1

Templates for Communicating Information about Software Requirements and Software Problems

Mira Kajko-Mattsson
Department of Computer and Systems Sciences,
Stockholm University/Royal Institute of Technology
Sweden

1. Introduction

For years, the need and scope of software system and process documentation has been heavily debated. Questions that have often been raised are whether one should document, what one should document, how much and when. This debate has lasted for more than 30 years. Still, however, we do not have any clear answers. What we claim and what we have always claimed is that both software system and software process documentation are very important for allowing the software systems to live a long life. (Bauer and Parnas, 1995; Briand, 2003; Card et. Al. 1987; Carnegie 1994; Clark et.al., 1989; Connwell, 2000; Cook and Visconti, 1994, El Elman et.al., 1998; Hogan, 2002; Holt, 1993; Kajko-Mattsson, 2005; Kantner, 1997; Kantner, 2002; Martin and McCluer, 1983; Parnas, 2000; Pence and Hon III, 1993; Rombach and Basili, 1987; Saunders, 1989; van Schouwen and Parnas, 1993, Visaggio, 2001; Visconti and Cook, 2000; Visconti and Cook, 2002)

Recently, the debate regarding documentation has intensified. This is because "agile" voices have been raised against extensive documentation (Beck, 2004; Cohn, 2006; Nawrocki, J et.al, Jasinski, M., 2002). These voices advocate its sparse use and production. According to them, documentation should be brief, however, precise enough to be useful within software production. It should be limited to the core parts of the system. In its extreme case, it should encompass only source code and a set of user stories. The remaining system structure may be communicated informally, mainly orally. There may be no requirement specifications nor design documents, or if there are any, then for the purpose of supporting the coding activity.

The agilists' standpoint with respect to documentation has created confusion within the software community. Right now, the software community has a problem of determining the amount and scope of required documentation. Still, it cannot agree on issues such as (1) what one should document, (2) how much, (3) when, and (4) for what purpose. Today, we do not have any concrete answers. We only claim that we should document just enough information for supporting the communication. But, how much enough is enough?

In this chapter, we present two templates of information required for describing and managing software requirements and software problems. We call them *Software Requirements Management Template (SRMT)* and *Problem Management Template (PMT).* These templates record information about the system to be developed or maintained and the process managing the development or maintenance effort. The templates have been evaluated within the industry in various parts of the world. Our goal is to establish information that is needed for communicating information about software requirements and software problems and the information required for their realization within a software lifecycle.

The chapter is going to be structured in the following. Section 2 motivates why we need to document. Section 3 describes the method taken to evaluate the *SRMT* and *PMT* templates. Section 4 presents the *SRMT* template covering information required for communicating requirements and their realization. Finally, Section 5 provides concluding remarks.

2. Why Do We Need Document?

Documentation, if correct, complete and consistent, has been regarded as a powerful tool for software engineers to gain success. Its purpose is to describe software systems and processes.

Good system documentation thoroughly describes a software system, its components and relationships. It facilitates the understanding and communication of the software system, eases the learning and relearning process, makes software systems more maintainable, and consequently improves the engineer's productivity and work quality (Bauer, 1995; Briand, 2003; Card, 1987; Conwell, 2000; Parnas 2000, Visconti, 2002).

Poor system documentation, on the other hand, is regarded to be the primary reason for quick software system quality degradation and aging (Parnas, 2000; Sousa, 1998; Visconti, 2002). It is a major contributor to the high cost of software evolution and maintenance and to the distaste for software maintenance work. Outdated documentation of a software system misleads the engineer in her work thus leading to confusion in the already very complex evolution and maintenance task.

The purpose of documentation however, is not only to describe a software system but also to record a process. Process documentation is extremely important for achieving (1) insight and visibility into the processes, (2) meaningful process measurement and thereby (3) high process maturity. It should be pervasive in and throughout all the types of lifecycle chores. Proper control and management of development, evolution 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 individual process tasks.

Irrespective of whether documentation supports systems or processes, it constitutes a collective knowledge of the organization. It supports communication and collaboration amongst various groups within the organization such as project managers, quality managers, product managers, support technicians, testers, developers, and other roles (Arthur, 1995). It enhances knowledge transfer, preserves historical information, assists ongoing product evolution and maintenance, and fulfills regulatory and legal requirements.

It provides an efficient vehicle for communication serving a variety of needs. We distinguish between two types of documentation:

- *Self to self*: Software professionals make personal notes. The goal is to remember what they have done, plan for future actions and analyze the results of their own personal processes.
- *Software professional to software professional*: Different roles communicate with one another, such as developers to developers, developers to testers, developers to maintainers, developers and/or maintainers to management, management to developers and/or maintainers, developer and/or maintainer to support personnel, and the like. The goal is to provide feedback to one another.

3. Method

In this section, we present the method taken to create the *PMT* and *SRMT* templates. First, in Section 3.1, we present our method steps. We then in Section 3.2 present the companies involved in creating and evaluating the templates.

