MONITORING SOFTWARE INSPECTIONS WITH PRESCRIPTIVE METRICS

Ilkka Tervonen and Juha Iisakka, University of Oulu, Finland

Summary:
It is largely accepted that inspection is the most effective means of finding defects, and thus plays an important role in the software development process. This paper therefore focuses on inspection, its evaluation and improvement, and presents a set of prescriptive software metrics for evaluating software inspections. The usability of the metrics is further illustrated with a case study involving a Finnish software company.

Ilkka Tervonen, and Juha Iisakka, Department of Information Processing Science, University of Oulu, FIN-90570, Oulu, Finland, Fax: +358 81 553 1890, E-mail: {tervo, isac}@rieska.oulu.fi

1. Introduction

Inspections are typically used as quality assurance techniques early in the development process, well before formal artifacts such as source code. The full potential of an inspection is rarely achieved in any of its current forms, however, and it requires further improvements. We recognize problems e.g. in labour intensivity, compatibility with incremental development methods and tailorability (Should we adapt the organization to the inspection method or the inspection method to the needs of the organization?). Researchers have looked for solutions to these problems by means of individual inspections, shared inspections and phased inspections (Gilb and Graham, 1993), (Johnson, 1993), (Knight and Myers, 1993), but still there are some aspects of inspection that have to be addressed e.g. the size of the material, the rate of inspection and the productivity of inspection. Our experiment in industry reported here pointed to major shortcomings in the inspection process, including excessively large amounts of material to be inspected, lack of any organized training for inspection, inadequate preparation time and intensity, inspection meetings that are not formal enough, checklists that are not used rigorously, and delays in checking after the author's corrections have been made.

This paper focuses on the evaluation aspect of inspections. A successful evaluation inevitably requires a set of metrics, any evaluation is a necessary predecessor of improvement (you can't improve something if you don't know the current situation). As improvement of the development process of a company is the major goal of our research, we need prescriptive metrics which set objectives and guide the improving of the inspection process. The major contribution of this paper is a set of prescriptive metrics by means of which we can improve inspections and further the development process. We first introduce software inspection and its role in software development, and then as a synthesis

of reported experiments, we define a framework for evaluating inspections. Finally we illustrate the usability of the framework to evaluate inspections in a case study.

2. Software inspection

It is largely accepted that inspection is an inseparable part of the development process, in which each artifact has to be inspected (as depicted in Figure 1). Inspection is traditionally defined with steps such as entry, planning, kickoff meeting, individual inspection, logging (inspection) meeting, edit, follow up, exit and release (Fagan, 1976, Gilb and Graham, 1993) .

