Empirical Software Engineering, 10, 31–55, 2005.
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

A Survey of Documentation Practice within Corrective Maintenance

MIRA KAJKO-MATTSSON mira@dsv.su.se
Software Maintenance Laboratory, Department of Computer and Systems Sciences, Stockholm University, Royal Institute of Technology, and IT University, Electrum 230 SE-164 40 Kista, Sweden

Editor: Natalia Juristo

Abstract. The purpose of documentation is to describe software systems and software processes. Consistent, correct and complete documentation of a software system is an important vehicle for the maintainer to gain its understanding, to ease its learning and/or relearning processes, and to make the system more maintainable. Poor system documentation, on the other hand, is the primary reason for quick software system quality degradation and aging. Proper process documentation records the process, its stages and tasks, executing roles, their decisions and motivations, and the results of each individual process task. It is extremely important for achieving insight and visibility into the processes, important for their meaningful process measurement and thereby pivotal for achieving high process maturity. In this paper, we report on the results of an explorative study in which we have identified a number of rudimentary documentation requirements relevant within corrective maintenance, and found out how they were implemented within eighteen software organizations in Sweden. The goal was to examine the industrial documentation practice within corrective maintenance. Our results show that the documentation within corrective maintenance is still a very neglected issue within the organizations studied. None of our organizations has fully implemented all our documentation requirements.

Keywords: Corrective maintenance, software system documentation, software process documentation.

1. Introduction

Documentation of a software system, if correct, complete and consistent, is a powerful tool for the maintainers to gain success. It greatly facilitates the understanding and communication of the software system, eases the learning and relearning process, makes software systems more maintainable, and consequently improves the maintainers' productivity and work quality (Bauer and Parnas, 1995; Briand, 2003; Card et al., 1987; Carnegie, 1994; Clark et al., 1989; Conwell, 2000; Cook and Visconti, 1994, El Emam et al., 1998; Hogan, 2002; Holt, 1993; Kantner, 1997; Kantner, 2002; Martin and McClure, 1983; Parnas, 2000; Pence and Hon III, 1993; Rombach and Basili, 1987; Saunders, 1989; van Schouwen et al., 1993; Visaggio, 2001; Visconti and Cook, 2000; Visconti and Cook, 2002). Poor system documentation, on the other hand, is the primary reason for quick software system quality degradation and aging (Parnas, 1994; Sousa and Mendes Moreira, 1998; Visaggio, 2001; Visconti and Cook, 2002). It is a major contributor to the high cost of software maintenance and to the distaste for software maintenance work (Martin and McClure, 1983). Outdated documentation of a software system misleads the maintenance engineer in her work thus leading to confusion in the already very complex maintenance task.

The purpose of documentation however, is not only to describe a software system but also to record all the relevant process steps leading to its creation, update, modification, and deletion. Process documentation should be pervasive in and through all of the types of development and maintenance chores. Proper control and management of development and maintenance can only be achieved if we have obtained sufficient visibility into the process, its stages and tasks, executing roles, their decisions and motivations, and the results of each individual process task. This visibility provides an efficient vehicle for communication serving a variety of needs amongst various roles such as project managers, quality managers, product managers, upfront maintainers (support), testers and other roles. Process visibility is the main prerequisite for achieving mature processes.

1.1. Contribution

In this paper, we have identified a number of rudimentary documentation requirements relevant within corrective maintenance and found out how they were implemented within 18 software organizations in Sweden. Our requirements concern the documentation of both the software systems and software processes, the traditional processes.[1] They have been collected from the software engineering literature published within the past two decades. Our goal was to examine the industrial documentation practice within corrective maintenance. Although our study is limited to only the management of documentation within corrective maintenance, some of its results may also indicate the state of documentation practice within development and other maintenance categories.

1.2. Author's Standpoint

Annually, many billions of dollars are spent on software systems by industry, government and commerce. A significant portion of this investment is lost due to poor-quality software and/or lack of or inadequate documentation. A disproportionate amount of resources are spent on maintenance. The majority of the software community today, including the author of this paper, claim that documentation is an important prerequisite for successful management of software systems during their life-cycle (Bauer and Parnas, 1995; Briand, 2003; Card et al., 1987; Carnegie, 1994; Clark et al., 1989; Conwell, 2000; Cook and Visconti, 1994, El Emam et al., 1998; Hogan, 2002; Holt, 1993; Kantner, 1997; Kantner, 2002; Martin and McClure, 1983; Parnas, 2000; Pence and Hon III, 1993; Rombach and Basili, 1987; Saunders, 1989; van Schouwen et al., 1993; Sousa and Mendes Moreira, 1998; Visaggio, 2001; Visconti and Cook, 2000; Visconti and Cook, 2002).

Software system documentation is a key component in software quality. Empirical studies have shown that poor or missing documentation is a major cause of defects in maintenance. Software maintenance costs tend to be less for well-documented systems than for systems supplied with poor and/or incomplete documentation. (Card et al., 1987; Conwell, 2000; Cook and Visconti, 1994; Pigoski, 1997; Pence and Hon III, 1993; Rombach and Basili, 1987) Well-documented systems do not namely age in the same

tact as the undocumented ones. If undocumented or inadequately documented, software systems quickly grow in size and complexity, their architecture gets gradually undermined, and software engineers loose their familiarity. It is becoming more difficult to control the negative side effects arising from the change. The impact of change becomes more and more difficult to asses. The engineers may unknowingly introduce new defects each time they make a change to the software system. Good documentation, on the other hand, leads to an improvement in life-cycle productivity and makes future modifications and their verification much less onerous.

Software processes should be documented as well (Carnegie, 1994; Conwell, 2000; El Emam et al., 1998; Hogan, 2002; Lepasaar et al., 2001, Sousa and Mendes Moreira, 1998). Process documentation is a software process capability indicator (Carnegie, 1994; El Emam et al., 1998). The documented process shows how it works and how the process improvement actions influence it (Lepasaar et al., 2001). It provides a wealth of data to a variety of roles monitoring projects, processes and products. Inadequate process documentation makes this monitoring impossible. It is the main reason for loosing control over the software production process and for decreasing the chances to improve the quality of the processes.

1.3. Related Work

The state of documentation has for many years been a favorite subject. Different types of studies have revealed that documentation has been one of the largest problems within the software industry. For instance, in the late 70s, Lientz and Swanson identified the quality of application system documentation as one of the six major problems (out of 26 problems) (Lientz and Swanson, 1980). In the mid 80s, Chapin compiled a number of problems as identified by supervisors and managers of application software maintenance (Chapin, 1985). Out of 38 problems, documentation predominated as a major one. It scored twice as high as the second-ranked problem which was personnel capability. In the 90's, Sousa and Moreira concluded that the lack of documentation was one of the three main factors adversely influencing software maintenance (Sousa and Mendes Moreira, 1998). As late as in 2000, Cook and Visconti recognized a system documentation[2] process immaturity in industry today (Visconti and Cook, 2000) pointing out that there is an acknowledgement of the problem but little action to back it up. Finally, David Parnas has for many years pointed out that the software system documentation is the most neglected aspect of software engineering (Parnas, 1994; Parnas, 2000), despite the fact that it is the primary medium of software engineering, and the primary product of a software project (Parnas and Clements, 1993). In order to increase the documentation's consistency and completeness, Parnas has created a method based on a mathematical model of a software system and a mathematical notation for describing it (Bauer and Parnas, 1995; Schowen and Parnas, 1993).

