Quality knowledge capturing and reuse in software inspection

Tervonen Ilkka, Iisakka Juha, Kokkoniemi Jouni and Hiitola Paula

Summary:
The present paper introduces a concrete approach to the accumulation of quality knowledge during everyday work and illustrates how quality knowledge (lessons learned) can be captured and reused during the inspection process. Two kinds of quality knowledge are discussed. First, the classification of defects allows the gathering of history data (frequencies and types of defects) and thus the identification of potential problems in employees' expertise. Second, the conceptual maps constructed as summary reports of inspection and brainstorming sessions provide a way of capturing and reusing quality-based experience knowledge. The experiments reported here concern small projects in a partner company.

Tervonen Ilkka, Iisakka Juha and Kokkoniemi Jouni, Department of Information Processing Science, University of Oulu, P.O. Box 3000, FIN-90401, Oulu, Finland
Tel: +358 8 553 1908, Fax: +358 8 553 1890, E-mail: ilkka.tervonen@oulu.fi
Hiitola Paula, Rautaruukki Oy, Information Systems, P.O. Box 93, FIN-92101, Raahe, Finland

1. Introduction

It is generally accepted in software companies (throughout the organization structure) that existing knowledge is not utilized enough. This is attributed to the abstractness of software descriptions, and the complexity of software quality issues and explained by the theory of tacit and explicit knowledge, i.e. people know more than can expressed in words. In our earlier research (Tervonen and Kerola, 1998) we outlined the quality-based approach, which was based on shared tacit quality knowledge and shareable explicit quality knowledge, and especially on dynamic interaction in the development and use of tacit and explicit knowledge. In this paper we adopt a more concrete approach and illustrate how quality knowledge (lessons learned in software inspection) can be captured and reused during the inspection process. We pursue the idea of the accumulation of knowledge during everyday work (Birk and Tauz, 1998), which seems to be the ultimate goal of lessons learned management, and consider the establishing of a framework and tool environment in which people can acquire experience in well-structured form during their regular project work.

Two kinds of quality knowledge are discussed in this paper. First, we record the frequencies and types of defects. The classification of defects allows the gathering of history data and thus the identification of potential problems in employees' expertise. Second, we capture quality-based experience knowledge, which may be focused on the inspection process or on the product document and related documents. The experiments reported here concern small projects in a partner company and illustrate our first efforts of installing a light (painless) inspection practice in a company.

2. Experience capturing and reuse

There are two methods for experience capturing and reuse in the area of software engineering, the Knowledge Management of Software Engineering Lessons Learned method and the Experience Factory method reported by Birk and Tauz (1998) and Basili et al. (1994). The former contains four parts, i.e. gaining, packaging, disseminating and retrieving the lessons learned, while there are three basic strategies for gaining experience: using available knowledge sources, using goal-oriented knowledge acquisition and the accumulation of knowledge during everyday work. Using available

knowledge sources means relying on existing reports that have not yet been processed into reusable form (e.g. project memos, presentation slides, meeting minutes). Using goal-oriented knowledge acquisition (centralized knowledge elicitation) means conducting investigations specifically for the purpose of identifying lessons learned, e.g. by means of interviews, questionnaires and surveys. The accumulation of knowledge during everyday work (decentralized knowledge elicitation) deals with establishing a framework and tool environment in which people can gain experience in well-structured form during their regular project work. The latter strategy is the ultimate goal of lessons learned management, but the more common way to start it is to acquire already existing non-reusable experience statements and package them into lessons learned (Birk and Tauz, 1998, Tauriainen, 1999)

The packaging process converts arbitrary experience into the form of structured lessons learned. The main steps are the validation, structuring, making consistent and storing of the experience statements. The representation and retrieval of lessons learned is a matter of building up a semantic network that provides guidance for writing and using the lessons. Representation creates the basis for later retrieval strategies. Intelligent retrieval is usually necessary for finding the information relevant to a given problem. Storing and dissemination signifies a representation structure to be chosen and implemented for the lessons learned. The selected storage medium determines how the lessons learned can be disseminated throughout the organization.