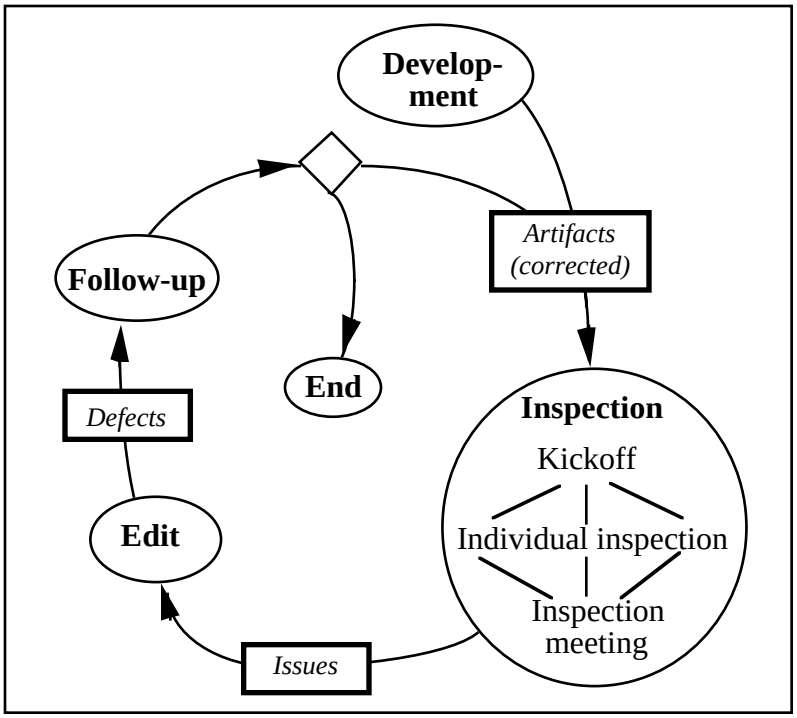

Figure 1: The core circle of software production

Figure 1 focuses on major phases such as kickoff meeting, individual inspection, inspection meeting, edit and follow-up. Our presentation follows the definitions of Gilb and Graham (1993). A kickoff meeting is held to ensure that the inspectors know what is expected of them, and it may include the distribution of documents, role assignments and training in inspection procedures, rules and checklists. In individual inspection the participants work alone on the material using the source document (material from earlier phases), and the procedures, rules and checklists provided. The role assignments help to focus the checking process on different directions for each individual inspector. The aim is to find the maximum number of unique major issues. Issues, potential questions and suggestions for improvement that have been identified earlier are recorded in an inspection meeting, a second purpose at which is to discover more major issues. In the edit phase the author evaluates the record of issues and classifies them as defects (need correction) or non-defects (require just a comment). Following the tradition of inspection, the difference between an issue and a defect terms is related to acceptance by the author, i.e. only accepted defects are recorded as such. In follow-up phase the inspection leader checks that

satisfactory editor action has been taken on all defects, e.g. a request for change made to the owner of the document or a correction made by the author. After that the exit status is evaluated based on specific exit criteria, i.e. the follow-up must be complete, checking rates must be within acceptable limits, and the number of errors should be below specified limits.

3. Prescriptive metrics for monitoring inspection

In general, the set of metrics can be categoritized into descriptive and prescriptive types (Fan and Yih, 1994). Descriptive metrics are derived from natural processes and objects. By analyzing the characteristics and behaviour of natural processes and objects, an objective metrics system describing the phenomena and facts can be derived. For example, the physical, electrical, strength and optical metrics of a material can be derived from observations. Prescriptive metrics stand for what the status should be in order to accomplish the goal, and these are derived from descriptive metrics along with relevant information such as knowledge, regulatory codes and standards. Thus prescriptive metrics = descriptive metrics + sound (goal) state information, i.e. prescriptive metrics specify values, ranges and relations of relevant factors as they should be in the sound state. As an example, human body temperature, blood pressure and pulse are prescriptive. Relative to descriptive metrics, prescriptive metrics attach semantics (goals) to their measures and thus add a reference dimension.

3.1 The evaluation and improvement circle

As discussed earlier, inspection is an inseparable part of the development process. Prescriptive metrics are needed to convert software inspection into a more disciplined engineering technique with stable performance, setting objectives for process improvement. The difficulty experienced with control over the quality of the inspection process lies in the fact that inspection is a mental activity and cannot be observed directly. This means that it is difficult using current inspection metrics to detect whether the inspector is really putting his mind to his work and to assess when the inspection process is actually complete (Fan and Yih, 1994).

As a synthesis of the inspection research discussed here, we introduce a set of prescriptive metrics and link it to the development-inspection process, as depicted in Figure 2. The inner circle characterizes the basic process, which is largely equal to the scheme presented in Figure 1. We focus on this inner circle of software quality measured in terms of issues, defects, LOC, complexity and completeness. In the outer improvement circle we gather other data indicative of the inspection quality status and grade of competence, which further drive the improvement of the inspection process and the training of inspectors. Inspection evaluation is viewed here as a summative and coordinative activity which produces an agreed opinion on software quality and a cumulative inspection quality evaluation. All in all, these types of information form the basis for prescriptive metrics, i.e. we can set objectives based on earlier research and compare the project data with them.