How do we differ from the above-mentioned studies? In contrast to Parnas, and Visconti and Cook's work (Bauer and Parnas, 1995; Schowen and Parnas, 1993; Visconti and Cook, 2000), we consider both system and process documentation. However, we limit our study to only the domain of corrective maintenance, related to the traditional

development and maintenance methodologies. In contrast to Lientz and Swanson, Chapin and Sousa's work (Chapin, 1985; Lientz and Swanson, 1980; Sousa and Mendes Moreira, 1998), we do not try to rank the documentation problem with respect to other maintenance problems. We only check its state within a limited number of software organizations.

1.4. Methodology

Our study was of a qualitative and explorative character, in which we attempted to determine ''what things existed'' within a limited and subjectively chosen study group (Walker, 1985, p. 3). Our work consisted of three steps: (1) definition of process requirements and creation of the interview questionnaire, (2) interview of the organizations, and (3) data analysis.

During the first step, we created a list of documentation requirements. The requirements were thoroughly described and motivated for from a theoretical point of view. We identified and named the process requirements, identified their goals, and motivated why they should be implemented. We then created a series of interview open-ended questions inquiring on how the documentation requirement was implemented in the industry. The interview questions are presented in Appendix A.

During the second step, we interviewed the maintenance organizations. The interviews were conducted by our students attending a course on ''Software Evolution and Maintenance'' at Stockholm University and Royal Institute of Technology. It is them who chose the organizations. When doing this, they followed the ''convenience sampling'' method (Kane, 1984, p. 93). It is the simplest kind of the non-probability sampling method meaning that one studies anyone who happens to be around, is available, or is willing to be interviewed. This was the only sampling method that was realistic for us to use. We do not possess knowledge of the total population of software organizations in Sweden, and it is also very difficult to make organizations show willing for an interview. In addition pertinent interviewees may be difficult to locate. These facts yield any statistical methods impotent (Walker, 1985). For these reasons, our students were free to choose the organizations. They were allowed to choose just any organization (large/medium/small and/or private/state). The only requirement was that one student should interview one organization and that this organization should conduct corrective maintenance. The organizations studied are presented in Table 1.

Only one overlapping of the organizations has occurred. This concerns *Organization 1* (see Table 1). Two students interviewed two different software engineers coming from the same company. These engineers, however, belonged to two different departments possessing different documentation cultures.

To distinguish the interview results from these two different departments, we have identified them as *Org 1a* referring to the interview results of the first department and *Org 1b* referring to the interview results of the second department within *Organization 1.*

During the third step, we analyzed the collected data, and contacted the organizations anew in order to validate our results, that is, to check with them the correctness and credibility of the interview results delivered by the students. When presenting our results,

Table 1. Data about the organizations studied.

Organization ID	Name	No. of employees	Type of sw products/ service	Software size
Organization 1a	Postgirot Bank AB	3200	Financial systems	6000 programs
Organization 1b	Postgirot Bank AB	3200 (170 maintainers)	Financial systems	6000 programs in 168 different systems
Organization 2	Anonymous	55	Administrative systems	>100000 LOC
Organization 3	Anonymous	101000	Telecommunication products	unknown
Organization 4	Foyer Consulting	1	Consulting services	varying
Organization 5	Anonymous	300 (75 maintainers)	Telecommunication products	10 systems, 10000 LOC
Organization 6	Anonymous	300 (17 maintainers)	Information data bases	db- 122 million LOC db management - 142 programs application sw - 15 systems
Organization 7	Anonymous	35	Administrative	100 classes
Organization 8	Nexus	250	Embedded systems	10000 LOC
Organization 9	Martinsson Business Consulting	400	Financial systems	Unknown
Organization 10	FOA	1000 (15 developers)	Scientific systems	140000 LOC
Organization 11	WM Data Public Partner	550	Consulting services	Impossible to assess
Organization 12	Anonymous	90	Insurance	150000
Organization 13	Anonymous	475	Consulting services	Unknown
Organization 14	Anonymous	20	Administrative	Unknown
Organization 15	Anonymous	300 (5 programmers)	Support systems	50000 LOC
Organization 16	Anonymous	2000	Credit card systems	4 million LOC
Organization 17	Tele2	2000	Support systems	250000 LOC/ interviewed department
Organization 18	Anonymous	15	Maintenance of web sites	>10000 LOC

we have treated the results of the two departments within *Organization 1* as two separate results (*Org 1a* and *Org 1b*). Hence, in the presentation of our results, we used the total of 19 organizations (17 organizations plus *Org 1a* and *Org 1b*).

1.5. Credibility of our Results

The subject handled in this paper is highly sensitive. The individuals interviewed were assured that their and their organization's anonymity would be preserved, if required.

The interviewees were asked about this twice; first, directly after the interview, and second, when this paper was ready for publication.

Many organizations agreed on being interviewed only on the condition that the name of their organization would stay anonymous. As can be seen in Table 1, only seven out of 18 organizations allowed to put their name on show.

It is difficult to get a total insight into the companies' data through only interviews. In this study, we had to rely on the answers of the individual interviewees. These results are not representative for the whole company, only part of it. They merely represent the closest environment of the interviewees (usually their departments). Hence, our results may not always fully characterize the entire organization.

Finally, we would like to point out that the explorative and qualitative character of our study using the convenience sampling method does not allow us to generalize our findings beyond our study group. The results presented in the paper are generalized only with respect to the organizations studied.

2. Documentation Requirements

In this section, we present 19 requirements for product and process documentation within corrective maintenance. These requirements are the following:

R1: The system documentation is correct, complete and consistent when the system is transferred from development to maintenance. The goal of this requirement is to achieve good quality and maintainability of the system documentation from the very beginning. By complete documentation, we mean the degree to which the system, its functional- and non-functional requirement specifications, design, code, test cases constraints, and other important issues are documented. The software system must be properly documented when being transferred from development to maintenance. Otherwise maintenance engineers have no support for learning the overall purpose and structure of the software system, and for determining the software components to be affected by corrective modifications. The maintenance engineers cannot achieve good quality of documentation during maintenance if it is not right from the beginning.

R2: The software system is documented at all granularity levels of system documentation during corrective maintenance. The goals of this requirement are to keep up the complete specification of the software system at all granularity levels, to keep up the knowledge of the system, to preserve its maintainability and traceability and to make the maintenance process more efficient.

Software systems are continuously modified, either due to corrective or non-corrective changes. These changes must be visible at all granularity levels of a system documentation. Unfortunately, during maintenance and evolution, while source code evolves, other system documentation items are not updated (Antoniol et al., 2000, p. 240). The tedious and costly activity of maintaining consistency and traceability of the software system is frequently sacrificed due to market pressure. This results in aged documentation, which in turn, fails in its fundamental communication function (Clark et al., 1989).

R3: User manual is consistent with the state of the software system. The goal is to make software systems more usable from the users' perspective. Another goal is to

reduce the maintenance effort and increase maintainers' productivity, mainly on the upfront maintenance (support) level (Adams, 1984; Kriegsman and Barletta, 1993).

Software systems are definitely more usable, if they are properly documented in user manuals. The ideal is that the system is self-documenting so that the user manual is not needed (Hofmann, 1998). However, many systems are not self-documenting; they require detailed instructions. Their usability often depends on the support users can get from the manuals and other documents. (Holt, 1993; Saunders, 1989; Kantner et al., 1997, Kantner et al., 2002.)