Birk and Tauz (1998) suggest an infrastructure with case-base reasoning for finding applicable lessons learned. The system should be capable of handling informal data and allow a gentle transition towards formal knowledge representation. Establishing a lessons learned management system requires an understanding of a number of issues. Lessons learned are not by-products of project work, and dedicated effort is necessary to make informal statements reusable in future projects. The packaging of lessons learned needs to be done by people who are explicitly assigned to the programme, at least part-time, and users need to be explicitly committed to providing their experience. This ensures trust and high motivation to provide relevant experience.

3. Structure for quality knowledge

As discussed earlier, we consider two kinds of quality knowledge in this paper. The classification of defects and the gathering of statistical data is a well known and necessary activity for process improvement. The capture of quality-based experience knowledge focused on the inspection process and/or software artifact is not so popular, and its benefits are not as evident. We will now first introduce a defect classification principle, and then discuss the structure of quality-based experience knowledge.

3.1 Orthogonal Defect Classification

When we gather statistical data on software defects, we need a classification which is valid in all phases of the development process. This means that classification is not phase-dependent but orthogonal to phases, i.e. a defect type covers more than one process phase. Our classification is based on the IBM ODC (Orthogonal Defect Classification) method (Bassin et al. 1998, Chillarege et al. 1992) which defines eight attributes that collectively capture the semantics behind each defect. The method distinguishes activity from the schedule phase. Activity refers to what the individual was actually doing when the defect surfaced, regardless of the scheduled phase. This distinction provides ODC-based analysis with a critical advantage. Since ODC is activity-based, it is process independent, meaning that it can be used effectively regardless of the process model or technology being used. A generic list of activities might include design review, code inspection, unit testing, function testing and system testing, while a trigger captures the environment or condition that must have existed for the defect to surface, like a catalyst bringing forth a heretofore dormant defect. Insofar as the defect removal process tries to target these catalysts, the corresponding ODC triggers can be mapped to specific activities. The defect type represents the meaning of the corrective action required in terms of nature and scope. An analysis based on defect types allows an organization to evaluate product stability through development and to identify process weaknesses.

In our experiments we changed the ODC method in a partner company to be more easily installable in small projects. We focus here on defect types and the capturing of experience data on numbers and types of defects. We use checklists as triggers for choosing the most appropriate defect type, which means that, as in the ODC method, triggers are used for finding the activity. This classification of defects is semi-automatic, because each checklist has a pre-defined classification which is suggested and the inspector either accepts it, chooses another alternative, or leaves it undefined.

3.2 Structure of quality-based experience knowledge

The structure of quality-based experience knowledge is based on two questions, as illustrated in Figure 1: "How should a defect be classified?" and "What checklist should be used?". Checklists, which should be updated and tailored to each company, give solutions to both questions. In addition, when these checklists are used in inspection, the inspectors receive support in defect classification. The checklist, which the inspector considers, suggests the defect type for classification, which makes the task easier than in open classification. The semi-automatic nature of the classification also pushes the inspectors into making it.

Figure 1 illustrates the situation in the partner company. Based on existing checklists, we classified the potential defect types into 13 classes. The main idea for this classification was to find an orthogonal classification for defects, which supports the further gathering of experience data (e.g. the frequencies of the defect types). Another important idea was its familiarity, i.e. minimization of the number of new concepts and definitions.