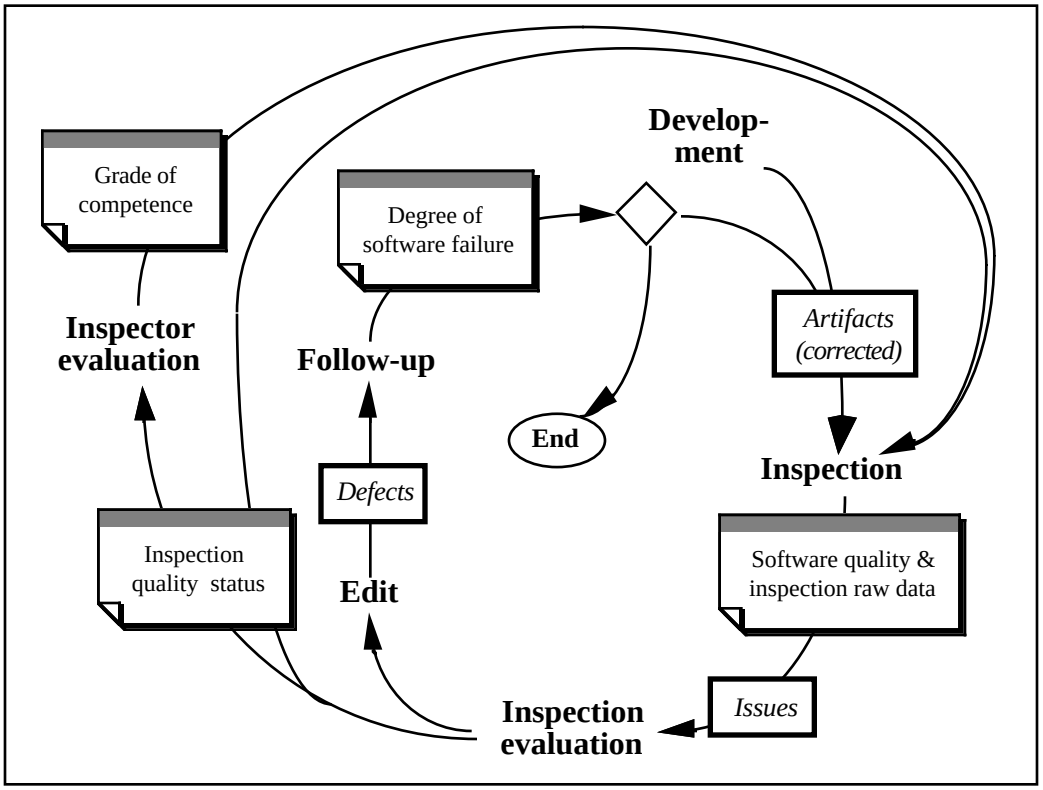

Figure 2: A set of prescriptive metrics linked to software production

3.2 Metrics in inspection

We now introduce the metrics inside each aggregate quality indicator. An outline of the prescriptive metrics and their links to quality indicators is depicted in Figure 3. We shall now explain the prescriptive metrics in more detail. It is necessary to gather software quality and inspection raw data in order to implement the inspection process, and we also try to find and record tentative characteristics of software quality, e.g. inspectors' opinions of software quality and potential suggestions for reinspection. The software quality and inspection data are measured and recorded with five metrics of Barnard and Price (1994) namely (1) the LOC (or LOT[1]) inspected, (2) number of inspectors, (3) preparation time, (4) inspection duration and (5) total faults (issues) detected. If we focus on code inspection in particular, we will also measure (6) complexity of code, measured with Halstead's volume (V) or McCabe's cyclomatic complexity (CC) values.

Degree of software failure provides information to help the inspection leader to determine when the inspection is complete, based on the scheduled timetable, follow-up status, checking rate and estimated number of major defects left, for example. It is important to remember that one in six of all edit changes will be incorrect or incomplete (Gilb and Graham, 1993), and therefore we evaluate completeness in the exit phase by means of the estimated major defects remaining. This metric for the degree of software failure is calculated on the basis of removal effectiveness (e.g. 74%, 61% and 55% for inspection of high level design, low level design and coding phases (Kan, 1995)) and the defect insertion rate (one out of six attempts fails).

[1]LOT = line of text

Inspection quality status is a long-term metric of the company's (or division's) accumulated inspection experience. Where the software quality and inspection raw data explain the status of a specific inspection, the inspection quality status will depict characteristics of the inspection process in the long term. Inspection quality status is a compilation of two types of metrics, a set of average quality metrics and a maturity metric explaining the extent of inspection adoption.

