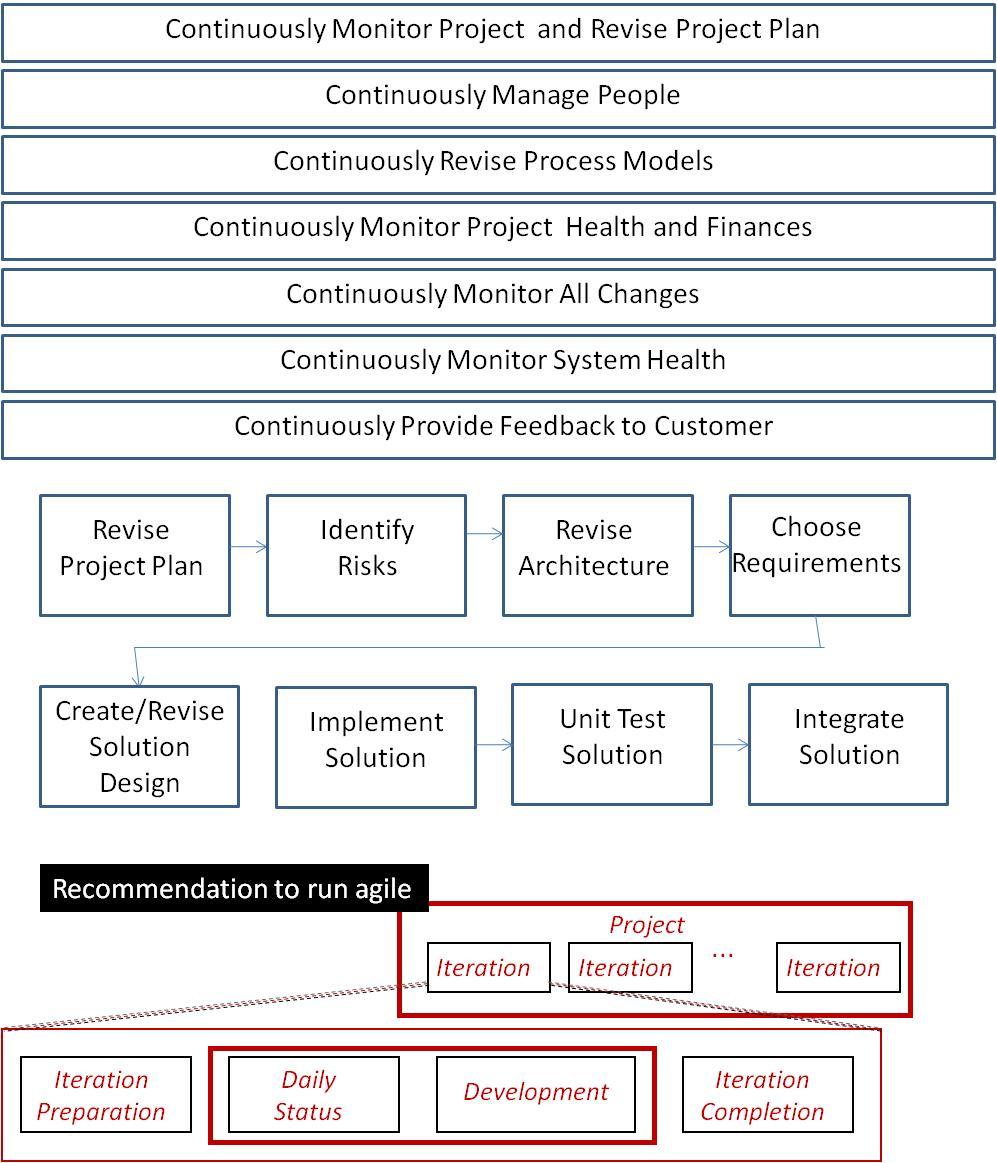
# Assignment 5

# Implementation

## Deadline: See separate specification

## Roles involved

* Process Owner: Mrs. Mira Kajko-Mattsson
* Industrial advisor: Mr.Stefan Britts
* Process Executors
  + Practitioners: IV 1300 Students



## Process

At last you may start implementing. What a long journey! Maybe this will help you understand that in many companies implementation corresponds to 5-10% of the overall development cost. Also, do you remember what Mira has said during one lecture? The productivity in some software companies is 1 LOC per day. Remember! Does it sound strange now or does it not?