It is not enough just to deliver user manuals. Its creators must make sure that their messages are delivered in the most helpful manner (Hendry et al., 1991). An incorrect information in the manuals fails as a communication facility. It may result in a substantial engagement of maintenance engineers at all organizational levels, especially on the up-front maintenance (support) level, who must mediate all inconsistencies in user manuals. Hence, correctly updated user manuals are pivotal for minimizing the maintenance cost, maximizing the maintenance productivity, and for improving the usability of a software system.

R4: Regression test case specification is revised and modified, if required. The goals are to reduce testing effort and to increase the quality of the software product. As soon as we make changes to a software system, we must not only test the modified parts but also the whole system or major part of it. This is because one modification in one part of the system may invalidate other parts (due to the ripple effect). To be sure that the whole system is still working, we should repeat all or the majority of the former tests. The repetition of the former tests is called ''regression testing''.

Regression test case documentation can be voluminous. It contains all the former tests that have been performed on the software system. New changes to the system can make some of the former tests redundant, obsolete or invalid. For this reason, the organizations should do some cleaning in their regression test case repository.

R5: Traceability between all levels of system documentation is ensured. The goals are (1) to keep up the consistency, completeness, and correctness of software system documentation, (2) to keep up the maintainability of the system, (3) to prevent system aging (Antoniol et al., 2000), and (4) to ease the understanding and learning/relearning process of the software system (Parnas, 1994).

According to the IEEE Std 610 (IEEE, 1999), ''traceability is the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match.''

With this definition, we understand that the traceability is the degree of a consistency of descriptions of different aspects of a software system in all types of system documents at all granularity levels. We also understand it as the exact identification and relation of a specific documentation item at one granularity level to another specific documentation item at some other granularity level. With this, we mean that a software requirement *R1* should be easily traceable to design specification *D1* and to software code unit *C1* and to test case specification unit *T1* and vice versa. Our understanding of traceability amongst all levels of system documentation is depicted in Appendix B.

R6: Traceability of change is ensured. The goal is to ensure that all the changes made to the software product can be traced from problem report to software system and vice versa. Another goal is to provide insight into the corrective maintenance process via problem reports.

The corrective changes to the product should be adequately documented. Documentation should identify all levels of the product affected by the change, down to the documentation line- or diagram level. By line, we mean a line in a text or in software code.

It is very important to know where exactly a software system has been changed due to which software problem. This helps measure *Annual Change Traffic (ACT)* (Boehm, 1981) and estimate future modification magnitude and effort within corrective maintenance. Insight into the corrective maintenance process via problem reports helps identify root causes of the software problems.

Emphasis should be placed on the ''whys'' behind each corrective modification. Understanding the reason for why a change to a product has been infused can help the maintainer in the design of future modifications, in performing root cause analyses pivotal for the evaluation of development and maintenance processes, and in evaluating the quality of a software product. Our understanding of traceability of change is presented in Appendix B.

R7: Guidelines for documenting a software system are defined. The goals are (1) to aid maintenance engineers in producing high quality documentation, (2) to aid in the evaluation of the quality of software system documentation, (3) to make the documentation process more effective, and (4) to keep up the quality of documentation uniform and consistent within the whole organization.

One may express one and the same thing in many different ways. The question is whether everybody will understand it uniformly. In many organizations, documentation is often terse, full of data jargon, or simply very badly written or drawn, if it is a diagram (Oskarsson and Glass, 1995). One way to remedy this is to create guidelines for what different types of documents should look like. Such guidelines are of great aid in producing high quality documentation. Having a common layout, writing rules such as how to build sentences, how to communicate (verbalize) information in writing, how to format the text and figures, how to draw diagrams, should aid in achieving a consistent, more readable and maintainable documentation.

R8: Guidelines for internal documentation are defined. The goal is to ensure the maximal readability and maintainability of software code. Software code and different kinds of program comments are examples of internal software code documentation. Even if the system is properly documented on all other documentation levels, we must still put a lot of emphasis on appropriately documenting software code. With this, we mean that software code must be visually organized according to some pre-defined layout, expressed according to some pre-defined organizational writing style, and accurately commented on.

R9: An organization has a checklist of all document types required for executing and supporting the corrective maintenance process. The goal is to aid the maintenance engineers in identifying pertinent documentation items required for maintaining the system.

Due to time pressure, many maintenance engineers only conduct the very basic activities required for conducting changes. Often changes require substantial update of

many different kinds of documents (electronic or paper ones). A lack of such a checklist could easily contribute to the fact that some documentation items might be omitted. Hence, the existence of and easy access to such a checklist would substantially improve the quality of documentation.

R10: Corrective maintenance process explicitly states where in the process each type of documentation should be updated/checked. The goal is to inform maintenance engineers where exactly in the corrective maintenance process they should work with what types of system and process documentation. Another goal is to ensure that the documentation process is adhered to.

Due to the enormous time pressure that the maintenance engineers encounter today, it is easy to neglect documentation in order to attend to some other urgent and high priority task. This may be enforced by the fact that maintenance engineers do not like writing documentation (Oskarsson and Glass, 1995). The documentation burden makes their task less interesting. Hence, documentation gets set aside in order to meet deadlines, or it is simply forgotten. One way of ensuring that all relevant documents get updated is to implement a documentation support into the corrective maintenance process.

R11: The corrective maintenance task is not approved as accomplished until the quality of all levels of system documentation for this task has been inspected. The goal is to ensure that all types of system documentation are consistent, complete and correct after a corrective change has been infused.

Still in many organizations, after the corrective task is completed in software code, the product is shipped to the user and maintenance engineers continue with the next tasks instead of updating other system documentation levels and checking their consistency. For this reason, we should infuse an inspection or a peer review during which the quality of all levels of system documentation is checked.

R12: The quality of the documentation is periodically checked by some documentation process manager. The goal is to ensure the future maintainability of the software system. Even if all corrective changes are controlled and inspected on all system documentation levels, inconsistencies may still arise in the system documentation. This is because one usually checks the quality of the software documents from the perspective of one change, but not from the total perspective. Periodically, we should check the quality of the entire system documentation and find out whether it follows the organizational standard.

R13: If not satisfactory or missing, the documentation of the existing software system/program is updated/created as soon as it is possible. The goal is to manage the inconsistencies in the system documentation as soon as they are discovered. The more one waits with updating the inconsistencies in the system documentation, the greater the burden to infuse modifications into the system in the future. Hence, all incomplete or inconsistent parts should be attended to as soon as it is possible.

R14 & 15: System Development Documentation and System Development Journal are available to the maintenance organization. The goal is to help maintenance organizations in the choice of appropriate decisions during postdelivery life cycle phase. When conducting maintenance, it is very important to know what the system looked like when being delivered and how it was developed during the development phase. For this purpose, we need access to system development documentation.

It is also very important to have access to all the experience gained during development. Experience is the most difficult piece of knowledge to transfer from development to maintenance (Pigoski, 1997). It communicates the reasons for making decisions during development, project goals and priorities, day-to-day problems, project successes and failures. Experience should be documented in a software development journal.

Software development documentation together with software development journal may substantially improve the ability to make the right decisions during maintenance. Many facts may be quickly forgotten, either because the developers may no longer be available or because the human ability to memorize is strongly constrained.

Knowledge of the original design and intentions behind its choice may guide the maintenance engineer in choosing appropriate ways to modify the software system during maintenance. Knowing which parts of the system were regarded as the most difficult ones may help explain problems in these parts, in pinpointing the chronic problems, and in suggesting future directions.