3.1 Steps

Our work consisted of two major consecutive phases, starting as early as in 1998. These phases are: (1) *Creation and Evaluation of Problem Management Template (PMT)* and (2) *Creation and Evaluation of Software Requirements and Management Template* (*SRMT*). Below, we briefly describe these steps.

In the years of 1998 – 2001, we did research on problem management within corrective maintenance. This research resulted in a problem management process model, called *Corrective Maintenance Maturity Model (CM³): Problem Management*. It covered the process activities and information required for managing software problems. It is the problem management information that is put into the *PMT* template. *CM³: Problem Management* was created within ABB (Kajko-Mattsson, 2001). The model and the template were then evaluated within fifteen software companies.

PMT is a specialized form of *SRMT*. Problem reports are requirements for change within corrective maintenance. They are generated due to specific software problems and defects. For this reason, we decided to adapt it to the *SRMT* template managing overall software requirements. The *SRMT* template was then evaluated in three consecutive different studies.

First, we evaluated the SRMT template within one Canadian company which practiced both traditional and agile development approach (Kajko-Mattsson, Nyfjord, 2008). We then continued evaluating it within six Chinese companies (Kajko-Mattsson, 2009). Finally, we evaluated it within one major Swedish company together with 60 software engineers. The evaluation was conducted in form of three consecutive workshops.

Due to space restrictions, we cannot describe the evaluation results in this chapter. However, we cordially invite our readers to study our publications in which we report on

the documentation status within the industry today (Kajko-Mattsson, 2000; Kajko-Mattsson, 2002, Kajko-Mattsson, 2005, Kajko-Mattsson & Nyfjord, 2008, Kajko-Mattsson, 2009).

3.2 Companies involved in the Evaluation Phase

Two ABB organizations supported us in creating *CM³: Problem Management*. These were *ABB Automation Products* and *ABB Robotics*. In addition, we received scientific support from *ABB Corporate Research.* (ABB, 2009)

Within ABB, we explored the process of problem management within corrective maintenance and the information managed within this process. Here, we interacted with all the possible roles involved within the process such as maintainers, team leaders, testers, support technicians, change control boards, and the like.

CM³: Problem Management and the *PMT* were then evaluated within fifteen non-ABB organisations in 2001. When choosing them, we attempted to cover a wide spectrum of the today's IT state. Deliberately, we chose different sizes and different types of software organisations. Our smallest organisation was a one-person consulting company. Our largest organisation was one major organization having 14 000 employees. The types of products maintained by these organisations varied from financial systems, business systems, embedded real-time systems, consulting services, administrative systems, and different types of support systems.

Regarding the *SRMT*, we first evaluated it within one Canadian organization. We did it in 2007. Here, we interviewed one representative (a process owner). The company develops products ranging from ground stations for satellite radar systems to e-commerce applications. We studied one small and one middle-sized project evolving data transaction systems with high quality and performance requirements.

We then continued to evaluate the *SRMT* template in a Chinese software development context. When visiting Fudan University as a guest professor in fall 2007, we took the opportunity to study the requirements management practice within some Chinese organizations (Kajko-Mattsson, 2009). For this purpose, we used students attending our course on advanced software engineering. The students chose the organizations according to the convenience sampling method (Walker, 1985). These companies practiced both traditional and agile development methods.

Finally, we evaluated the *SRMT* together with 60 software engineers within one big software development company. Due to the fact that the company wishes to stay anonymous, we call it *IT Exchange*. The evaluation was made in form of three consecutive workshops. As illustrated in Figure 1, 65,57% of the workshop participants were involved in programming activities. The remaining 34,43% were involved in other non-programming activities.

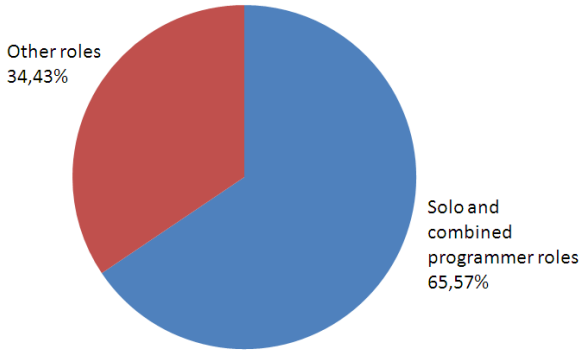

Fig. 1. Distribution of roles at IT Exchange by programmer and non-programmer roles

Figure 2 shows a more detailed distribution of roles involved. Here, we may see that the range of roles involved in the three workshops spanned from programmers, to testers, project managers, technical staff, architects and managers. These individuals were involved both in traditional and agile development and maintenance.