Ok, let us start the implementation phase now. During this phase, you will mainly concentrate on implementing requirements. Therefore, many of you who do not have a developer role will be assigned to be developers by your respective project managers. However, you, in your particular role, will have to make sure that your process models work as you expect them to work. For this reason, you will have to continuously revise your process models. Below, I suggest the steps for the *Implementation* phase. These steps, however, may or may not occur in sequence or in parallel.

1. Revise Project Plan
   1. You as a project manager, will have to revise your project plans, put some more details into them such as for instance scheduling, task assignment to specific individuals and the like. This activity is not a one-off activity. You will have to continuously revise your plans. For each new revision, you will have to create new versions of your plans so that you may follow the changes in the project.
   2. If your company decides to follow an agile development approach, you may then get assisted with planning by your developers. See Mira’s lecture on Agile methods and get ideas for how to involve the whole team in project and iteration planning.
2. Continuously Manage People
   1. During one of the lectures, we have discussed the people issue. People are the most important asset within software organizations. However, if mismanaged, they may become counter-productive. You, as a project manager, will have to continuously monitor the atmosphere within the projects and look for problems. Use interactive-oriented people for spotting problems and ease the project atmosphere.
3. Continuously revise your process models
   1. Your process models will never be optimal. Now, when you have stepped into the implementation phase, you will have the opportunity to monitor your process models and evaluate whether they are good enough for developing the system. You, as a representative of each role together with the process manager, will have to continuously revise your respective process models. In cases when major changes need to be done, you will have to submit them first for inspection and then to the Advisory Board for decision making. All changes and motivations behind them must be documented. Example of a major change might be that you have decided to use agile methods to perform the Implementation phase. I recommend that you introduce retrospectives at the end of each iteration phase (see lecture slides on agile methods).
4. Identify risks
   1. All projects have risks. You will have them as well. Therefore, we advise you to identify project risks, analyze them, choose a representative subset of them and create action plans for managing them. This activity must be conducted as one of the first steps during the *Implementation* phase. It will then run continuously during the whole project. See Mira’s suggestion for Risk Identification Game (RIG) in Appendix A for how to identify and commonly agree upon risks.
5. Continously monitor project health and finances
   1. You, in your role as a project manager, will have to continously monitor the project. You look for problems, identify new risks, monitor the already identified risks and report them to product owner or other management. One of ways to monitor the project health is daily meeting. Consider it as one of the ways of identifying risk symptoms.
   2. You, in your role of an architect and designer, will have to continuously monitor whether the architecture and design are followed, and if not, take appropriate measures.
   3. You, as a measurement manager, will have to measure the project and continuously report on the measument results.
   4. You, as a marketing and finance manager, will have to check whether the project exceeds the planned budget, and if so, then take approprite measures.
   5. You as a business manager and analyst will have to be receptive to all customer input and take actions, if necessary. For instance, customer might have requested changes to some requirements. In such a case, you will have to involve most of the roles within your company to first determine whether it is possible to make such a change, to identify its impact, to then determine the price of the change, negotiate it with the customer and so on. Also, you will have to be receptive to all kinds of feedback you receive from projects. For instance, one may suddenly discover that some of the requirements cannot be implemented. In such a case, you will have to discuss this with the customer and commonly agree upon solutions.
   6. You, as a tester, will have to continuously check and revise your test plans and test cases. Especially important is it to monitor all the changes to the requirements. This automatically implies changes to the test plans and test cases. A good recommendation is to introduce continous integration testing and to conduct parts of system and acceptance tests at the end of each iteration.
   7. You, as a documentation manager, must make sure that everybody documents what is requested from them to document. If changes need to be made to the documentation requirements, then you have to take appropriate measures. For instance, you may get feedback from other roles that the templates designed by you miss some important information.
   8. You, as a configuration manager, make sure that everbody follows version and configuration management rules. In case of problems, you will have to take appropriate measures. For instance, you might discover the rule that all the changes to documents and code should be assigned to new versions is not optimal from the productivity perspective. Therefore, you may, for instance, decide that some documents will be assigned new versions as soon as they have passed some major milestone.
6. Continuosly monitor system health
   1. You, as a quality manager, should continuously check the quality of the system. Please remember that a software system does not only include code but also all the documents. Are they readable and well structured? Is there any traceability among requirements, design, code and test cases?
7. Implement
   1. You, as a developer, start implementing the requirements. You may do it solo or you may do it together with another developer. What is important for you to know is that for each of your tasks, you will have to create a solution design, create test cases and write code. While implementing, you will discover that the orginal requirements are ambiguous, incomplete, difficult to understand and or other. To remedy the problem, you may have to discuss it with other roles and find solutions or you may have to, via a project manager, report on the problem to the business manager, who will have to discuss it with the customer. You may also discover missing requirements. In this case, you will have to report it to the project manager who together with other roles will take appropriate measures. Finally, you have to test all the code you have developed. All test cases you use and the testing results should be documented.