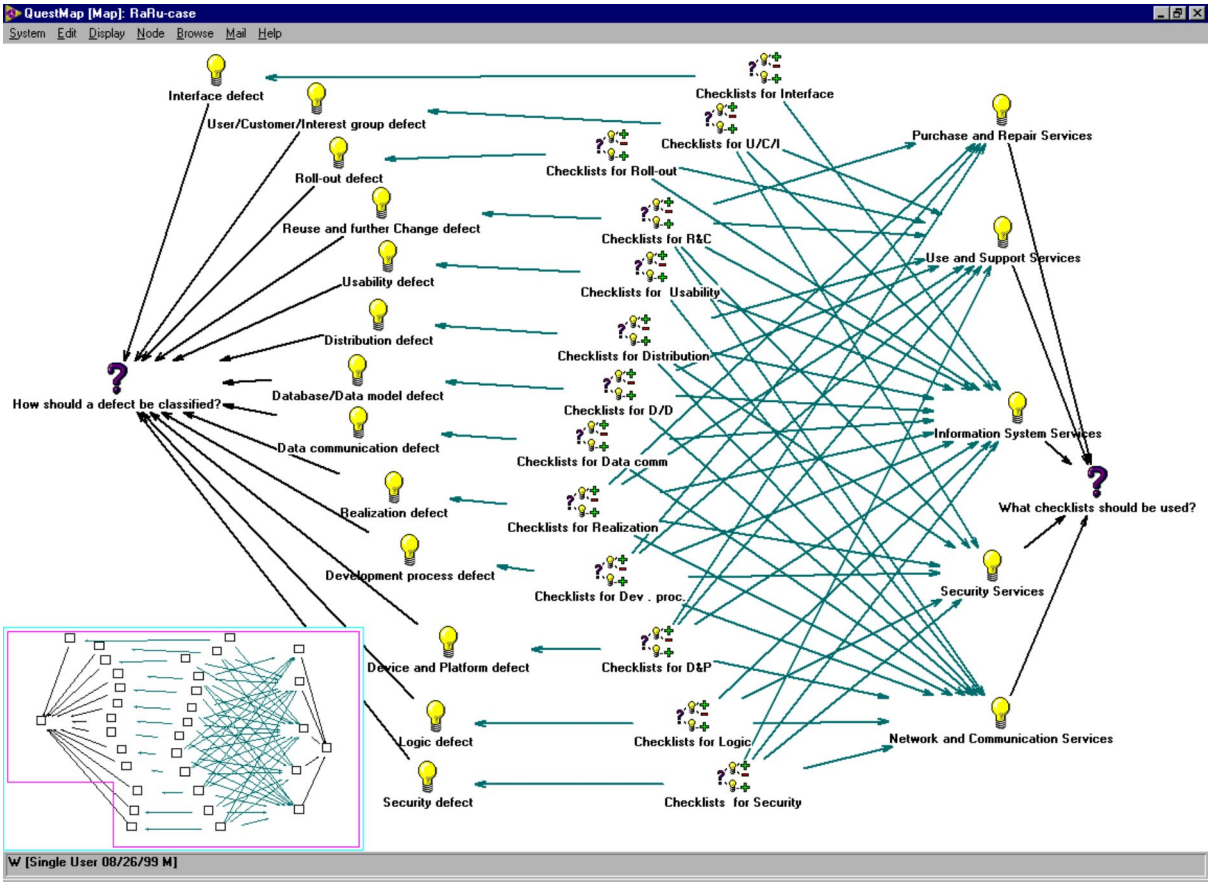

Figure 1. The structure of quality knowledge in the partner company

The checklists, from which the classification was derived, had been used in the company for years. The structure in Figure 1 illustrates the practice with small projects in a company. A small project is an order of work (OoW) which typically produces a specification for further implementation (which may be a project). An OoW may also be a maintenance activity which extends/removes/updates some part of an earlier implementation. An OoW is typically distributed among branches of activities, which are Information System Services (ISS), Use and Support Services (USS), Purchase and Repair Services (PRS), Security Services (SS) and Network and Communication Services (NCS). Depending on the complexity of the OoW, the system manager who is responsible for it decides which branches of activities will participate in it. As Figure 1 shows, each branch of activity uses a specific combination of checklists.

The defect classification is refined in more detail in Table 1, ten defect classes with related subclasses and the last three without refining. The idea in this classification is to suggest to the inspector the most relevant class (based on earlier consideration during the tailoring of checklists). The inspector can accept it, change it, or leave it blank if he/she cannot choose it.

Table 1. The defect classes with subclasses

distribution defect	1. distribution of work load, 2. scalability, 3. performance in problem situation
development process defect	1. conformable to standards, 2. appropriate level of abstraction, 3. planning of work, 4. distribution to branches of activities
usability defect	1. learnability, 2. ease of use, 3. effectivity of use, 4. checking of user input, 5. functionality of the application, 6. response time
device and platform defect	1. installation, 2. capacity, 3. configuration, 4. operability of the platform
logic defect	1. traceability and justification, 2. verifiability, 3. internal logic, 4. required data
data model defect	1. data modelling, 2. operability of data structures, 3. technical solutions in the database
data communication defect	1. data transfer capacity, 2. costs
realization defect	1. benefits, 2. realism, 3. robustness
security defect	1. operability of the firewall, 2. authentication, 3. encryption
roll-out defect	1. education, 2. support for users
user/customer/interest group defect	
reuse and further change defect	
interface defect	