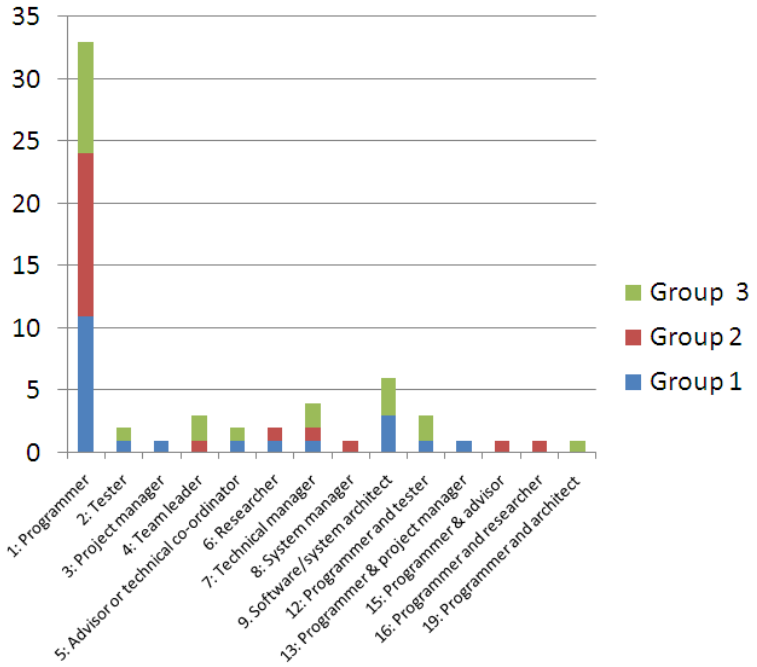

Fig. 2. Roles involved in the workshops at IT Exchange

4. Templates

This section presents and motivates the *SRMT* and *PMT* templates. It first gives an overall view and motivation of the templates in Section 5.1. It then describes the two templates in Section 5.2.

4.1 Overall Presentation of the Templates

The *SRMT* and *PMT* are similar, however, they vary in their contents. Both describe the information about either new requirements or software problems and their management throughout the software lifecycle. However, they require different types of information. This is because the *SRMT* describes new functionality that is going to be created whereas the *PMT* describes old functionality that is going to be changed.

It is difficult to provide a good and clear description of new requirements and software problems. They may be described in several ways using different terms. They can be depicted in different environments and on different system levels (Wedde et.al, 1995).

A proper requirement and problem description is the most important prerequisite for effective implementation. A poor, sketchy, or misleading description may lead to misinterpretation, and thereby, to misimplementation (Kajko-Mattsson, 2000). For this reason, a description of a requirement and a problem should be clear, complete and correct. It has to be communicated in a structured and disciplined way.

To aid in minimising the reporting time for the submitters and in maximising the quality of the reported data, the software organisation should give guidance on how to provide and structure requirement and problem description data. This can be done in form of templates.

The templates should document essential information about all additions and changes made to a system. Since the templates are used to document and communicate information to a wide variety of roles, it is important to include enough information to meet the needs of those roles. At minimum, the information should answer the following questions:

- What is to be implemented/changed?
- Why is it going to be implemented/changed?
- How is it going to be implemented/changed?
- What is the budget?
- Where is it going to be implemented/changed?
- Who is going to make the implementation/change?
- Is the description clear and concise? Are all clues and leads adequately described to allow impact analysis to begin?
- If not, is the submitter available for clarification?
- What information is extraneous to the request?

System Documentation	Process Documentation
General Requirement Description: Requirement ID, Requirement Title, Requirement Description, Requirement Type, Internal/External Req., Rationale, Event/Use Case ID, Related to Requirement(s), Conflicting Requirements, Non-functional requirements, Constraints (Solution, Technical, Budget, Resource), Intended User, Specific user who stated the req., Customer Satisfaction, Customer Dissatisfaction, Assumptions, Reference documents. **Requirements Evaluation Data** Business Value, Other value, Requirements Priority (Rank), Acceptance Criteria, Fit Criteria, Risk **Other Description Data** System Data (System ID, Sub-System ID, Component ID), Adjacent/interfacing Systems ID, Environment	**Requirements Reporting Data** Requirements Reporting Date, Originated by, Reported by, Requirements Owner **Requirements Management Data** Preliminary Implementation Plan, Preliminary outline of activities (Design, Implementation, Testing, other), Change Activities, Planned and Actual Activities(s) (*Activity Description, Activity Start Date, Activity End Date, Expected/Actual Result of Activity Take, Activity Conducted By, Activity Approved By, Effort Spent On Activity, Cost of Action)* **Requirements Management Progress Data** Requirement Management Status, Requirement Mngmt Status Date, Requirement Age, Requirement Changes **Requirements Completion Data** Actual Completion Date, Planned Completion Date, Relation To Tests, Released In, Requirements Completion Approved By, Signed Off Date, Signed Off By, Estimated Total Effort, Actual Total Effort, Estimated Total Cost, Actual Total Cost **Post Implementation Data** Analysis of the Requirements Implementation Process, Lessons learned