R16: All system changes are recorded during corrective maintenance. The goal is to have control over all the corrective modifications made to the software system. Just as with the software system development documentation, we should archive information about problems found in and changes made to the software system during maintenance. Hence, all problems and changes should be communicated in a formal and tractable way. If the problems are not reported in writing but orally, they may be easily forgotten or misunderstood. A lot of additional effort may be put on reworking a problem only to discover that it has already been resolved (Arthur, 1988). The changes can neither be properly tested nor traced to any maintenance requirement (problem report in our context). The organization cannot achieve control over the number of reported problems and their severity. It has no data on the change traffic within corrective maintenance, and hence, it cannot effectively plan for corrective maintenance on an annual basis (Boehm, 1981). Many fundamental software engineering tasks such as assessment of product quality, change validation and verification, measurement and estimation of current and future modification size and effort cannot be accurately performed.

R17: History of the reported software problems is recorded for each product/ product component. The goal is to better understand the behavior or rather misbehavior of the system. There is high probability that defective system parts will continue to be defective if no drastic measures are made. Recording the history of problems may give good insight into the course of modification events, the frequency of problems, their severity and influence on certain parts of the system. The problematic parts should be in turn candidates for re-engineering.

Problem history may also help evaluate the software quality and the effectiveness of the methodologies. The effectiveness of the methodologies may be determined by identifying the deficiencies in the development and maintenance processes. Examples of such deficiencies might be inadequate testing, inadequate tools, lack of product or process knowledge, and other deficiencies. This information could be very useful when suggesting process improvements.

R18: System Maintenance Journal is kept for monitoring the corrective maintenance process. The goal is to record all experience gained, practical problems encountered during corrective activities, decisions made and other issues. Just as with

development journal, it is essential to find out what practical problems maintenance engineers encountered when conducting changes. Motivations behind decisions are very important. It may happen that an organization has chosen a certain solution alternative in order to make the software product execute faster and cheaper. This may be easily forgotten after some time.

R19: An organization arranges education/training in written proficiency for maintenance engineers. The goal is to improve the quality of documentation. It is especially important for the engineers responsible for updating user manuals, the so-called ''technical writers''.

Software engineers write professionally for both themselves, other engineers and the general public. In many cases, they produce bad quality documentation. We believe that many times it is due to the lack of training in written proficiency. Programmers are trained to write in Cobol, Java, C++, but not to write in English, Swedish or other languages. They are not trained to verbalize themselves in writing. The lack of writing skills leads to resistance to document. Please observe that many organizations in Sweden do not use their native language as the official organizational language. They use English instead. This implies extra difficulties when documenting. The resultant documentation is often difficult to understand.

To start teaching them how to write is actually too late now. Organizations may however somewhat remedy this by creating common rules for simple written communication (Buican, 1990; Malcolm, 2001; Norman and Holloran, 1991), by creating appropriate techniques that might help produce quality documentation (Delanghe, 2000, Saunders, 1989), or simply by employing a professional technical writers (Edmands, 1988, Ramsay, 1997).

3. The State of Documentation Practice

In this section, we present the results of our interviews. For each documentation requirement, we present its status within the organizations studied. This status is also summarized in Table 2.

R1: The system documentation is correct, complete and consistent when the system is transferred from development to maintenance. In 53% of the organizations studied, the system documentation is correct, complete and consistent in this phase. In these organizations, this is an absolute requirement for the transfer of a software system from development to maintenance. Usually, the developers must follow a certain checklist of all the system parts that have to be delivered to the maintenance organization. The delivery must then be approved by the maintainers.

For almost 16% of the organizations studied, this requirement is not entirely fulfilled. System development documentation is completed somewhat later during maintenance. In 26% of the organizations studied, the system documentation is either missing or it is not correct and complete in this phase. In some of these organizations, the steering group closes the project before the maintenance organization has a chance to check the system documentation. Finally, for the remaining 5% of the organizations studied (*Org 16*), this fact is unknown. This is because the system was developed a very long time ago.

Table 2. Documentation requirements (R1 – 19).

Documentation requirement	Org 1a	Org 1b	Org 2	Org 3	Org 4	Org 5	Org 6
R1: The system documentation is correct, complete and consistent when the system is transferred from development to maintenance	+	P	–	+	P[10]	+	–
R2: The software system is documented in system is documentation at all granularity levels during corrective maintenance.							
• Requirement specification	+	+[1]	+	+	+[1]	+	P
• High Level Design specification	+	+	+	+	+[1]	+	P
• Low-Level Design specification	+	+	–[3]	–[3]	+	+	P
• Software code documentation	+	+[2]	+[2]	+[2]	+	+	+[2]
• User Manual	+	P	+	+[8]	+	+	+
• Unit test case documentation	+	+	–	+	+	+	–
• Integration test case documentation	+	+	+	+	+	+	–
• System test case documentation	+	+	+	+	+	+	P
• Regression test case documentation	+	+	+	+	+	+	–
R3: User manual is consistent with the state of the software system.	P	+	+	+	+	+	P
R4: Regression test case specification is revised and modified.	+	+	+	+	P	+	–
R5: Traceability between all levels of system documentation is endured.	–	P	–	+[27]	P	+	–
R6: Traceability between the problem report and the system documentation is ensured.	U	P	–	–	P	+	–
R7: Guidelines for documenting a software system are defined.	–	–	+	P	+	+	–
R8: Guidelines for internal documentation of software code are defined.	–	+	+	–	+	+	–
R9: An organization has a checklist of all documents required for executing and supporting the corrective maintenance process.	–	+	–	–	+	+	–
R10: Corrective maintenance process explicitly states where in the process each type of documentation should be updated/checked.	–	–	–	–	+	+	–
R11: The corrective maintenance task is not approved as accomplished until the quality of all levels of system documentation for this task have been inspected.	–	–	P	–	+	+	–
R12: The quality of the documentation is periodically checked by some documentation process manager.	+	+	+	+	+	–	–
R13: If not satisfactory or missing, the documentation for the existing software system/program is updated/created as soon as possible.	P[14]	P[10]	P[15]	–	P	–	–
R14: System Development Documentation is available to the maintenance organization.	+[13]	+	+	+	–	+	+

Org 7	Org 8	Org 9	Org 10	Org 11	Org 12	Org 13	Org 14	Org 15	Org 16	Org 17	Org 18
+	+	+	+	+	–	+	P[10]	–	U	+	–
+	+	+	–	+	–	+[1]	+[1]	–[5]	–	+	–
+	+	+	–	+	–	+	+[1]	–	–	+	+
–	+	+	–	+	–	+	–	–	+	+	+
+	+	+	+	+[2]	+[2]	+	+[2]	+[2]	+[2]	+	+[2]
+	P[6]	P[1,6]	P[6]	U[7]	+[1]	+	P[8]	P[8]	+[9]	+	P
–	+	P	–	+	–	+	–	–	–	+	P
+	+	+	–	+	+	+	–	–	–	+	P
+	+	+	–	+	+	+	P[6]	–	–	+	P
+[4]	+	–	–	P	–	+	–	–	–	+	–
+	P[6]	P[6]	P[6]	U[7]	–	+	+[8]	P[6,8]	+[9]	+	–
–	+	–	–	–	–	+	–	–	–	+	–
–	+[27]	P	–	–	–	+[27]	–	–	–	+	–
P	P[18]	+[27]	–	+[27]	P	+[27]	P[18]	–	P	P[18]	–
–	P	P	–	P	–	+	P	–	–	P	–
+	–	P	P	P	P	+	+	–	–	+	–
–	–	+	–	–	–[11]	P	–	–	–	+	–
–	–	–	–	–	–	+	–	–	–	+	–
+[12]	+	+	–	–	–	+[12]	–	–	–	+	–
–	–	+	–	–	–	+	–	–	–	–	–
P	+	+	+	–	–	+	+	–	–	+	–
+	+[16]	+	+	+	–	+	+	–	U	+	–

Table 2. Continued.