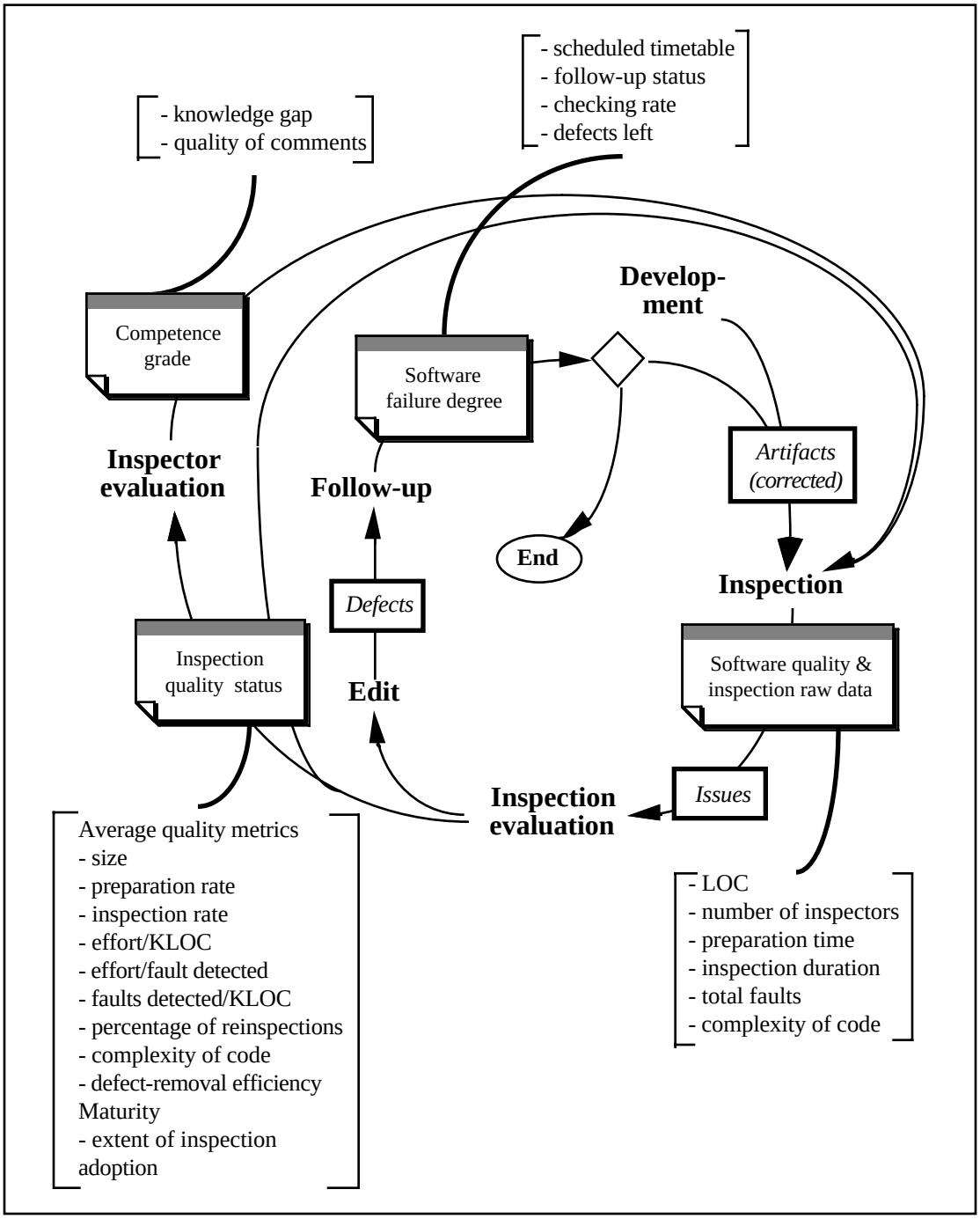

Figure 3. An outline of the prescriptive metrics

Average quality metrics can be characterized in terms of time used, number of the inspectors participating and number of defects detected. Our set is based on the metrics

derived by Barnard and Price (1994), and includes: (1) average size of material inspected, (2) average preparation rate, (3) average inspection rate, (4) average effort per KLOC (or KLOT), (5) average effort per fault detected, (6) average faults detected per KLOC (or KLOT), and (7) percentage of reinspections. For code inspection, we measure (8) average complexity of code, and if total coding faults (i.e. faults detected during inspection, testing and use) are available, we also measure (9) defect-removal efficiency. The equations behind these metrics use the data items introduced earlier with the software quality and inspection raw data.