Fig. 3. Software Requirement Management Template

System Documentation	Process Documentation
General Problem Description: ProblemID, ProblemTitle, ProblemDescription, Internal/External Problem., Related to Problem(s), Problem Effects and Consequences, Problem Symptoms, Problem Conditions, Problem Reproducibility, Alternative Execution Paths, Assumptions, Reference documents. **Requirements Evaluation Data** Problem Priority (Rank), Problem Severity, Risk **Other Description Data** System Data (System ID, Sub-System ID, Component ID), Adjacent/interfacing Systems ID, Environment, **Problem Reporting Data** Problem Reporting Date, Originated by, Reported by, Problem Owner	**Problem Management Data** Preliminary Resolution Plan, Preliminary outline of activities (Design, Implementation, Testing, other), Change Activities, Planned and Actual Activities(s) (*Activity Description, Activity Start Date, Activity End Date, Expected/Actual Result of Activity Take, Activity Conducted By, Activity Approved By, Effort Spent On Activity, Cost of Action)* **Problem Management Progress Data** Problem Management Status, Problem Mngmt Status Date, Problem Age **Problem Completion Data** Actual Completion Date, Planned Completion Date, Relation To Tests, Released In, Requirements Completion Approved By, Signed Off Date, Signed Off By, Estimated Total Effort, Actual Total Effort, Estimated Total Cost, Actual Total Cost **Post Implementation Data** Analysis of the Problem Resolution Process, Lessons learned

Fig. 4. Problem Management Template

The *SRMT* and PMT consist of two main sections; one dedicated to system documentation and the other one dedicated to process documentation. As listed in Figure 3 and Figure 4, each section covers a set of attributes bearing on coherent information. The attributes concerning the system documentation are (1) *General Requirement/Problem Description,* (2) *Requirement/Problem Evaluation Data*, and (3) *Other Description Data*. The attributes concerning the process documentation are (1) *Requirement/Problem Reporting Data*, (2) *Requirement/Problem Management Data*, (3) *Requirement/Problem Management Progress*, (4) *Requirement/Problem Completion Data*, and (5) *Post Implementation Data.*

4.2 System Documentation

In this section, we describe the tree clusters used for documenting the system. Just because the descriptions of new requirements and problems somewhat differ, we first describe and

explain the attributes used for describing the new requirements in the *SRMT* template. For each of the clusters in the *SRMT*, we then describe their correspondences in the *PMT* template.

General Requirement/Problem Description

The *General Requirement Description* describes basic requirement information needed for identifying, understanding, and classifying requirements (Atlantic, 2007; Higgins, 2002). It covers the following attributes:

- *Requirement ID:* Each requirement should be uniquely identified. This allows the requirement to be traced throughout the whole lifecycle process. Usually, its ID corresponds to a numerical value. Some of the requirements may however be identified with an alphanumerical value.
- *Requirement Title:* A title is a short name of a requirement. However, it is not an identifier. It rather corresponds to a mnemonic problem identification. It usually consists of several keywords. It is very helpful in communicating on requirements and in doing manual searches in the tool recording the requirements. It allows one to quickly browse through a requirements list without having to read the whole requirement description.
- *Requirement Desciption:* General information describing the requirement in free text. The requirement orginator describes his own needs and motivates them. This description may be quite comprehensive. Usually, there is no space limit for this field.
- *Requirement Type:* Specification of whether the requirement concerns some new or some existing behavior of the system or whether it concerns some non-functional requirement specifying the characteristics of some functinality.
- *Internal/External Requirement:* Specification of whether the requiement was requested externally by the customer or internally within the development organization. This specification enables priority assignement to the software requirement. Usually, all external requests get higher priority than the internal ones.
- *Rationale:* Motivation of the requirement d'être, that is, the rationale behind the requirement to help the developers understand the requirement and the reason behind it. This helps the developer understand the application domain. Rationale is of great importance in monitoring requirement‘s evolution during its lifecycle. It disambiguates unclear requirements, and thereby, it prevents from changes leading to unexpected effects.
- *Event/Use Case ID:* List of events and/or use cases descripting the requirement. The use cases, if any, should always be identified. They provide a basis for specifying the requirements.
- *Related To Requirement(s):* Link to other requirements related to the requirement at hand. By following this link, one may achieve an overall picture of groups of requirements and their relationships. In this way, one may discover inconsistencies and duplications among the requirements. One may also record the hierarchies among the major and minor requirements, like the ones presented in Figure 5.