Documentation requirement	Org 1a	Org 1b	Org 2	Org 3	Org 4	Org 5	Org 6
R15: System Development Journal is kept available to the maintenance team.	–	–	+	–	–	–	P[14]
R16: System changes are recorded during corrective maintenance.	P[14]	P[14]	+	+	+	+	P[5]
R17: History of the reported software problems is recorded for each product/ product component.	U	P	–	P[24]	+	+	P[24]
R18: System Maintenance Journal is kept for monitoring the corrective maintenance process.	–	+[19]	P[20]	–	P[20]	P	–
R19: Organization organizes education/ training in written proficiency for maintenance engineers.	–	–	P[21]	–	P	–	–

Comments on the table:

+ The documentation requirement is implemented.
– The requirement is not implemented.
P Partial implementation meaning that the organization only partially implements the documentation requirement or that the implementation of this requirement varies in the organization.
U Unknown.

1. Not updated during corrective maintenance.
2. Only updated as comments in software code.
3. There is no division into high-level and low-level design specification.
4. Regression test case documentation is integrated together with unit-, integration-, and system test case documentation.
5. Communicated orally.
6. Not systematically updated.
7. Unknown due to the fact that user manuals should be developed and maintained by the user.
8. A crib.
9. Created/implemented/satisfied recently.
10. Completed somewhat later.
11. Only for development.
12. Only via peer reviews.
13. It is not easy to find information in it.
14. Only in some systems.
15. Depends on the priority of the requirement.
16. Utilized very often.
17. Informal implementation.
18. Traceability from code to problem report.
19. Stored but not utilized.
20. In the form of minutes.
21. Only a few seminars.
22. Via a mentor.
23. Only when starting major projects.
24. Only on a whole product level, or major component level.
25. A very informal one.
26. State written requirements for what the language should look like. No active education, however.
27. Amongst the documents they are using within corrective maintenance.

Org 7	Org 8	Org 9	Org 10	Org 11	Org 12	Org 13	Org 14	Org 15	Org 16	Org 17	Org 18
–	–	+	P[25]	P	–	P	–	–	U	+	–
+	+	+	P	+	+	+	+	+	P[9]	+	–
P	–	P[24]	–	+	+	+	P[24]	–	P	+	–
–	–	+	P[20]	P[20]	P	+	–	–	P	+	–
–	–	–	P[26]	–	–	P[23]	–	–	–	P[22]	–

R2: The software system is documented at all granularity levels of system documentation during corrective maintenance. Only 16% of the respondents claim that their organizations update their system documentation at all granularity levels during corrective maintenance. Additional 11% claim that they update all documents required except for requirement and/or high-level design specifications (*Org 4* and *Org 13*). In these organizations and organizations such as *Org 1b* and *Org 14*, one does not update requirement specifications within corrective maintenance.

For the remaining respondents (63%), the software system is not documented at all granularity levels. The choice of granularity levels varies. Two organizations, for instance, have only software code and user manuals. For them, user manuals serve as a requirement specification. Two other organizations communicate their requirements via email.

Concerning the regression testing, 42% of the organizations studied do not regression test their systems within corrective maintenance. Hence, they do not update any regression test case documentation.

R3: User manual is consistent with the state of the software system. Our results show that all organizations, except for *Org 11*, have user manuals. Usually, a particular group, the so-called technical writers, is responsible for monitoring the changes in the system and for documenting them in user manuals. 47% of respondents claim that their user manuals are consistent with the state of the software system. The manuals always get updated before the delivery of a new release. Almost 32% of the respondents claim that their user manuals are partly consistent and complete. These manuals are not updated regularly. They may be updated only when major changes have been infused into the system. The reason is the high cost. Finally, in the remaining organizations, the manuals are not updated at all or this fact is simply unknown. In *Org 18*, one does not update the manuals because their customers do not demand them, despite the fact that they pay the full price for their products.

R4: Regression test case specification is revised and modified, if required. Only 42% of the organizations studied revise and modify their regression test case

repositories. However, they may not always conduct regression testing on all system levels. For instance, five organizations, *Org 1a*, *Org 1b*, *Org 5*, *Org 8*, and *Org 17* conduct regression testing at all levels. *Org 2* conducts it on integration and system level. In *Org 3*, the choice of testing levels depends on the product to be tested, however, all products undergo the regression testing on the system level. Finally, *Org 13* conducts regression testing mainly on the unit level. The testing on other levels may be chosen, if required.

As has been mentioned before, 42% of the organizations do not conduct regression testing within corrective maintenance at all. They only test the changes made on mainly the unit and integration level, sometimes (not always) on the system level, and ship the product to the customer. The reason is lack of time and the high cost.

R5: Traceability between all levels of system documentation is ensured. Only 11% of the organizations studied have achieved the traceability amongst their documents. Here, we refer to the organizations that update and use documentation at all system granularity levels (*Org 5* and *Org 17*). These organizations have special roles dedicated for checking the traceability amongst the documents.

Almost 16% of the organizations studied (*Org 3*, *Org 8* and *Org 13*) claim that they have achieved the traceability amongst the documents that they are using. Eleven percent have achieved partial traceability amongst their documents. For instance, *Org 1b* has achieved only one-way forward traceability from requirements to code, but not vice versa. *Org 9* has achieved one-way backwards traceability, that is, traceability from code to requirements. They have no traceability to test case specifications.

The remaining organizations have not achieved any traceability at all. At its best, their code may be consistent with its comments. User manuals are only updated in connection with major changes.

R6: Traceability of change is ensured. Only 5% of the organizations (one organization *Org 5*) have achieved full traceability of change—the organization that updates and uses documentation at all system granularity levels within corrective maintenance. In this organization, one is not allowed to terminate the corrective task, if not all relevant documents have been updated and the traceability of change has been ensured. In this organization, one may easily see if the change has been made either due to customer complaint or an enhancement, or whether it has been generated internally within the organization. One may trace all the system parts that have been changed, one may see who has made changes, what process activities have been undertaken to conduct the change and so on.

Sixteen percent of the organizations studied (*Org 9*, *Org 11*, and *Org 13*) claim to have achieved traceability of change amongst the documents they are using. However, they are not using all the documents at all system granularity levels.

Forty two percent of the organizations studied have achieved partial traceability of change, usually one-way traceability from software code to problem report using the problem report identifier. One may see what modules have been changed, but not which of its lines. To see this, one mainly conducts the ‘‘diff’’ command using a version management tool. Finally, 32% of the organizations have not fulfilled this requirement at all. At its most, one makes comments in the code, and records the problem report identifier for some important problems.

R7: Guidelines for documenting a software system are defined. Only 21% of the organizations studied successfully implement this requirement. In these organizations, one requires that the guidelines are followed meticulously. Forty seven percent do not implement this requirement at all. The remaining organizations (32%) only provide some very general guidelines and/or ready layouts for different types of documents. Hence, they have fulfilled this requirement only partially.