We can estimate the maturity of inspection in a division in terms of the extent of inspection adoption, and the maturity of inspection in a company as an average of these figures. Maturity of inspection is a fairly new metric. Grady and von Slack (1994) define the equation for the extent of adoption as follows: extent of adoption = inspection process maturity * (percentage of projects using inspections + weighted percentage of documents inspected) * constant. *Inspection process maturity* is a constant from 1 to 14 based on a five-level model in which levels 1 and 2 are informal peer reviews of formal walkthroughs (weights 1 and 3), level 3 is an industry-typical inspection process (weight 10), level 4 is a best-practice inspection process (weight 12), and level 5 links formal inspections to a defect-prevention activity (weight 14). *Percentage of projects using inspections:* The number of inspections in a project should be four or more for inclusion in this percentage. *Weighted percentage of documents inspected:* Different types of defects require different relative costs for fixing them. The equation could be (5.7 * percentage of requirements + 2.5 * percentage of design + 2 * percentage of test + percentage of code)/ 11.2). *Constant* (e.g. 0.05) is used to fit the extent-of-adoption metric into a range of 0 to 100.

Grade of competence explains the skills of the inspector in terms of a knowledge gap or the quality of the inspector's comments, for example. The knowledge gap referred to is the difference between the inspector's and the designer's knowledge, which can be represented in terms of the number of years of experience and the number of cases worked on in the problem domain (Fan and Yih, 1994). A positive knowledge gap is recommended for successful inspection. Iniesta (1994) introduces a more advanced principle based on the author's feedback regarding the improvement proposals (issues). This means that the authors say whether they accept the improvement proposals as representing defects and classify them as very interesting, normal, spelling (do not contribute to the information content), of little interest, or of not interest at all. This evaluation allows the inspection leader to analyse the competence of the inspectors and decide on any training needed for inspectors.

3.3 Suggested values for metrics

According to the characteristics of the prescriptive metrics, we need objectives for the measures concerned. It is natural that the metrics for inspection quality status should be most essential when monitoring the inspection process, and thus we also present a summary of these in Table 1. The suggested values are derived from papers introducing experiences in industry, which are not in general statistically reliable. One should also be wary of variations derived from individual human factors, interruptions and the number of problems encountered. Jones (1991), for example, warns that variations in inspection preparation and meetings can deviate from the estimates by + - 50 percent.

For simplicity, we consider together the metrics 1 and 8 (i.e. average size and average complexity). The size of requirements and design documents may be up to 10 pages per chunk (Grady, 1992), which is about 1/4 - 1/5 of the estimates of Jones (1991). The average complexity of code may be based on the LOC inspected, on the cyclomatic number or on the volume of a module. When using the LOC metric, the suggested size of the chunk (or file) to be inspected should be less than 500 LOC (Barnard and Price, 1994) or less than 300 NCSL[2] (Ebenau and Strauss, 1995). The cyclomatic number of a function should be less than 15, and that of a chunk (or file) less than 100, whereas Halstead's volume value for a function should be less than 100 and that for a chunk (or file) less than 8000 (Testwell, 1995).

average LOC inspected	< 40/500 LOC (function/chunk)
other complexity metrics	CC < 15/100, V < 100/8000, (function/chunk)
average preparation rate	< 150 - 200 LOC
average inspection rate	< 150 - 200 LOC
prep. time / insp. time ratio	1.25 - 1.4
average effort / KLOC	50 hours/KLOC
average effort / fault detected	0.5 - 1 hours/fault
total faults detected / KLOC	40 faults/ KLOC
reinspection or exit	0.25 majors/page
defect-removal efficiency	74 %, 61 %, 55 %
extent-of-adoption	26

