

Bidirectional Tracing of Requirements in Embedded Software Development

Barbara Draxler

Fachbereich Informatik
Universität Salzburg

Abstract

Nowadays, the increased complexity of embedded systems applications requires a systematic process of software development for embedded systems. An important difficulty in applying classical software engineering processes in this respect is ensuring that specific requirements of safety and real-time properties, which are usually posed at the beginning of the development, are met in the final implementation. This difficulty stems from the fact that requirements engineering in general is not mature in the established software development models. Nevertheless, several manufacturers have recently started to employ available requirements engineering methods for embedded software.

The aim of this report is to provide insight into manufacturer practices in this respect. The report provides an analysis of the current state-of-art of requirements engineering in embedded real-time systems with respect to requirements traceability. Actual examples of requirements traceability and requirements engineering are analyzed against current research in requirements traceability and requirements engineering.

This report outlines the principles and the problems of traceability. Furthermore, current research on new traceability methods is presented. The importance of requirements traceability in real-time systems is highlighted and related significant research issues in requirements engineering are outlined. To gain an insight in how traceability can aid process improvement, the viewpoint of CMMI towards requirements engineering and traceability is described. A short introduction in popular tracing tools that support requirements engineering and especially traceability is given. Examples of traceability in the development of real-time systems at the Austrian company AVL are presented. An analysis of flaws in requirements traceability is performed, based on these examples and on the existing research surveyed in the report.

Contents

1	Requirements Engineering	6
1.1	Importance of Requirements	6
1.1.1	Importance of Requirements in Projects	6
1.1.2	Types of Requirements	8
1.1.3	Requirements in the Lifecycle of a Project	9
1.1.4	The Problem and the Solution Domains	11
1.1.5	Requirements and Quality Issues	12
1.2	Requirements Engineering Process	13
1.3	Requirements Management	16
1.4	Requirement Documentation	17
2	Requirements Traceability	19
2.1	Definition	19
2.2	The Need for Traceability	20
2.3	Traceability Problems	22
2.4	Traceability Methods	26
2.4.1	Traceability Types	26
2.4.2	Requirement Relations	27
2.4.3	Common Traceability Methods	27
2.4.4	Traceability Methods in Research	30
3	Requirements Engineering and Embedded Systems	36
3.1	General Real-Time Systems	36
3.2	Requirements Engineering and Real-Time Systems	39
4	CMMI - Capability Maturity Model Integration	42
4.1	The history of CMMI	42
4.2	The disciplines of CMMI	43
4.3	Components of Process Areas	45
4.4	The CMMI Representations	48
4.5	CMMI Levels	50
4.5.1	Capability Levels	50
4.5.2	Maturity Levels	53

4.6	CMMI and Requirements Engineering	55
5	Requirements Engineering Tools	58
5.1	IBM Rational RequisitePro	58
5.1.1	Tracing Support in RequisitePro	63
5.2	IBM Rational ClearQuest	65
5.3	Harvest Change Manager	67
6	The Tracing Implementation at AVL	72
6.1	Requirements in AVL Projects	72
6.2	Traceability between User Requirements and Product Requirements	74
6.3	Traceability between Product Requirements and Uses Cases in the Software Requirements Specification	76
6.4	Traceability between Use Cases and Architecture Design	78
6.5	Transition of the System Model to the Source Code	79
6.6	Traceability of Change Requests	80
6.7	Analysis	81
7	Conclusion	83

List of Figures

1.1	The Waterfall Model. [25]	10
1.2	The Requirements Engineering Cycle. [12]	13
2.1	Example of a Requirements Table.	28
2.2	An Example of a Requirements List.	29
2.3	Example of Rich Tracing. [2]	30
2.4	The Model Components of TOOR. [5]	32
2.5	Contribution Structures: Traces between Artifacts and Agents. [8]	35
4.1	CMMI Process Area Components. [11]	46
4.2	CMMI Continuous Representation. [14]	51
4.3	CMMI Staged Representation. [15]	54
5.1	The information in Microsoft Word and RequisitePro is kept consistent.	59
5.2	RequisitePro provides an add-on for Microsoft Word.	60
5.3	RequisiteWeb is the web application of RequisitePro. [19]	61
5.4	A user defined view.	62
5.5	Applying of the traceability information with the property pane.	64
5.6	Traceability Matrix.	65
5.7	Traceability Tree (traced into).	66
5.8	Traceability Tree (traced out of).	67
5.9	The ClearQuest Workplace.	68
5.10	An example of the integration of RequisitePro and ClearQuest.	69
5.11	Harvest Change Manager Hierarchical Tree. [21]	70
6.1	Requirements Pyramid. [22]	74
6.2	Use Cases. [24]	77
6.3	Traceability Matrix. [24]	77
6.4	Example of a Sequence Diagram. [24]	79
6.5	Traceability Chain. [22]	80
6.6	Change Request in ClearQuest and the associated Packages in Harvest. [23]	81

List of Tables

1.1	Project Impaired Factors. [13]	7
1.2	Project Success Factors. [13]	8
3.1	Differences of Hard Real-Time Systems and Soft Real-Time Systems. [3]	37
4.1	The main differences between the staged and continuous representation. [11]	49
6.1	Example requirements from the customer requirement specification. [24]	73
6.2	Example Enhancement Request. [24]	75
6.3	Example Requirements from the Product Requirements Specification. [24]	76

Chapter 1

Requirements Engineering

The following chapter will describe the basics of requirements engineering. It will first put an emphasis on the importance of requirements and the need for requirements engineering and will later describe the common activities in the requirements engineering process. Moreover it will put a focus on requirements documentation and requirements management. The end of this chapter will describe the importance of requirements traceability, a topic which will be pictured in more detail in the later chapters.