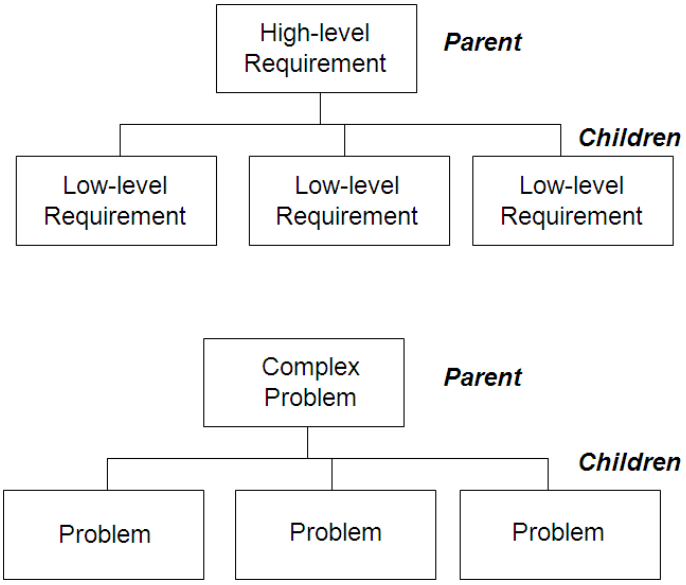

Fig. 5. Hierarchies among the requirements and problems

- *Conflicting Requirements:* Links to conflicting requirements. This information is used for future measures such as negotiations with the customers or provision of extra resources to create an appropriate design that matches these requirements.
- *Non-Functional Requirement(s):* A link to non-functional requirements specifying the criteria to be used when developing the requirement.
- *Constraints:* List of restrictions imposed on resources, budget, technical or solution constraints. These restrictions may lead to a modification of a requirement, and thereby, limit the range of its solutions.
- *Intended User:* Identification of all types of users of the requirement.
- *Customer Satisfaction:* Specification of the degree of satisfaction of how the requirement will meet the customer's needs and expectations.
- *Customer Dissatisfaction:* Degree of customer dissatisfaction, if the requirement is not successfully implemented.
- *Assumptions*: The software system is intrinsically incomplete. The gap between the system and its operational domain is bridged by *assumptions*, explicit and implicit (Lehman, 2000). These assumptions fill in the gaps between the system and the documented and validated requirements of the operational domain.
- *Reference Document(s):* Links to the documentation further describing the requirements.

In the *General Problem Description cluster*, the attributes *ProblemID, ProblemTitle, ProblemDescription, Problem Type, Internal/External Problem, Related to Problem(s),* and *Reference Documents* have their correspondences in *Requirements ID, Requirements Title, Requirements*

Description, Requirements Type, Internal/External Requirement and *Related Requirements,* respectively. Except for *Problem Type* and *Requirement Type*, these attributes connote the same meaning. While *Requirement Type* implies either functional or non-functional requirement, *Problem Type* refers to a specific type of a problem, such as design problem, problem with manuals, and the like.

In addition to the above-mentioned information, one needs to record information that is specific to problem management. This concerns the following attributes:

- *Problem Effect(s) and Consequence(s):* Description of the effects and consequences of the problem. This information is pivotal for assessing the severity and priority of a software problem. Problems implying severe consequences should be attended to as soon as possible.
- *Problem Symptom(s):* A description of an observed manifestation of a problem. Compared to a consequence of a software problem, a symptom may merely indicate a minor system misbehaviour. It does not always directly lead to system failure. This information may greatly help maintenance engineers understand the software problem and localise its cause.
- *Problem Conditions:* Descriptions of the conditions under which a software problem has been encountered. This information must be specified, if deemed relevant. Otherwise, the maintainer will not be able to reproduce the problem.
- *Problem Reproducibility:* a clear description of how to get a software program into a particular erroneous state. It specifies a series of steps that can be taken to make the problem occur. This greatly facilitates the problem investigation process.
- *Alternative Execution Paths:* An identification of all the paths leading to the reproduction of the software problem. This information is pivotal for understanding and resolving the problem.

Requirement/Problem Evaluation Data

The *Requirement Evaluation Data* cluster describes the data essential for evaluating and prioritizing the requirements. It covers the following attributes:

- *Business Value*: Business value is defined for the purpose of meeting some business objectives by implementing the requirement. It is used for prioritizing the requirements.
- *Other Value*: Other values may be specified. Among them are the values of stepping into a new market, attracting new customers, and other opportunities.
- *Requirement Priority*: Evaluation of the urgency of implementing the requirement. Usually, the budget does not allow the companies to implement all requirements. Hence, one needs to prioritize them. The higher the priority, the more urgent it is to implement the requirement.
- *Fit Criterion/Criteria*: A fit criterion describes a condition that a software product must fulfill in order to meet the requirement (Sampayo do Prado Leite 2009). Its purpose is to provide a contextual information so that the requirement will be testable.
- *Risk(s):* Identification of risks related to the requirement. Requirements risks may have major impacts on the success of software projects (Appukkutty et.al, 2005). They may drown the software projects, if they are not properly managed,

In the *Problem Evaluation Data* cluster, the value of *Problem Priority* and *Risk(s)* connotes the same meaning as *Requirements Priority* and *Risk(s)* in the *Requireents Evaluation* cluster. Regarding the attributes such as *Business Value, Acceptance Criteria* and *Fit Criteria,* they are not relevant in the context of problem management. In addition, a new value is added. It is *Problem Severity* measuring the effect of the disruption caused by a software problem.

