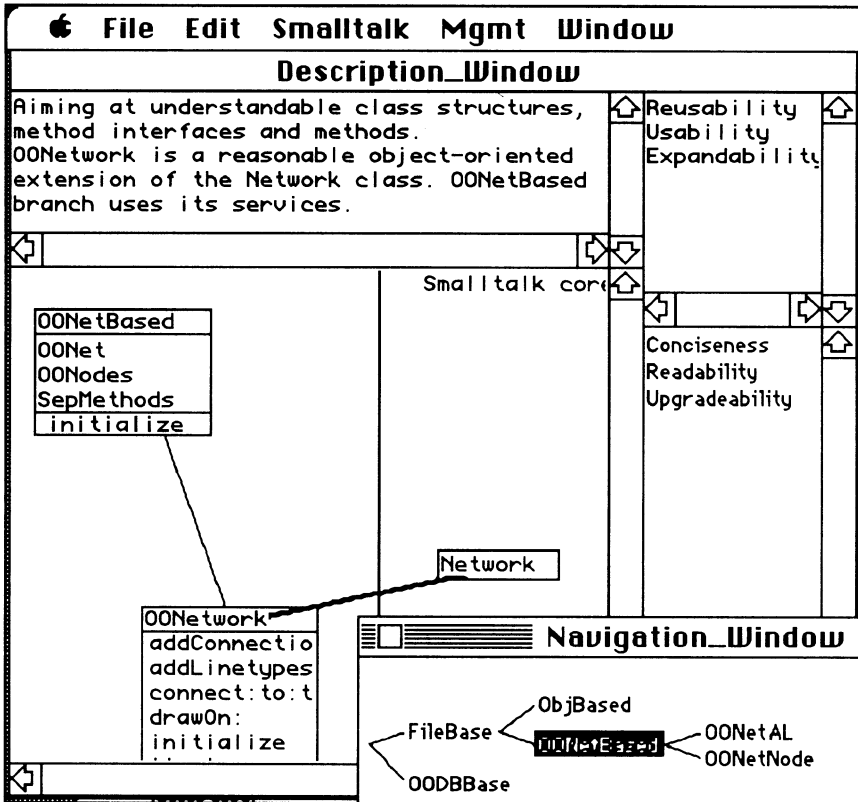


Figure 5: The Description Window

The Navigation and Description windows are the major windows from quality assessment and justification viewpoints. In this excerpt we consider the designer's work, i.e. his justifications considering a specific alternative. We use the same goal-rule structure depicted in Figure 4.

In this example we use the rule the designer's own opinion, as a design rationale to characterize the "value" of an alternative. The goals are set using the quality factors Reusability, Usability and Expandability. The Expandability can be reached using the rules (in terms of Conciseness, Readability and Upgradeability). The readability means "Aiming at understandable class structures, method interfaces and methods". The designer has enhanced this design rationale by his opinions of the alternative, i.e. "OONetwork is a reasonable object-oriented extension of the Network class" and "OONetBased branch uses its services".

In the inspection phase the inspector walks through the checklists (e.g. those in Figure 4) and recognizes that "OONetwork" fulfils the checklist 1.3,

204 Software Quality Management

i.e. "Does a subclass reasonably extend the services of the superclass?" Inspectors can in equal way enhance the justifications by their inspection rationales. In individual checking phase these rationales are suggestions of issues and improvements for further public and group inspections. In public and group inspections the inspection rationales are recorded to present the issues and improvements for the author.

6 Related work

A collaborative inspection is a rather new approach in software quality engineering which causes that the supporting environments are rare. We introduce three environments which support a part of the characteristics linked to three types of inspection. Two of them, CSRS (Collaborative Software Review System, [5]) and TRST (Technical Review Support Tool, [4]), provide support for individual inspection, recording of issues, evaluation of the inspection and evaluation of the inspectors. The third tool, CM/1 [1], is a potential supporting environment for public inspection, although not yet applied to this area.

In "CSRS" [5], each program object, such as a function, procedure, macro, variable or data type declaration, is retrieved from the source code and placed at a node of its own in a hypertext-style database. The data model of the "CSRS" comprises nodes such as source node, issue node, action node, comment node, evidence node, consolidated issue node and consolidated action node. The review participants first individually review the source code and document any suspected problems or defects in issue nodes. Any question concerning the source is also recorded in comment nodes. The action nodes represent solutions to the problems in issue nodes. Because private review leads to redundant issues, they must be summarized into a single consolidated issue, and related actions into a consolidated action. Actual reworking activities are based on a single consolidated action. The benefits of "CSRS" can be summarized under three points: (1) it results in a richly linked repository for all review artifacts and thus facilitates reworking, project scheduling and access to design/maintenance rationales, (2) it reduces dependence on traditional review methods as used in same-time, same-place group work, and (3) it provides automated support for the roles of producer, review leader and reviewer.