1.1 Importance of Requirements

Requirements engineering is a methodology that is going through change right now. In the beginning of software development requirements engineering was thought to be in the early stages of the development phase and to provide a system specification. But that has proven to be a wrong assumption. Requirements evolve during the development and the software development changed because products are to be delivered faster and faster, there is no more time for a deep and full requirements analysis.

Requirements are the basis for every project, if there were no requirements there would be no goal to work for. Requirements are therefore the driving force of the development. This section will define what requirements are, and will provide the context in which requirements can occur.

1.1.1 Importance of Requirements in Projects

The lack of good requirements engineering can lead to failure in software projects, therefore requirements engineering should not be underestimated and should be well executed.

Project Impaired Factors	% of Responses
1. Incomplete Requirements	13.1%
2. Lack of User Involvement	12.4%
3. Lack of Resources	10.6%
4. Unrealistic Expectations	9.9%
5. Lack of Executive Support	9.3%
6. Changing Requirements & Specifications	8.7%
7. Lack of Planning	8.1%
8. Didn't Need It Any Longer	7.5%
9. Lack of IT Management	6.2%
10. Technology Illiteracy	4.3%
Other	9.9%

Table 1.1: Project Impaired Factors. [13]

In 1994 the Standish Group published "The Chaos Report" [13] and presented in that paper ten risk factors for project failures (Table 1.1) and ten reasons why projects succeed (Table 1.2). The risk factors and the reasons for project success were gained through surveys of companies. To gain results the Standish Group surveyed IT executive managers for their opinions about why projects succeed. The percentage of responses in the tables represent the amount of responses on reasons that lead to failure or success. Some of the main problems concerning requirements are:

- Requirements may be hardly organized and no requirements management may be available.
- Requirements may be poorly expressed and are not available in a well structured manner.
- Moreover, requirements can be weakly related, which will lead to problems that will be described and solved in later chapters.
- Requirements may change too quickly and the change impact of the requirements and its affected artifacts cannot be estimated.
- Moreover, unrealistic expectations are another major problem.

All these problems can be addressed at low cost by a systematic requirements engineering process.

Project success factors are not the inverse of project failures. As seen in Table 1.2, good planning and management support are regarded as important for a successful project.

Project Success Factors	% of Responses
1. User Involvement	15.9%
2. Executive Management Support	13.9%
3. Clear Statement of Requirements	13.0%
4. Proper Planning	9.6%
5. Realistic Expectations	8.2%
6. Smaller Project Milestones	7.7%
7. Competent Staff	7.2%
8. Ownership	5.2%
9. Clear Vision & Objectives	2.9%
10. Hard-Working, Focused Staff	2.4%
Other	13.9%

Table 1.2: Project Success Factors. [13]

Further surveys confirm the importance of requirements as a reason for project success and project failure. The BULL report [17] states that one of the main criteria of project failures was "the inability to meet project requirements". Another survey quoted in [17] was the survey of KPMG Canada which concluded that weak definitions of requirements in the planning phase can also contribute to project failure.

1.1.2 Types of Requirements

Requirements can exist in various types: the stakeholder requirements, also known as the user requirements and the system requirements. Moreover, there exists the distinction between functional and non-functional requirements. This section will describe these types and their origin.

- **Stakeholder Requirements**

Stakeholder requirements, also called user requirements are requirements written from the point-of-view of system stakeholders. System stakeholders are usually people who are affected by the system, such as end-users, managers, engineers, customers and more. It is common that their requirements are not formal and very often they are not detailed. They consist of natural language, informal diagrams and use notations that are appropriate to describe the problem.

Requirements Example 1 (RE_1): *An example for a stakeholder requirement may be that an image manipulation tool shall support different filetypes, like .jpeg or .png*

- **System Requirements**

System Requirements are more detailed requirements specifications.

They are usually derived from the informal user requirements. These requirements may provide an abstract model of the system. This can be achieved by using mathematical models, graphical notations and other models.

RE₂: Examples for a system requirement may be, following the example of RE₁ the stakeholder requirements, how different filetypes can be supported and which compression algorithms are needed.

RE₃: Another example following RE₁ may be the system specifications which are needed to run an image manipulation tool.

An even further distinction of requirements can be into functional and non-functional requirements. These two types can appear in both, stakeholder requirements and system requirements.

- **Functional Requirements**

Functional requirements describe what the system should do.

RE₄: A functional requirement for an image manipulation maybe that images of banknotes cannot be manipulated nor opened in the tool. This is done to avoid the production of false money.