R8: Guidelines for internal documentation are defined. The status for this requirement looks somewhat better than for the requirement R7. Forty two percent of the organizations studied have defined guidelines for software code. Twenty one percent of the organizations studied have defined guidelines, but these guidelines may either be very general or they may not always be followed. Finally, 37% of the organizations do not provide any guidelines at all. Their programmes are free to choose their own programming and commenting style.

R9: An organization has a checklist of all document types required for executing and supporting the corrective maintenance process. Only 26% of the organizations studied provide such a checklist for their employees. Their engineers are obliged to use it when conducting changes to their systems. Such a checklist may be deduced either via system handbooks or process supporting tools. For 5% of the organizations, this requirement is partially implemented. They have a checklist for only major maintenance changes. The remaining organizations, 68%, do not implement this requirement at all.

R10: Corrective maintenance process explicitly states where in the process each type of documentation should be updated/checked. This requirement is implemented in only 21% of the organizations studied. Usually, it is built in the tool supporting the process. Hence, there is little risk that some process step may get omitted. The rest depends on the discipline of the individual engineer.

R11: The corrective maintenance task is not approved as accomplished until the quality of all levels of system documentation for this task has been inspected. Only 26% of the organizations have a formal inspection and approval of the quality of system documentation after each corrective change. Another 11% conduct an informal approval such as peer reviews. Almost 58% do not inspect/review their changes in the system documentation at all. The responsibility stays with the software engineer. Some of these organizations, however, are on their way to infuse some kind of a quality control.

R12: The quality of the documentation is periodically checked by some documentation process manager. Almost 37% of the organizations studied perform this type of a quality check. The time interval varies depending on the budget and time. In most of these organizations, however, such quality checks are mainly made before shipping major releases (usually every six months). The remaining organizations do not perform this type of control at all. The main reason is lack of time and resources.

R13: If not satisfactory or missing, the documentation for the existing software system/program is updated/created as soon as it is possible. Our results show that 32% of the organizations studied fully implement this requirement. Incompleteness and/or inconsistency are attended to as soon as possible, depending on the priority and severity of other change tasks. Just as any other request for change, they are reported

in problem reports, assigned priority and severity, and managed just as any other change request. In *Org 17*, however, one attends to the inconsistencies and incompleteness immediately, that is, as soon as they are discovered. They claim that they cannot afford to do it otherwise. The future changes would be too difficult and costly.

Twenty six percent of the organizations studied fulfil this requirement only partially. The main reason is the lack of time at the moment of discovery of this type of inconsistency. Hence, this task is set aside to be conducted as soon as time allows it.

R14 & 15: System Development Documentation and System Development Journal are available to the maintenance organization. Almost 74% of the organizations studied have access to system development documentation. It is mainly used in cases when major changes are infused into the system, or when one needs to educate and train new hires. Twenty one percent of the organizations studied do not have access to system development documentation at all. Some of them, however, have expressed an unhesitating need for it.

Regarding the system development journals, only 16% of the organizations studied have access to them. They mainly use them when major important changes are made. The remaining organizations do not have system development journals, or if they do, then these journals are very informal.

R16: All system changes are recorded during corrective maintenance. All except one organization (*Org 18*) implement this requirement. The system changes are however recorded at the system level. Sixty eight percent of the organizations studied record all changes to all their software systems. The remaining ones do it only for some systems.

R17: History of the reported software problems is recorded for each product/ product component. Thirty two percent of the organizations studied keep history of the reported software problems for their software components. They usually use this information for identifying and analyzing problematic software components. Twenty six percent of the organizations studied do not keep history of the software problems at all. The remaining ones implement this requirement only partially. They keep history of software problems on the whole system level or major subsystem level.

R18: System Maintenance Journal is kept for monitoring the corrective maintenance process. Twenty one percent of the organizations studied keep a system maintenance journal. This journal may be easily accessed and consulted. It provides an important basis for decisions on future changes.

Forty two percent of the organizations studied do not keep a maintenance journal at all. They either use a mount-to-mouth method or some individual engineers may keep their own records for personal use. The have stated many varying reasons for this: (1) too much stress in the project, (2) lack of time and resources, or (3) lack of need or interest (they just want to write code). Due to high personal turnover or sick leave, some of these organizations have realized that it would be useful to have such a journal.

Regarding the remaining organizations, they keep minutes from different meetings during which important change decisions are made. These minutes are recorded and may be accessed if need arises.

R19: An organization arranges education/training in written proficiency for maintenance engineers. None of the organizations studied gives some form of education in written proficiency. Twenty six percent of the organizations implement this requirement partially by giving a series of seminars or by stating written directives on what the language should look like. Finally, in one organization, *Org 17*, the newly hired engineers always get a mentor helping them document the system. This often implies that the engineers inherit the mentor's writing style, which may not always be optimal.

Many interviewees have expressed the need for implementing this requirement. According to them, the quality of most types of documentation would be substantially improved if the engineers were given this type of education and training.

4. Conclusions

In this paper, we have presented the results of our survey of the state of documentation practice within corrective maintenance in Sweden. We have defined a set of basic requirements important for documenting software systems and process steps during corrective maintenance, and found out how they were implemented within 18 subjectively chosen organizations.

Our results show that documentation within corrective maintenance is still a very neglected issue. None of the organizations studied has fully implemented all our requirements. Only three organizations (Org 5, Org 13, and Org 17) have almost fulfilled all of them. Our results also show that the documentation culture may vary within one and the same organization. This can be proved by checking the status of two departments (Org 1a and Org1b) belonging to Organization 1. Regarding the status of the organizations studied, we have learned the following lessons:

- Maintainers are not provided with full support for conducting maintenance from the very beginning. Still, the system documentation is not correct, complete and consistent when the system is transferred from development to maintenance. It may be or may not be completed later during the maintenance phase.

- Regarding the status of documentation after delivery, software systems are not continuously documented at all granularity levels. The choice of granularity levels varies. At its worst, the system may be documented only at the code level. Manuals, if any, are not always consistent with the state of the software system. Traceability of system documentation and traceability of change have not been achieved.

- Attempts to achieve and improve the quality of system documentation are minimal within the organizations studied. The majority of the organizations do not provide guidelines for how to document their software systems. And, if they do, these guidelines are not always followed. None of the organizations studied educate and train their maintenance engineers in how to write documentation. At its most, engineers may attend to a series of seminars, be provided with some written directives, or they may be assigned a mentor helping them write documentation.

- Support for identifying the documents to be updated during corrective maintenance is minimal. The majority of the organizations studied do not provide the checklists of the documents required for executing and supporting the corrective maintenance process. Nor do they explicitly identify the process steps during which different types of documentation should be updated and/or checked.

- The quality and maintainability of the system documentation is not systematically monitored and improved. The majority of the organizations studied do not inspect and approve the quality of system documentation after each corrective change. Nor do they check and improve it periodically.

- Support for making decisions about future changes is poor. Very few organizations have access to system development and system maintenance journals providing important feedback on various decisions and experience gained during development and maintenance. Maintainers have vague insight into all the corrective modifications made to the system and their history. Hence, they cannot effectively evaluate the quality of their systems and the effectiveness of their development and maintenance methodologies.