How does this classification work? We have, for example, five checklists for distribution planning: DP1: "Does the distribution plan fulfil the requirements of the application?", DP2: "Have the alternative distribution plans been walked through?", DP3: "Has the required data transfer capacity been estimated?", DP4: "Are all time zones taken into account?", DP5: "Have the effects of potential failures been studied and recovery techniques defined?". In a workshop with company staff we discussed the relevant classification for defects uncovered by checklists and suggested the classification for these five as follows; DP1 -> "Logic defect, subclass; traceability and justification", DP2 -> "Distribution defect", DP3 -> "Data communication defect, subclass; data transfer capacity", DP4 -> "Logic defect, subclass; internal logic", DP5 -> "Distribution defect, subclass; performance in problem situation" or "Data model defect, subclass; technical solutions in the database". The main objective of these checklist and classification pairs is to help the inspectors in inspection and defect classification. In the classification step the inspector chooses the related trigger and classifies the defect, e.g. as a "Logic defect", and refines it to "traceability and justification defect". If the inspector is not sure that he/she can classify the defect at that level of detail, he/she can always choose the main class, i.e. " Logic defect " in this case. If the inspector doesn't agree with the classification it is always

possible to write in a separate opinion on the defect class.

4. Accumulation of quality knowledge

When we have experimented with this structure of quality knowledge in a partner company, the core information for the accumulation of quality knowledge was gathered in small projects (orders of work). The frequency of these projects is 500-600 per year, although at the moment we have inspected only a few of them. We plan to increase the frequency, so that inspection will be a regular part of small projects in the future.

The accumulation of quality knowledge in inspection is implemented in two cycles and in three forms. Two of the forms are implemented in a Lotus Notes[1] environment and one in a QuestMap[2] environment. The Notes forms are a "Quality Plan" form and an "Inspection Minutes" form. The "Quality Plan" reports the distribution of the OoW (order of work) to branches of activities (discussed in Figure 1) and records the participants and the type of inspection chosen, i.e. pair inspection for very small OoW and normal inspection for others. In summary, the experiments in inspection (e.g. what worked well, what problems were encountered) are also recorded with this form. The audit person, who must be chosen for each OoW separately, takes care of these summary experiments. The audit person is an independent person outside the project who is responsible for quality issues (e.g. that the inspection practice is followed). In the first inspections the quality manager acted as an audit person, but as a result of continuing education the number of capable audit persons is increasing.

In the first cycle, knowledge acquisition is implemented in a brainstorming session (third hour), which follows the logging meeting session in the inspection process. The audit person is responsible for the recording inspectors' comments on the inspection process, the rules and the checklists used. These comments are first recorded as a part of the "Quality Plan", but in the second cycle the quality manager periodically walks them through and draws a conceptual map (or adds to earlier maps) organized by a QuestMap tool (cf. Figure 2).