The "TRST" [4] is a review management tool which supports (1) the collecting improvement proposals from single reviewers, (2) the organising and keeping track of reviews of many documents, (3) the controlling of changes made on a document as a result of a technical review, (4) the measuring of the quality of the review itself, and (5) the measuring of the quality of the document. Author's feedback on the improvement proposals is one of the advanced characteristics of the "TRST". This means that the authors can say whether they find each improvement proposal very interesting, normal, spelling (do not contribute to the information content), of little interest,

or not interesting at all. This evaluation allows the review leader to analyse the quality of reviews and the competence of reviewers. Both the improvement proposals and its evaluation form potential source material for justification.

The "CM/1" [1] uses a graphical hypertext system to represent the richness and complexity of human conversation. It is specially intended for teamwork and team communication - as opposed to the typical enterprise-wide nature of e-mail or Notes-style document sharing. This groupwork based approach requires someone to act as a moderator or facilitator for the electronic meeting. The role of the moderator is partly administrative, but it also includes keeping the group focused on making progress on its project.

All in all, the design rationale approach in the context of graphical editors is rare. This has been justified by its conceptual and technical complexity, i.e. it is difficult to manage all justification information, and it is difficult to produce flexible implementations and usable user interfaces. Anyway, this work is progressing and we know at least one initiative to enhance a commercial graphical editor (in this case a meta editor) with a design rationale tool (cf. [8]).

7 Conclusions

As the primary contribution the paper suggests proposals to three shortcomings, i.e. how to make inspections more economical, how to tie inspections to the development method, and how to make inspections more tailorable. We define the inspection process with three types of inspections, individual inspection, public inspection and group inspection. Thus the economical aspect means that the individual inspectors should easily understand the software descriptions (baselines) to be inspected, the participants of public inspection must flexibly communicate with each other, whereas the participants of group inspection require summary reports of open issues to be considered in the meeting. The use of equal CASE tool in development and inspection makes the process more consistent, and the use of tailored rules and checklists makes the inspections more adaptable. We have illustrated the use of design and inspection rationale with a supporting tool ("QDA tool") for quality-driven assessment. Finally, we reviewed related work and find that the design rationale approach in the context of graphical editors is rare. Probably the reason to this is its conceptual and technical complexity, i.e. it is difficult to manage all justification information, and it is difficult to produce flexible implementations and usable user interfaces.

References

1. Conclin J., and Yourdon E., Groupware for The New Organization, *American Programmer*, vol 6, no 9, 1993, pp. 3-8
2. Fagan M.E., Design and Code Inspections to Reduce Errors in Program Development, *IBM Systems Journal*, vol 15, no 3, 1976, pp. 182-211



206 Software Quality Management

3. Gilb T., and Graham D., *Software Inspection*, Addison-Wesley, Wokingham, England, 1993
4. Iniesta J.B., A Tool and A Set of Metrics to Support Technical Reviews, in Ross M., Brebbia C.A., Staples G., and Stapleton J.(eds.), *Software Quality Management II vol.2: Building Quality into Software*, Computational Mechanics Publication, Glasgow, Scotland, 1994, pp. 579-594
5. Johnson P.M., and Tjahjono D., Improving Software Quality through Computer Supported Collaborative Review, in (Ed. De Michelis G., Simone C., and Schmidt K.), *Proceedings of the Third European Conference on Computer Supported Cooperative Work*, Milan, Italy, 1993, pp. 61-76
6. Kunz W., and Rittel H.W.J., *Issues as elements of information systems*, Working Paper 131, Center for Planning and Development Research, University of California, 1970
7. Lorenz M., and Kidd J., *Object-oriented Software Metrics*, Prentice Hall, Englewood Cliffs, NJ, 1994
8. Lyytinen K., Kerola P., Kaipala J., Kelly S., Lehto J., Liu H., Marttiin P., Oinas-Kukkonen H., Pirhonen J., Rossi M., Smolander K., Tahvanainen V-P., and Tolvanen J-P., *MetaPHOR: Metamodeling, Principles, Hypertext, Objects and Repositories*, Technical Report TR-7, Department of Computer Science and Information Systems, Jyväskylä, 1994
9. McCall J.A, Richards P.K, and Walters G.F., *Factors in Software Quality*, Volumes I, II, and III, RADC reports, 1977
10. Russell G.W., Experience with inspection in ultralarge-scale developments, *IEEE Software*, vol 8, no 1, 1991, pp. 25-31
11. Tervonen I., *Quality-driven assessment: a pre-review method for object-oriented software development*, Dissertation thesis, University of Oulu, Department of Information Processing Science, Research Papers, Series A19, 1994
12. Wilson P., Computer supported cooperative work: an overview, *Intelligent Tutoring Media*, vol 1, no 3, 1990, pp. 103-116