Summing up, the documentation practice within the organizations studied is poor. Even if the majority of the interviewees understood the necessity of documentation, they claimed that the prognoses for improvements did not look promising. Writing good documentation is hindered due to many reasons, the reasons mainly related to the current status of both their systems and processes. The quality of their systems is substantially undermined. Improving it would require major re-engineering effort, which the organizations cannot afford right now. Furthermore, the organizations studied lack clear and detailed guidelines for what the documented quality software systems should look like. Concerning the processes within the organizations studied, they do not provide sufficient directives for when, how, and why to document certain development and maintenance process steps. In other words, the engineers are not motivated enough for writing process documentation.

Many of our interviewees claimed that detailed documentation standard models would greatly help them establish their system and process documentation processes, and thereby improve their documentation discipline. As Parnas says, documentation is the primary medium of software engineering, and we, researchers, have failed in providing this medium (Parnas and Clements, 1993). We have failed to provide the software industry with standard models of system and process documentation. We should definitely consider it in our research ages ago. Nothing actually hinders us to start doing it now. Better late than never!

Acknowledgments

We would like to thank all the Swedish industrial representatives (our interviewees) who have helped us to conduct this study. They are Gudmundur Thorvardarson, Sasan

Enayatollah, Jan-Eric Claesson, Leif Thedvall, Jan Lindviken, Claes Ericsson, Jordanis Caracolias, Åsa Gustafson, Per Foyer, Göran Näsman, Helena Lindström, Bo Andersson, Toni Baknor, Gunnar Holm, Urban Kvarby, Åsa Wikenståhl-Paulson, Anneli Haage and Peter Jörgne.

We would also like to thank all the students that helped me find and interview the above-mentioned practitioners. They are Lola Andersson, Andreas Eriksson, Fredrik Börjesson, Nina Grundemark, Sandra Giarimi, Rasmus Gunnar, Karl Heling, Mia Kokko, Magnus Pettersson, Maria Pettersson, Åsa Maria Pettersson, Jolanta Plisko, Maria Luisa Quiroga de Arreyes, Fabian Rivière, Helena Sjöberg, Fredrik Schmidt, Urban Örthberg, Leila Vanhala-Packendorff, Tove Wättestam.

Finally, I wish to express my gratitude to professor Jan Wretman at the Statistical Department at Stockholm University for his support and feedback concerning the choice of the methodology chosen for this study.

Appendix A. Our Questionnaire

A.1. Introductory Questions:

- What is the name of your company?
- What is the number of employees?
- What type of products do you maintain?
- What is the size of your products (in LOC or number or programs/systems)?

R1: Is the system documentation at your organization correct, complete and consistent when the system is transferred from development to maintenance?

R2: On what levels does your organization document the system during corrective maintenance?

- Requirement specification
  - High Level Design specification
  - Low-Level Design specification
  - Software code documentation
- User Manual
- Unit test case documentation
- Integration test case documentation

- System test case documentation
- Regression test case documentation

R3: Does our organization actively update user manual?
Is it consistent with the current (present) state of the software system?

R4: Does your organization actively control and modify the contents of regression tests?

R5: Is software code consistent with other types of documentation?
What types of documentation is it consistent with?

R6: Can each corrective change in system documentation be traced back to the problem report, and vice versa?

R7: Does your organization provide any guidelines for how to document software systems?

R8: Has your the organization defined rules for internal documentation of software code, for instance, programming guidelines, guidelines for how to structure software code, guidelines for how to comment on software code, and the like?

R9: Does your organization provide a checklist of all documents required for executing and supporting the corrective maintenance process?

R10: Does the corrective maintenance process clearly identify the process steps during which documentation should be checked and modified?

R11: Are all corrective maintenance tasks inspected/reviewed and approved?

R12: Is the quality of the system documentation periodically checked by some documentation process manager?

R13: Does your organization attend to the incorrectness, incompleteness and inconsistency in the software system documentation. If yes, how often? If not, why?

R14: Does the maintenance team have access to System Development Documentation?

R15: Does the maintenance team have access to System Development Journal?

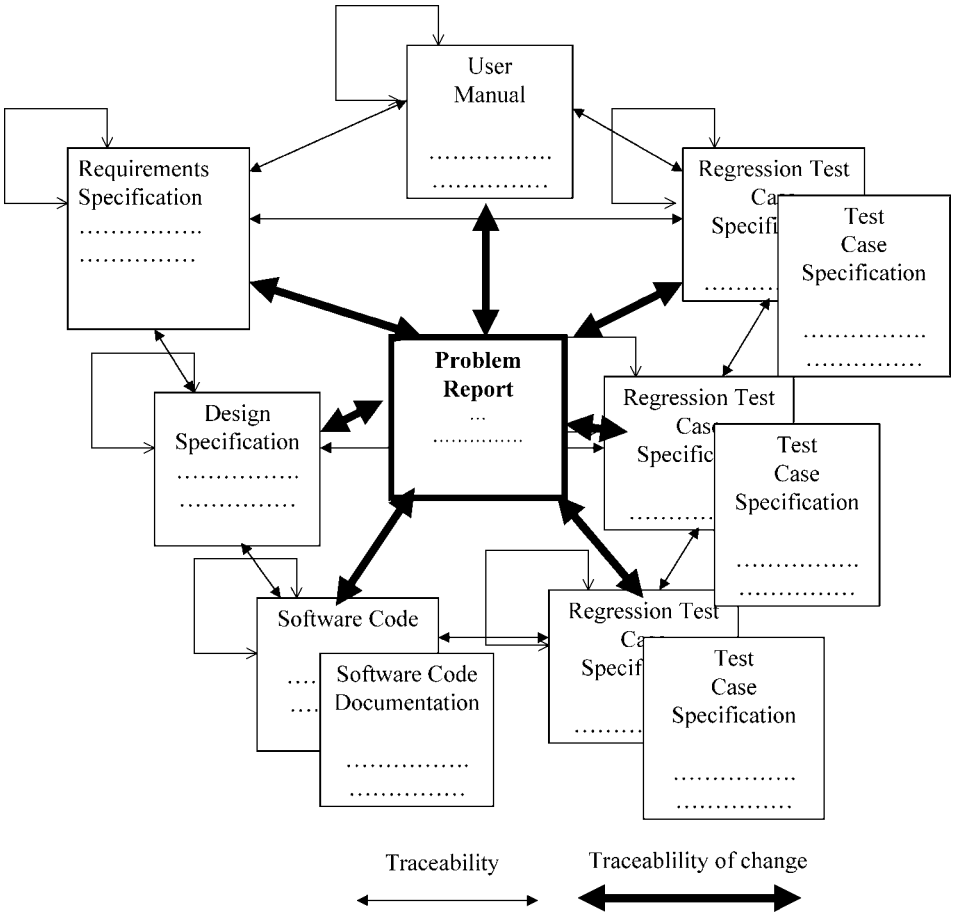
R16: Does your organization record all the corrective maintenance changes?

R17: Does your organization record the history over the reported software problems and defects for each product/product component?

R18: Does your organizations keep System Maintenance Journal for recording system changes?

R19: Does your organization organize education/training in written proficiency for engineers?

Appendix B. Our Understanding of Traceability and Traceability of Change



Notes

1. We do not consider lightweight processes.
2. User manuals were excluded in this study.

References

Adams, E. N. 1984. Optimizing preventive service of software products. *IBM J. Res. Dev.* 28(1): 2–14.

Antoniol, G., Canfora, G., Casazza, G., and De Lucia, A. 2000. Information retrieval models for recovering traceability links between code and documentation. *Proc. IEEE Int. Conf. Softw. Maint*. 40–49.