Table 1. Suggested values for inspection quality metrics

The average preparation and inspection rate is largely accepted to be less than 150-200 LOC/Hr for code and less than 250-400 LOT/Hr for design documents, although the ratio of preparation time to inspection time is thought to vary between 1.25 to 1.4 (Grady, 1992). Jones (1991) gives much greater suggestions for preparation and the inspection meeting, and based on these the ratio of preparation time to inspection time would be below 0. We think that these values are typical of the software industry before emphasis was placed on inspection and optimum rates. His suggestions are 25 pages/h for prepapration and 12 pages/h for meeting in the case of requirements, 45 pages/h and 15 pages/h for functional specification, 50 pages/h and 20 pages/h for logic specification, 150 LOC/h and 75 LOC/h for source code and 35 pages/h and 20 pages/h for user documents.

The defect-removal efficiency estimates are between 50-90 % (Gilb and Graham, 1993), or 74 % for HLD, 61 % for LLD and 55 % for code (Kan, 1995). For total inspection effort, Barnard and Price (1994) suggest that 50 hours should be spent per KLOC as an average effort unless data are available from previous projects. In an experiment they also estimate the productivity (average effort / fault detected) of a sample project to be 0.5 hours per fault, and the total faults detected / KLOC to be 106. Ackerman et al. (1989) and Russell (1991) estimate the productivity (i.e. fix time) to be about 1 hour per fault and the total faults detected decrease correspondingly to 33-37 per KLOC. Iniesta (1994) gives an average value of 20 for improvement proposals (issues) raised per

[2]NCSL = noncommentary source lines

document. The size of the document was not mentioned, but as the average time of review was 2 hours, we can imagine that it was about 400-500 LOC. This means that the total faults detected / KLOC will be about 40-50. Based on these examples we estimate the predictive value of productivity at 40 faults/KLOC, which is quite close to the estimate of 50 total defects/KNCLS of Ebenau and Strauss (1995).

The percentage of reinspections is based on the formula that we need reinspection if more than 90 faults are detected in a single inspection or if the average number of faults detected per KLOC for that inspection is more than 90 (Barnard and Price, 1994). In one example they found 11% reinspections. As this is our only example, we prefer to use as a prescriptive value the guideline for the exit step. Gilb and Graham (1993) give a guideline for reinspection/exit decisions, according to which an exit is suggested if there are less than 0.25 major defects per page remaining (or two to three for beginners). A necessary prerequisite for this decision is that the checking rate did not exceed the optimum rate (150-200 LOC/hour) by more than 20% on average, as faster checking causes a disturbance in our formula. The minimum maturity level of inspection is based on an example of Grady and van Slack (1994), who in their experiment find an increase from 20 percent adoption in 1992 to 26.2 percent adoption in 1993. Thus, we use the value 26 as an acceptable limit for the extent-of-adoption metric.

With software quality and inspection raw data we set objectives for the number of inspectors (between 4 to 6 software engineers, Grady 1992), and for inspection meeting duration (max. 2 hours, Gilb and Graham, 1993). The two hour limit also guides the chunking of the material and thereby the suggested time for preparation.

When evaluating the grade of competence, we may use two suggested values for evaluating inspectors (Iniesta, 1994). The proposals (issues) accepted indicate the general status of the inspectors (in an experiment, 63 % of the proposals were accepted by the author), whereas the interest indicator gives more accurate information, allowing evaluation of an individual inspector. An average value 3.5 of 4 (max) indicates a professional inspector.

4. An example of the framework

The above metrics have been experimented with in an evaluation of the inspection process in a division of a Finnish telecommunications company. The major purpose was to illustrate their appropriateness in practice.