**Please remember that all types of changes must be managed via a change/problem reporting process. All of them must be documented. Nothing should be communicated orally! Also, all changes should be documented.**

1. Integrate
   1. You, as a tester and developer continuously integrate the ready made components into the overall system. You may start doing it as soon as you have something to integrate (please see continous integration in Mira’s lecture on agile methods). Do not forget to write and follow integration test cases. When integrating the components, look for interface problems. All problems should be formally reported and their solutions documented. See suggestion for reporing on problems in Appendix B.
2. Continuously provide feedack to your customer
   1. This is normally done via business manager. However, you may invite your customer to various meetings, demonstations and the like. Of course, when providing feedback follow the rules you have specified in your contract. If you decide to follow an agile implementation approach, then I strongly recommend that you have demonstrations of your iteration results to your customer at the end of each iteration.

## Phase 5 Deliverables

The assignement will result in the following deliverables:

* *The whole system is integrated and integration tested. Deliver all integrated code.*
* *Deliver all integration test cases and test results. If you have followed the agile approach, then most of the system and acceptance tests should be performed by now. Deliver system and acceptance test cases and results as well.*
* *Deliver all system documentation.*
* *All problems and changes that have been reported, their solutions and the decisions made behind them.*
* *All the minutes of the meetings, if any.*
* *A specification of all the changes made to your processes.*
* *Effort required for the implementation phase in the BCPM.* The effort should be provided for each task and individual practitioner/student and it should compile the results for the whole company.
* *The company’s expenditures so far to be delivered by Financial Mananger.*
* *Experience gained during this phase.* Here, you list problems, good sides of the BCPM process phase, important decisions made and motivations behind the decisions.
* *All the documentation is delivered to Bilda “Phase 5 Deliverables”.*

# Appendix A Risk Identification Game (RIG)

The Risk Identifaction Game (RIG) helps you to commonly agree upon risks that you may encounter within your project. It is also a good way of making sure that everybody is involved in risk identification and that everybody makes her/his voice heard. All you need is a large sheet of paper, post-it notes and a pen. On the large sheet of paper, you will draw the *Consequence and Risk Landscape (CRL)* which should be designed according to Figure A1.