Arthur, L. J. 1988. *Software Evolution: The Software Maintenance Challenge*. John Wiley & Sons.

Bauer, B. J., and Parnas, D. L. 1995. Applying mathematical software documentation: An experience report, proceedings. *Proc. 10th Annual Conference on Computer Assurance*. 273–284.

Boehm, B. W. 1981. *Software Engineering Economics*. Prentice-Hall, pp. 538–540.

Briand, L. C. 2003. Software documentation: How much is enough? *Proc. IEEE Int. Conf. Softw. Maint. Reeng*. 13–15.

Buican, I. 1990. International English. *Proc. IEEE Int. Conf. Prof. Commun.* 21712–21714.

Card, D., McGarry, F., and Page, G. 1987. Evaluating software engineering technologies. *IEEE Trans. Softw. Eng.* 13(7): 845–851.

Carnegie Mellon University and Software Engineering Institute. 1994. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley.

Chapin, N. 1985. Software maintenance: A different view. *AFIPS Proc. Natl. Comput. Conf*. AFIPS Press, Reston, Virginia, 54, pp. 508–513.

Clark, P. G., Lobsitz, R. M., and Shields, J. D. 1989. Documenting the evolution of an information system. *Proc. IEEE Natl. Aerosp. Electr. Conf*. 1819–1826.

Conwell, C. L. 2000. Capability maturity models support of modeling and simulation verification, validation, and accreditation. *Proceedings, IEEE Winter Simulation Conference*. 819–828.

Cook, C., and Visconti, M. 1994. Documentation is important. *CrossTalk* 7(11): 26–30.

Delanghe, S. 2000. Using learning styles in software documentation. *Proc. IEEE, Transact. Prof. Commun*. 201–205.

Edmands, A. 1988. The integrated editor: Involvement of an editor in the entire information development process. *Proc. IEEE Int. Prof. Commun. Conf*. 119–122.

El Emam, K., Drouin, J-N., and Melo, W. 1998. *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*. IEEE Computer Society Press.

Hendry, D., et al. 1991. How people use softcopy manuals: A case study. *Proc. IEEE Int. Prof. Commun. Conf*. 221–224.

Hofmann, P. 1998. Away with words! How to create wordless documentation. *Proc. IEEE Int. Prof. Commun. Conf*. 437–438.

Hogan, J. M. 2002. The real world software process. *Proc. IEEE 9th Asia-Pac. Softw. Eng. Conf. (APSEC'02)*. 366–375.

Holt, P. O. 1993. System documentation and system design: A good reason for designing the manual first. *Proc. IEEE Colloquium on Issues in Computer Support for Documentation and Manuals*. 1/1–1/3.

IEEE Standards Collection, 1999. Software Engineering. The Institute of Electrical and Electronics Engineers, Inc.

Kane, E. 1984. *Doing Your Own Research*. Marion Boyars, London & New York.

Kantner, L., et al. 1997. The Best of both worlds: Combining usability testing and documentation projects. *Proc. IEEE Int. Prof. Commun. Conf*. 355–363.

Kantner, L., et al. 2002. Structured heuristic evaluation of online documentation. *Proc. IEEE Int. Prof. Commun. Conf*. 331–342.

Kriegsman, M., and Barletta, R. 1993. Building a case-based help desk application. *IEEE Expert* 8(6): ISSN: 0885-9000, pp. 18–26.

Lepasaar, M., Varkoi, T., and Jaakkola, H. 2001. Documentation as a software process capability indicator. *Proc. IEEE Int. Conf. Manage. Eng. Technol*. 1, pp. 436.

Lientz, B. P., and Swanson, E. B. 1980. *Software Maintenance Management, A study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley.

Malcolm, A. 2001. Writing for the disadvantaged reader. *Proc. IEEE Int. Conf. Prof. Commun*. 95–100.

Martin, J., and McClure, C. 1983. *Software Maintenance, The Problem and Its Solutions*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 07632.

Norman, R. L., and Holloran, R. W. 1991. How to simplify the structure of administrative procedures to make them easier to write, review, produce, and use. *Proc. Int. Conf. Prof. Commun*. 2, pp. 447–450.

Oskarsson, Ö., and Glass, R. L. 1995. *ISO 9000 i Programutveckling – Att Konstruera Kvalitetsprodukter*. Studentlitteratur, in Swedish.

Parnas, D. L. 1994. Software aging. *Proc. 16th Int. Conf. Softw. Eng*. 279 – 287.

Parnas, D. L. 2000. Requirements documentation: Why a formal basis is essential. *Proc. IEEE 4th Int. Conf. Requir. Eng*. 81 – 82.

Parnas, D. L., and Clements, P. C. 1993. A rational design process: How and why to fake it. *IEEE Trans. Softw. Eng.* 12(2).

Pence, J., and Hon III, S. 1993. Building software quality into telecommunication network systems. *Quality Progress*. October, 95 – 97.

Pigoski, T. M. 1997. *Practical Software Maintenance*. John Wiley & Sons.

Ramsay, J. 1997. Corporate downsizing: Opportunity for a new partnership between engineers and technical writers. *Proc. IEEE Int. Prof. Commun. Conf*. 399 – 403.

Rombach, H., and Basili, V. 1987. Quantitative assessment of maintenance: An industrial case study. *Proc. IEEE Conf. Softw. Maint*. 134 – 144.

Saunders, P. M. 1989. Communication, semiotics and the mediating role of the technical writer. *Proc. IEEE Int. Prof. Commun. Conf*. 102 – 105.

Sousa, M., and Mendes Moreira, H. 1998. A survey of the software maintenance process. *Proc. IEEE Conf. Softw. Maint*. 265 – 272.

van Schouwen, A. J., Parnas, D. L., and Madey, J. 1993. Documentation of requirements for computer systems. *Proc. IEEE Int. Symp. Requir. Eng*. 198 – 207.

Visaggio, G. 2001. Aging of a data-intensive legacy system: Symptoms and remedies. *Journal of Software Maintenance and Evolution: Research and Practice*. Wiley & Sons, 13(5): 281 – 308.

Visconti, M., and Cook, C. R. 2000. An overview of industrial software documentation practices. Technical Report 00-60-06, Computer Science Department, Oregon State University, April.

Visconti, M., and Cook, C. R. 2002. An overview of industrial software documentation practice. *Proc. 12th IEEE Int. Conf. Chilean Comput. Sci. Soc*. 179 – 186.

Walker, R. (ed.) 1985. *Applied Qualitative Research*. Gower Publishing Company Ltd.

Mira Kajko-Mattsson is a PhD within Software Maintenance. Presently, she works as a lecturer and researcher at the Department of Computer and Systems Sciences at IT University in Stockholm in Sweden. Her early educational background lies within humanities, which she had studied at the University of Poznan in Poland and Stockholm University in Sweden. In 1984, she switched her interests to computer science. In 1986, she took the BA degree in Data and Systems Sciences at DSV. In 1998, she took her PhL degree in software engineering. In 2001, she earned her PhD in Software Maintenance in 2001. Presently, she is the author of more than 40 international publications (book chapters, journal and conference articles). Her research area is corrective software maintenance. She has recently created a process model of problem management within corrective maintenance *CM*^{3}*: Problem Management*. Right now, she is working on developing a process model of upfront maintenance (support) called *CM*^{3}*: Upfront Maintenance*. Her email address is mira@dsv.su.se.