Other Description Data

The *Other Description Data* cluster provides the context of the requirement and the problem. It covers the attributes identifying the system(s) and its(their) environment and the like. It includes the following attributes:

- *System Data*: To avoid confusion where the requirement/problem must be implemented/resolved, one needs to identify the system, subsystem and component. It is especially imperative in cases when the organizations manage several products with similar functionality.
- *Interfacing System ID*: Identification of the adjacent systems that are or may be impacted by the requirement/problem at hand.
- *Environment*: Specification of the environment in which the requirement/problem will be implemented/resolved. They concern hardware, software, and data environments in which the requirement must function.

4.3 Process Documentation

In this section, we describe the clusters used for documenting the system. Just because the information describing the management of new requirements and software problems does not differ much, we describe and explain the *SRMT* and *PMT* together.

Requirement/Problem Reporting Data

The *Requirement/Problem Reporting Data* cluster records when and by whom the requirement or software problem has been identified and to whom it has been assigned (Kajko-Mattsson, 2001). It covers the following attributes:

- *Requirement/Problem Reporting Date*: The date when the requirement or problem was stated/reported. This date is used for determining the age of a requirement or software problem. In the context of a requirement, a high age is an indicator that the requirement must be revisited so that it does not imply risks to the project. In the context of a software problem, a high age indicates that the software organization has probably neglected its resolution.

- *Requirement/Problem Originator***:** The originator of the requirement or problem must be identified. This information is needed for tracking and clarification purposes.

- *Reported By*: Name of the role who reported on the requirement or problem. This individual may be some engineer who reported on the requirement or problem on the *Requirement/Problem Originator's* account.

- *Requirement/Problem Owner*: Role or group of roles (team) responsible for managing the requirement or solving the software problem. The owner makes decisions on the requirement implementation or problem resolution throughout the whole implementation/resolution process. Usually, the owner is the role who originally entered the requirement.
- *Date required*: Date when the requirement must be implemented or the software problem must be resolved.

Requirement/Problem Management Data

The *Requirement Management Data* cluster communicates information about the requirement or problem management process. It covers both planned and actual actions taken to implement the requirement or to resolve the problem, identifies the roles involved in these actions, records the effort required for implementing the requirement or resolvning the problem, and the effectiveness of the implementation activities (Higgins, 2002). The cluster covers the following attributes:

- *Implementation Plan*: The preliminary outline of the activities to be taken to implement the requirement or to resolve the problem.
- *Planned and actual activities*: The activities and their estimated/actual effort and cost. It covers the following information:
  - *Activity Description*: Identification and description of the activity.
  - *Activity Start Date*: Date when the activity started.
  - *Activity End Date*: Date when the activity ended.
  - *Expected/Actual Result*: Description of the expected/actual results of the activity.
  - *Activity Conducted By*: Name of the role responsible for performing the activity.
  - *Activity Approved By*: Name of the role who approved the activity and its results.
  - *Effort Spent on Activity*: Estimated/actual effort spend on the activity.
  - *Cost of Activity*: Estimated/actual cost of the activity.

Requirement/Problem Management Progress

The Requirement Management Progress cluster tracks the status of the requirement implementation or problem resolution process. This status is essential for monitoring and controlling the requirement or problem. It records the status value, the date when the requirement/problem changed status values, the overall requirement implementation or problem resolution progress, and the requirement/problem age. The following attributes are suggested for descring the progress:

- *Requirement/Problem Management Status*: Status value indicating the progress of implementing the requirement or resolving the problem.
- *Requirement/Problem Management Status Date*: Date when the requirement/problem stepped into the particular status state.

- *Requirement/Problem Age*: Time period elapsed from the date when the requirement/problem was recognized and reported. This value is used for assuring that high priority requirements/problems get attended to as soon as possible.
- *Requirement/Problem Change(s)*: Link to change requests concerning the requirement/problem at hand.

Requirements Completion Data

The *Requirement Completion Data* cluster covers information about the completion of the requirement implementation or problem resolution process. It records planned and actual completion date, roles involved in approving and signing off the completion, and the total effort spent on requirement implementation or problem resolution. The cluster includes the following attributes*:*

- *Planned/Actual Completion Date*: Date when the requirement/problem was or was planned to be completed and tested.
- *Relation to Test(s):* Identification of tests to be used for testing the requirement or problem solution.
- *Released In*: Identification of the release(s) in which the requirement/problem was implemented/resolved.
- *Requirement Completion Approved By*: Name of the role who approved the requirement implementation or problem resolution. Usually, it is the owner.
- *Sign Off Date*: Date when the requirement/problem was signed off by the organizational authority.
- *Signed Off By*: Identification of the roles involved in signing off the requirement/problem completion.
- *Estimated/Actual Total Effort*: Total effort spent/to be spent on implementing/resolving the requirement/problem.
- *Estimated/Actual Total Cost*: Total cost spent/to be spent on implementing/resolving the requirement/problem.