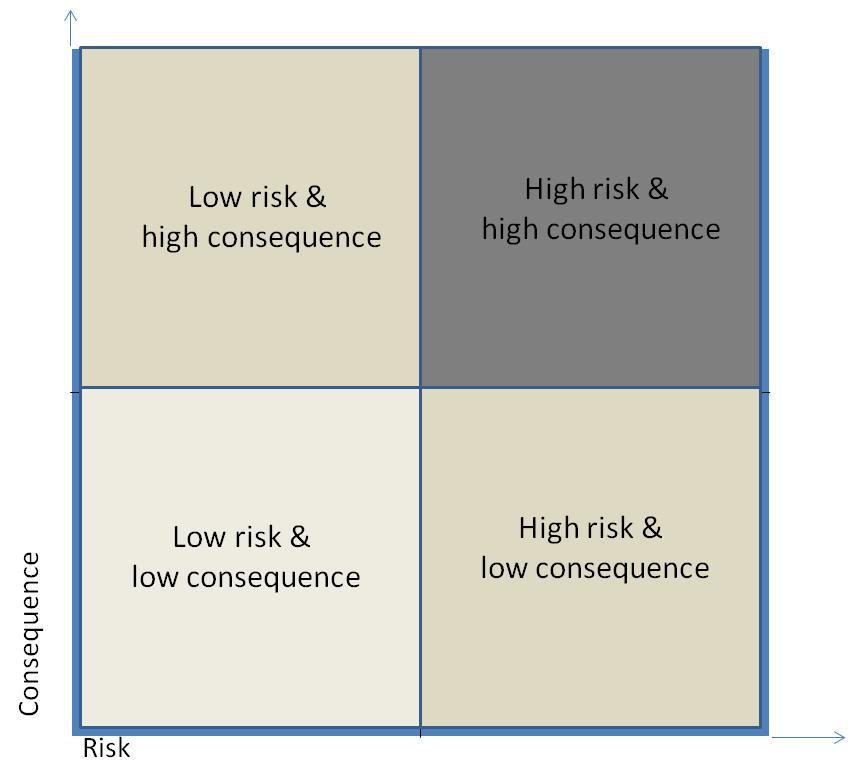


Figure 1. Layout of Consequence and Risk Landscape (CRL)

The landscape provides a platform for your brainstorming. Place it on a table and make sure that the whole team sits around it. Then, take turns in identifying risks. To make sure that no one is omitted, start with Participant 1, continue with Participant 2 and so on till you are done with the first round of risk identification. Make sure that you agree on the direction and sequence of allowing people to present their risks. You may do it clockwise or anti-clockwise. It does not matter. What matters is that no one is excluded from the game and that during one round one participant only provides one risk. If you have several risks to communicate then you wait till the next round.



Now, let us play the game. All the participants should have is a block of post-it notes and a pen. Participant 1 starts writing on a risk on a post-it note. For instance, she might suggest the following risk “*we won’t be able to implement all the requirements*”. Just write a short sentece describing the risk. Place it then on one of the quadrants. If you believe that the risk and conseqence are low then place it in the inner quadrant. The presenter of a risk, while presenting it, should say it aloud but should not spend too much time on explaining it. The other participants around the table neither should object, protest or agree with the risk. This only takes time and leads to unnecessary and unproductive discussions.

Take as many rounds around the table as you feel are needed. Usually, in the first three rounds everybody will have some unique idea about risks. In the latter rounds, some participants may still have some ideas whereas the others might have run out of ideas. Those who have run out of ideas may now move around the other participants’ risks on the *Consequence and Risk Landscape* (CRL). For instance, Participant 2 believes that Participant 1’s risk has been placed in the wrong quadruple, then she may move it to another one. While doing it, she may motivate it by providing the reason. For instance, the risk “*we won’t be able to implement all the requirements*” has been placed in the low risk and low consequence quadrant. Participant 2 believes that it should be put in the outer quadrant with the high risk and high consequence. The reason for that might be that now the team better understands the requirements and thereby they have realized that they were too ambitious.

The fact that Participant 2 has moved the risk to another quadrant does not imply that other participants should not move it to other quadrants. Anybody is allowed to move a risk whenever he or she wishes. For instance, Participant 1 may not agree with Participant 2 that it is a high risk with a high consequence. So, she or he might move it back to the inner quadrant. Of course, she or he should motivate the move.

In the latter rounds, you may simply move around the risks on the landscape till you commonly agree on where to place them. Make sure however that one participant only makes one move in one round. If some participant does no longer have any suggestion, then he should say “Pass or I give up”.

# Appendix B Requirements, Change and Problem Management

In this appendix, two templates are presented. They convey information required for describing and managing software requirements and software problems. The templates are called *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 SRMT template may be used for driving testing process as presented on one of the lectures. The text below is taken directly from Mira’s book chapter. In your project work, you do not need to use all the template fields. You may choose the ones that you believe are helpful to manage details about your requirements, changes to requirements and details about problems that have been discovered during testing.