There are some characteristics of the development process typical of the division in question which have serious impacts on both measurement and the metrics used in the inspection. First, the metrics were not collected and recorded regularly but are estimates based on interviews of employees. Second, we focus here on code inspection and not on specification inspection, for example, which is more of a verification-type technique. Third, a major part of the coding is maintenance-oriented which means that the material to be inspected is much larger than is recommended in theory. Fourth, the inspection process does not follow the definitions given in inspection theory (Gilb and Graham, 1993), but is less formal and includes peer inspections and walkthroughs. Fifth, due to the huge amount of material, the inspections are sometimes "tentative", i.e. they do not even aim to reach the objectives of the inspection process. Sixth, the schedule for a project does not include inspection as an activity proper, which causes problems in motivation and in allocating

voluntary inspectors. We can characterize the inspection shortcomings in the division in more detail as follows:

Software quality and inspection raw data: The inspection process is not formal enough, and therefore we do not gather the necessary data for a further inspection evaluation. The kickoff meetings are not held regularly, but instead the chief designer distributes the material to inspectors. The number of inspectors varies between 2 and 4, and sometimes it has been difficult to find voluntary inspectors. The invitation to an inspection (logging meeting) usually comes in time, but the material to be inspected is sometimes distributed too late. Some versions of the checklists are defined, but they are too general to be usable in practice. A classification of defects is defined, but it is not used rigorously. In the inspection meeting the inspectors discuss and even vote on the software quality and potential suggestions for reinspection.

Degree of software failure: The rework status of the baseline is recorded. All classified defects will be corrected. Checking after the author's corrections have been made sometimes takes place too late. The inspection leader determines when the inspection is complete, based on the scheduled timetable and the follow-up status. The term "accepted with changes" confuses the status of the material inspected, i.e. does it require reinspection or not? Some software engineers complained in the interview that the material may even be accepted with changes too often.

Inspection quality status: Due to maintenance-oriented development, the amount of inspection material is huge. Although the number of lines of code maintained is on average one page, the whole set of material to be inspected may amount to 100 pages of code. The designers mark the lines of code to be focused on in inspections. A common practice in code inspection is to focus on module interfaces such as the number and type of parameters and on shared computer memory and to skip checking of the correctness of the algorithms. The preparation time varies from none to four hours, and the inspectors who are responsible for the code (i.e. the chief designers and programmers in a project) can spend as much as four hours on preparation. The logging (inspection) meetings do not always remain within the two hour limit, and their work does not always focus only on discovering "items" or "issues", but includes discussions and even suggestions for correction. A short time for discussing corrections has been recognized as useful in practice. Inspectors may suggest the use of library functions or their own code. The discussion also helps to find a consensus regarding the types and seriousnesses of the defects, and further supports accurracy of recording and correction. Most metrics for inspection quality status have lower values than in Table 1, i.e. average effort / KLOC is less than 50 hours, average effort / fault detected is less than 0.5 hours, and total faults detected /KLOC is less than 40 (the division prefers testing to inspection in the coding phase, for example). The limit value for exit is not evaluated, nor is the defect removal efficiency. The estimated value for the extent of adoption is 63 (i.e. much better than the suggested value). The evaluation is based on the following components: inspection process maturity weight = 7 (almost level 3, shortages in stated objectives, trained practitioners and metrics), percentage of projects using inspections = 100 (inspection is used in all projects), weighted percentage of documents inspected = 80 (percentage of inspection in different phases varies from 50 to 90 percent), and the constant = 0.05.

Grade of competence: The allocation of chief designers and programmers to inspection ensures a positive knowledge gap. The inspection meetings are partly used for

training novice software engineers. This raises a threat of a negative knowledge gap, and is one reason for increasing informality in inspections. Sometimes the participation of novices has a contradictory influence, forcing the use of formal inspection and increasing productivity and efficiency. No further evaluation of the issues detected or the inspectors (Iniesta, 1994) is performed.

As these metrics are not collected or recorded regularly, the estimates presented are based on the authors' understanding of the division's software inspection process. Our example demonstrates that the taxonomy presented here outlines and organizes the evaluation of inspections, and provides guidelines of the necessary rigor. Even this qualitative evaluation supports our search for further improvements.

5. Conclusions

The major contribution of this paper is a set of prescriptive metrics by means of which we can evaluate and improve inspections. The usability of these metrics is illustrated in a case study in a division of a Finnish software company.