Post-Implementation Data

The *Post Implementation Data* cluster holds the information about the post-mortem analysis of the requirement implementation or problem resolution process. The analysis results should provide an important feedback for improving the future requirements management or problem resolution. The attributes belonging to this cluster are the following:

- *Analysis of the Requirement/Problem Implementation/Resolution Process*: Evaluation of the process used for implementing the requirement.
- *Lessons Learned:* List of experiences as encountered during the implementation/ resolution of the requirement/problem.

5. Final Conclusions

In this chapter, we have presented two templates: the *SRMT* template used for communicating software requirements within development and evolution and the *PMT*

template used for communication software problems within corrective maintenance. We then evaluated these two templates within more than 20 software organizations. Due to space restrictions, we cannot describe the evaluation results. However, we cordially invite our readers to study (Kajko-Mattson, 2000; Kajko-Mattson, 2001; Kajko-Mattson, 2002; Kajko-Mattson, 2005; Kajko-Mattson & Nyfjord, 2008; Kajko-Mattson, 2009).

The attributes as suggested in both templates are highly relevant both within heavyweight and lightweight software development. Many of them however, are not explicitly documented. They may however be communicated in an oral form. The rigidity of the documentation is dependent on the type of software systems being developed or maintained. In cases when the software system is not a safety critical or business critical and it is not expected to live a long life, one may compromise on the amount and scope of the information to be documented in favor of oral communication. In case of critical systems, one should not compromise on even one single attribute in both the *SRMT* and *PMT* template. For instance, the engineers within *IT Exchange* claim that they would not be able to survive for long without the information that is listed in the two templates.

So far, very little research has been done on documentation. To our knowledge, there are only a few publications reporting on this subject. Hence, we claim that this domain is strongly under-explored. More research is required for agreeing upon the scope and extent of documentation so that the short-term and long-term benefits may be gained in both the heavyweight and lightweight contexts.

6. References

ABB. (2009) ABB Group Sweden, http://www.abb.com/secrc, Accessed in July 2009.

Atlantic Systems Guild. (2007). Volare Requirements Specification Template, available at: http://www.systemsguild.com/GuildSite/Robs/Template.html, Accessed in December 2007.

Antoniol, G.; Canfora, G.; Casazza, G.; De Lucia. A,. (2000). Information Retrieval models for Recovering Traceability links between code and documentation, *Proceedings of IEEE International Conference on Software Maintenance*, pp. 40-49.

Appukkutty, K.; Ammar, H.H.; Popstajanova, K.G. (2005). Software requirement risk assessment using UML, *Proceedings of International Conference on Computer Systems and Applications*, p. 112.

Arthur, L.J. (1995). Software Evolution: The Software Maintenance Challenge, John Wiley & Sons.

Bauer, B.J. & Parnas, D.L. (1995). Applying mathematical software documentation: An Experience Report, *Proceedings of 10th Annual Conference on Computer Assurance,* pp. 273-284.

Beck K. (2004). *Extreme Programming Explained: Embrace Change,* 2nd Edition. Upper Sadle River, NJ, Addison-Wesley.

Boehm, B.W. (1981). *Software Engineering Economics*, Prentice-Hall.

Briand, L.C. (2003). Software Documentation: How Much is Enough?, *Proceedings of IEEE International Conference on Software Maintenance and Reengineering*, pp. 13-15.

Card, D.; McGarry, F.; Page, G. (1987). Evaluating Software Engineering Technologies, *Journal of IEEE Transactions on Software Engineering*, Vol. 13, No. 7, pp. 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, *Proceedings of National Computer Conference,* AFIPS Press, Reston, Virginia, Volume 54, pp. 508-513.

Clark, P.G.; Lobsitz, R.M.; Shields, J.D. (1989). Documenting the Evolution of an Information System, *Proceedings of IEEE National Aerospace and Electronic Conference,* pp. 1819-1826.

Cohn M. (2006). *Agile Estimating and Planning*, Pearson Education, Upper Saddle River, NJ.

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

Cook, C. & Visconti, M. (1994), Documentation Is Important, *Journal of CrossTalk*, Vol. 7, No. 11, pp. 26-30.

Delanghe, S. (2000), Using Learning Styles in Software Documentation, *Proceedings of IEEE, Transactions of Professional Communication*, pp. 201-205.

Higgins S.A et. al, (2002). Managing Product Requirements for Medical IT Products. *Proceedings of Joint International Conference on Requirements Engineering*, pp 341-349.

Hofmann, P. (1998). Away with Words! How to create Wordless Documentation, *Proceedings of IEEE International Professional Communication Conference*, pp. 437-438.