# 5. 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.

**5.1 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 requierments 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 addtitions 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 implemention/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?

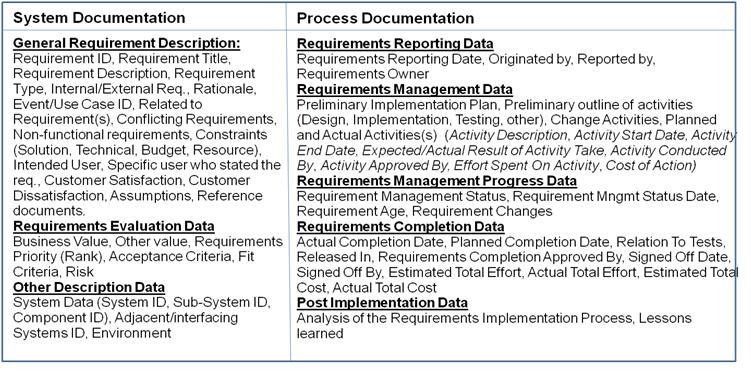


Fig.3. Software Requirement Management Template

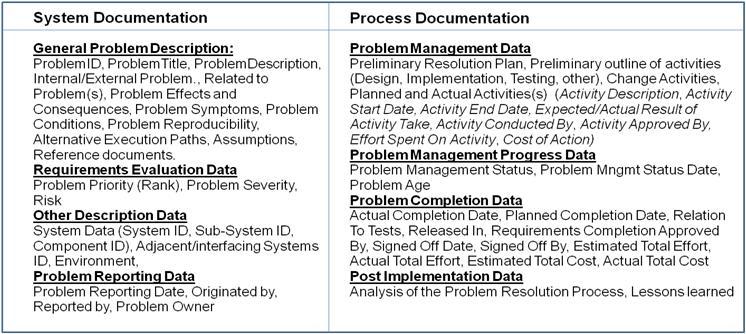


Fig.4. Problem Management Template

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 Fig. 3 and Fig. 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*.

**5.2 System Documentation**

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

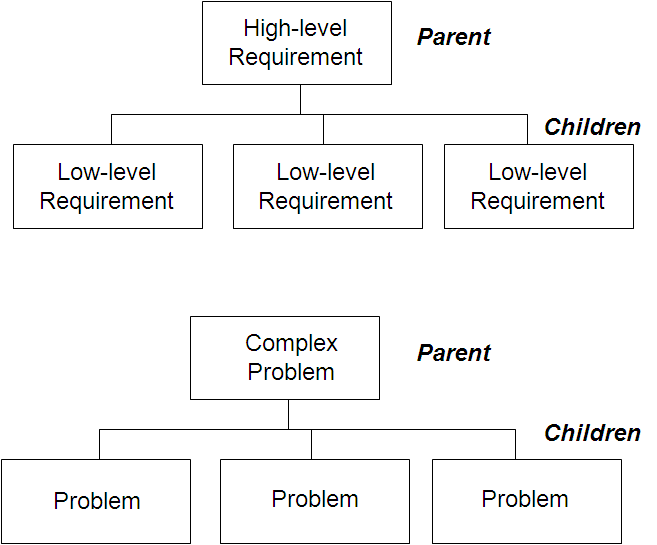


Fig.5. Hierarchies among the requirements and problems

among the requirements. One may also record the hierarchies among the major and minor requirements, like the ones presented in Fig. 5.

* *Conflicting Requirements:* Links to conflicting requrements. 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:* aclear 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 a 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 risksmay have major impacts on the success of software projects (Appukkutty et.al, 2005). Theymay 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 a disruption caused by a software problem.

**Other Description Data**

The *Other Description Data* cluster provides the context of a requirement and 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.

**5.3 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* clustercommunicates 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* tracksthe 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 a 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 completed 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/resolvning the requirement/problem.

**Post-Implementation Data**

The *Post Implementation Data* clusterholds 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/ resoltuion of the requirement/problem.