Prescriptive metrics, as compared with descriptive metrics, attach goals to their measures and thus add a reference dimension. In software production we gather, side by side with software baselines, issues and defects, other data which indicate inspection quality and maturity status, degree of software failure and the competence of the inspectors.

The set of prescriptive metrics presented here can be used both to recognize shortcomings in inspection practice and to search for improvements in inspection. When implementing an improvement, we must act simultaneously to improve the process and train the inspectors. We have recognized two specific topics of improvement, improving the use of checklists and giving up face-to-face inspections. The use of checklists is a problematic area of inspections. If we do not use resources to improve them, they will remain too general and useless. Tailoring of checklists may be based on the inspection practices of the company, on a specific development method, or on cumulative information on the benefits of checklists. One major reason for giving up face-to-face meetings arises from the fact that they waste time and are cumbersome to arrange. The change over to individual, peer and public inspections is thus an understandable alternative, although these cause some new problems, such as insufficient understanding of the material and communication problems.

We recognized some characteristics of the development process typical of the division and company in question, although the majority of the shortcomings are those recognized as threats to inspection, i.e. the size of material to be inspected is too large, training for inspection is not properly organized, the preparation time and intensity is inadequate, inspection meetings are not sufficiently formal, checklists are not used, and the author's corrections are checked too late. Even though inspections are not measured regularly, the presented taxonomy of metrics outlines and organizes the evaluation of inspections in our example.

References

Ackerman A.F., Buchwald L.S., and Lewski F.H., Software Inspections: An Effective Verification Process, IEEE Software, vol 6, no 3, 1989, pp. 31-36

Barnard J., and Price A., Managing Code Inspection Information, IEEE Software, vol 11, no 2, 1994, pp. 59-69

Ebenau R.G., and Strauss S.H., Software Inspection Process, McGraw-Hill, New York, 1994, ISBN 0-07-062166-7

Fagan M.E., Design and Code Inspections to Reduce Errors in Program Development, IBM Systems Journal, vol 15, no 3, 1976, pp. 182-211

Fagan M.E., Advances in Software Inspections, IEEE Transactions on Software Engineering, vol 12, no 7, 1986, pp. 744-751

Fan C-F., and Yih S., Prescriptive Metrics for Software Quality Assurance, Proceedings of the First Asia-Pacific Software Engineering Conference, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 430-438

Gilb T., and Graham D., Software Inspection, Addison-Wesley, Wokingham, England, 1993, ISBN 0-201-63181-4

Grady R.B., Practical Software Metrics for Project Management and Process Improvement, Prentice Hall, Englewood Cliffs, 1992, ISBN 0-13-720384-5

Grady R.B., and van Slack T., Key Lessons In Achieving Widespread Inspection Use, IEEE Software, vol 11, no 4, 1994, pp. 46-57

Iniesta B. J., 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, Southampton, U.K., 1994, ISBN 1-85312-353-6, pp. 579-594

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, Kluwer Academic Publishers, Dortrecht, Netherlands, 1993, pp. 61-76

Jones C., Applied Software Measurement, McGraw-Hill, New York, 1991, ISBN 0-07-032813-7

Kan S.H., Metrics and Models in Software Quality Engineering, Addison-Wesley, Readings, 1995, ISBN 0-201-63339-6

Knight J.C, and Myers E.A., An Improved Inspection Technique, Communication of the ACM, vol 36, no 11, 1993, pp. 51-61

Lorenz M., and Kidd J., Object-oriented Software Metrics, Prentice Hall, Englewood Cliffs, NJ, 1994, ISBN 0-13-179292-X

McCall J.A, Richards P.K, and Walters G.F., Factors in Software Quality, Volumes I, II, and III, RADC reports, 1977

Ould M.A., Strategies for Software Engineering: The Management of Risk and Quality, John Wiley &Sons, West Sussex, England, 1990, ISBN 0-471-92628-0

Russell G.W., Experience with Inspection in Ultralarge-Scale Developments, IEEE Software, vol 8, no 1, 1991, pp. 25-31

Testwell, CMT++ User's Guide, Complexity measures Tool for C/C++, Testwell Oy, Tampere, Finland, 1995