Holt, P.O. (1993). System Documentation and System Design: A Good Reason for Designing the Manual First, *Proceedings of IEEE Colloquium on Issues in Computer Support for Documentation and Manuals*, pp. 1/1-1/3.

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

Kajko-Mattsson, M.; Forssander, S.; Andersson, G. (2000). Software Problem Reporting and Resolution Process at ABB Robotics AB: State of Practice, *Journal of Software Maintenance and Evolution, Research and Practice*, Vol. 12, No. 5, pp. 255-285.

Kajko-Mattsson, M. (2001). *Corrective Maintenance Maturity Model: Problem Management*, (2001). PhD thesis, ISBN Nr 91-7265-311-6, ISSN 1101-8526, ISRN SU-KTH/DSV/R--01/15, Department of Computer and Systems Sciences (DSV), Stockholm University and Royal Institute of Technology.

Kajko-Mattsson M. (2002). Evaluating CM^3: Problem Management, (2002). *Lecture Notes in Computer Science, Proceedings of Conference on Software Advanced Information Systems Engineering*, Springer-Verlag, Volume 2348, pp. 436-451.

Kajko-Mattsson, M. (2005). A Survey of Documentation Practice within Corrective Maintenance, *Journal of Empirical Software Engineering Journal*, Kluwer, Volume 10, Issue 1, January, pp. 31 – 55.

Kajko-Mattsson M. & Nyfjord, J., (2008). A Template for Communicating Information about Requirements and their Realization, *Proceedings of IAENG International Conference on Software Engineering*, BrownWalker Press: Boca Raton, USA.

Kajko-Mattsson, M. (2009). Status of Requirements Management in Six Chinese Software Companies, *Proceedings of International Conference on Industrial Engineering, IAENG International Conference on Software Engineering*, BrownWalker Press: Boca Raton, USA.

Kantner, L., et.al., (1997). The Best of Both Worlds: Combining Usability Testing and Documentation Projects, *Proceedings of IEEE International Professional Communication Conference*, pp. 355-363.

Kantner, L., et.al., (2002). Structured Heuristic Evaluation of Online Documentation, *Proceedings of IEEE International Professional Communication Conference*, pp. 331-342.

Lepasaar, M.; Varkoi, T.; Jaakkola, H. (2001). Documentation as a Software Process Capability Indicator, *Proceedings of IEEE International Conference on Management of Engineering and Technology*, Vol. 1, p. 436.

Malcolm, A. (2001). Writing for the Disadvanteged Reader, *Proceedings of IEEE International Conference on Professional Communication*, pp. 95-100.

Nawrocki, J.; Jasinski, M.; Walter, B.; Wojciechowski, A. (2002). Extreme Programming Modified: Embrace Requirements Engineering Practicies, (2002). *Proceedings of IEEE Joint International Conference on Requirements Engineering (RE'02),* pp. 303-310.

Norman, R.L & Holloran, R.W. (1991). How to simplify the structure of administrative procedures to make them easier to write, review, produce, and use, *Proceedings of International Conference on Professional Communication*, Vol.2 pp. 447 – 450.

Parnas, D. L. & Clements, P.C. (1993). A Rational Design Process: How and Why to Fake it, *Journal of IEEE Transactions on Software Engineering,* Vol. SE-12, No. 2.

Parnas, D. L. (1995). Software Aging, *Proceedings of 16th International Conference on Software Engineering*, pp. 279-287.

Parnas, D. L. (2000). Requirements documentation: why a formal basis is essential, *Proceedings of IEEE 4th International Conference on Requirements Engineering*, pp. 81-82.

Pigoski, TM. (1997). *Practical Software Maintenance*, John Wiley & Sons.

Ramsay, J. (1997). Corporate Downsizing: Opportunity for a New Partnership between Engineers and Technical Writers, *Proceedings of IEEE International Professional Communication Conference*, pp. 399-403.

Sampayo do Prado Leite J.C. & Doorn, J.H. (2009). *Perspectives on Software Requirements*, Kluwer Academic Publishers.

Saunders, P.M. (1989). Communication, Semiotics and the Mediating Role of the Technical Writer, *Proceedings of IEEE International Professional Communication Conference*, pp. 102-105.

Sousa, M. & Mendes Moreira, H. (1998). A survey of the software maintenance process. *Proceedings of IEEE International Conference on Software Maintenance*, pp. 265– 272.

van Schouwen, A.J.; Parnas, D.L.; Madey, J. (1993). Documentation of Requirements for Computer Systems, *Proceedings of IEEE International Symposium on Requirements Engineering*, pp. 198-207.

Visconti, M. & 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. & Cook, C.R. (2002). An overview of Industrial Software Documentation Practice, *Proceedings of 12th IEEE International Conference of the Chilean Computer Science Society*, pp. 179-186.

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

Wedde, K. J.; Stalhane, T.; Nordbo, I.; (1995). A Case Study of a Maintenance Support System, Proceedings of IEEE International Conference on Software Maintenance, pp. 32-41.