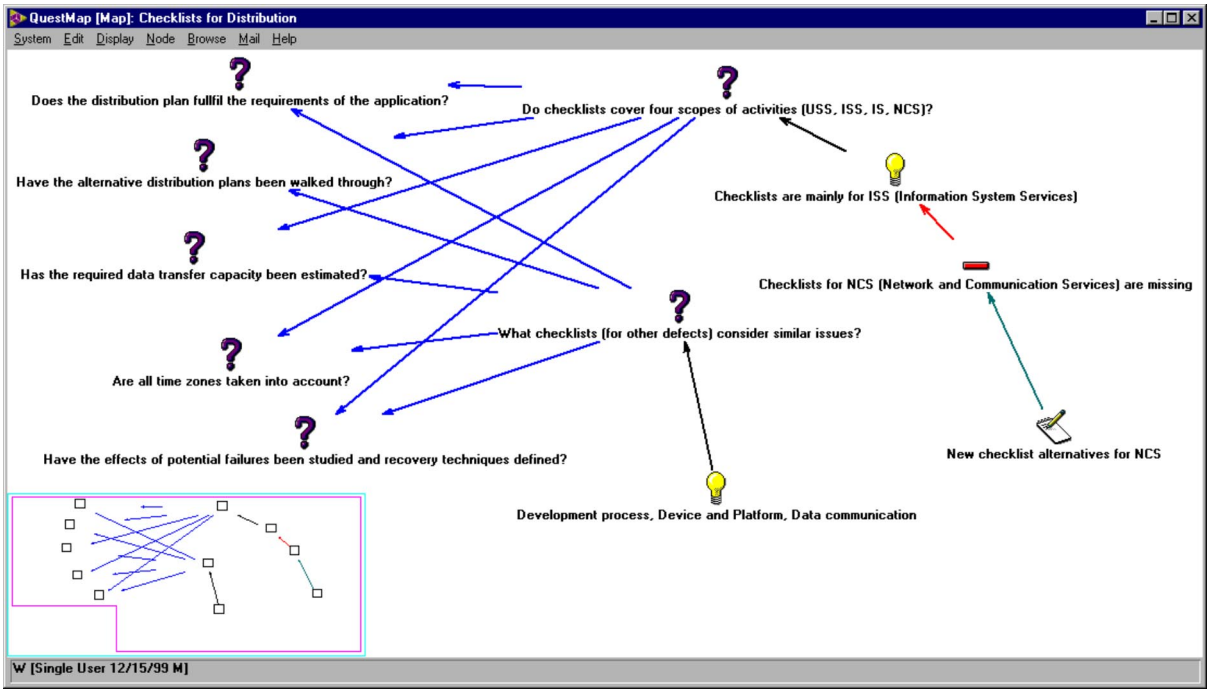

Figure 2. An excerpt from a discussion considering the coverage of the checklists

[1]Lotus Notes is a trademark of the Lotus Corporation
[2] QuestMap is a trademark of the GDSS Inc.

These conceptual maps (organized under the main structure in Figure 1) form the experience knowledge base, which will be updated through project experience and reused in new projects. Although the drawing of conceptual maps requires extra effort, we think that the easier understanding of these maps will justify the work.

The discussion of how well the five checklists (discussed earlier) cover all four branches of activities, i.e. USS (Use and Support Services), ISS (Information System Services), SS (Security Services) and NCS (Network and Communication Services) is depicted in Figure 2. The participants in the discussion argue that these checklists are mainly for ISS and that checklists for NCS, in particular, are missing. As a correction to this shortage they have also suggested new checklists tailored to NCS, e. g. "Have the data transfer costs been estimated?". If the participants agree that these checklists are relevant, they will be added to the normal group of checklists. Another discussion was focused on the topic of whether there are similar checklists connected with other types of defects (classification presented in Table 1). There is some doubt as to whether similar checklists could be found in the classes Development process defect, Device and Platform defect and Data communication defect.

5. Conclusions

The present paper adopts a concrete approach to the accumulation of quality knowledge during everyday work and illustrates how quality knowledge (lessons learned in software inspection) can be captured and reused during the inspection process. Two kinds of quality knowledge are discussed in the paper: the classification of defects, which allows the gathering of history data (frequencies and types of defects) and thus the identification of potential problems in employees' expertise, and conceptual maps constructed as summary reports of inspection and brainstorming sessions, which provide a way to capture and reuse quality-based experience knowledge. Work on this approach has just started, with small projects in a partner company. Although the classification of defects is based on checklists that have been in us for years, tailoring and adapting the approach to the company's practice will take a year at a minimum. The future will show whether the benefits gained in the form of a better understanding of conceptual maps are greater than the extra effort required for drawing them.

References

Basili V., Caldiera G., and Rombach H.D.: The Experience Factory, Marciniak J.J. (ed.), Encyclopedia of Software Engineering, vol 1, John Wiley & Sons, 1994, pp. 469-476

Bassin K.A., Kratschmer T., and Santhanam P.: Evaluating Software Development Objectively, IEEE Software, vol 15, no 6, 1998, pp. 66-74

Birk A., and Tauz C.: Knowledge Management of Software Engineering Lessons Learned, Proceedings of the Tenth Conference on Software Engineering and Knowledge Engineering, Illinois, Skokie: Knowledge Systems Institute, 1998, pp. 24-31

Chillarege R., Bhandari I.S., Chaar J.K., Halliday M.J., Moebus D.S., Ray B.K., and Wong M.: Orthogonal Defect Classification: A Concept for In-Process Measurements, IEEE Transactions on Software Engineering, vol 18, no 11, 1992, pp. 943-956

Tauriainen A.: Experience Capturing Process and Its Enactment, Master Thesis, University of Oulu, Department of Information Processing Science, 1999

Tervonen I., and Kerola P.: Towards Deeper Co-understanding of Software Quality, Information and Software Technology, 1998, vol 39, pp. 995-1003