- **Non-Functional Requirements**

Non-functional requirements place constraints on how the functional requirements are going to be implemented.

RE₅: An example for a non-functional requirement, based on RE₄ may be that the tool shall check the image for an occurrence of banknotes within a given time interval.

1.1.3 Requirements in the Lifecycle of a Project

Development models like the waterfall model (Figure 1.1) of software engineering have had a major impact and have influenced the software development for a long time. In this special model, the requirements engineering is done at the beginning of a project. This led to the widespread assumption that requirements engineering is done before the software development is started. Another wrong assumption was that requirements would not change significantly during the development process. An impact of this assumption was the belief that requirements engineering should be separated from system design.

However, one of the typical activities in the end of the development is the acceptance testing. In this activity the system is tested against the stakeholder requirements. This is a proof that requirements are still in use even in the end of the development.

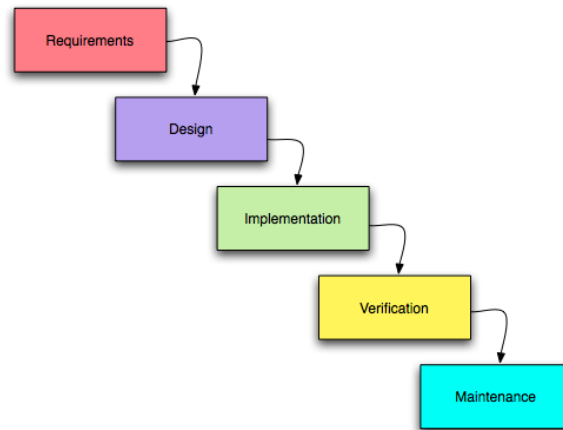


Figure 1.1: The Waterfall Model. [25]

Due to the change of the demands in software development, requirements engineering's position in the lifecycle needs to be rethought [12]. Requirements engineering is not something that can be finished in the early stages of a project, since requirements will change during the development, as the environment changes and the view of the stakeholders evolve. The role of requirements engineering needs to be re-assessed, mainly due to four driving points. These four points reflect the increased change rate of the current economic situations.

1. New approaches of systems development are being used, e.g., construction by configuration emerged, and approaches based on reuse become more popular.
2. There is a need for faster software delivery. The competition is getting harder, and only the fastest developers will get the chance for getting a project
3. The requirements change rate is increasing. This point is a consequence of the second point, as the fast delivery makes it impossible to have enough time for an in-depth requirements engineering process. Moreover, due to the fast change of the environment new requirements may appear whereas other become obsolete.
4. An improved return on investment (ROI) on software assets is needed. This is especially relevant for companies that want to get as much as possible in return for their investment.

Requirements are not limited to be the driving force of the development of just one software project. Requirements can also act as a means of communication between projects in organizations. Usually, organizations wish to maximize the reuse of artifacts of any kind between projects. They want to manage the families of similar products and they want to use the management to coordinate activities. Moreover, they want to optimize the process by learning from the experience of other projects.

As requirements play a central role in the development and are changed as the development progresses, they need to be maintained. Requirement changes throughout every phase of the project and need to be taken into account, therefore requirements engineering is strongly connected with change management. Requirement change is inevitable, because the environment in which the software is going to be used also changes continually. Changes may arise during the design, due to evolved stakeholder views or problems during development. It does not matter where the changes come from, their impact on quality, cost and schedule needs to be assessed. The outcome of these assessments leads to the acceptance or rejection of a change. Moreover, it forms the basis of negotiation of the change costs with the stakeholders, and helps to organize the redevelopment work.

Requirements engineering is a process that will change over the next few years. The old position of requirements engineering in the beginning of the development has proven to be not helpful, as changes occur during the development. Furthermore, new demands have appeared, which will lead to the fact that the process of requirements engineering and system development will have to be integrated.

1.1.4 The Problem and the Solution Domains

From the management and engineering point of view,, a clear distinction between problem domain and solution domain is preferred in requirements engineering. The early steps in the development should be associated with the most abstract levels of system description. For example, the stakeholder requirements are in the problem domain. Further development steps, like the definition and refinement of system requirements take place in the solution domain.

It is important to make a clear distinction between these two domains. The problem domain should define the problem, no more and no less. Moreover, reference to other solutions should be avoided. This separation gives more freedom to engineers.

If this distinction of the domains is not fulfilled, problems in the later development can arise. The lack of a clear distinction may lead to the facts,

that the real problem is not sufficient understood, that the system is not able to be scoped and it cannot be evaluated which functions it shall include. In the worst case, it may happen that a wrong solution is developed. Also, if the system is described only by means of solutions, it may lead to quarrels about the system between developers and users. It is therefore important to store the problems and the solution in different documents.

RE₆: An example for a good separation of the domains is following: Imagine a traffic junction system. Stakeholder shall describe the problem by what they want to achieve with the use of the system. Therefore they may desire to maximize traffic flow while minimizing accidents at a traffic junction. The system engineers may consider numerous solutions, like bridges, traffic lights or roundabouts. The system engineers then decide which solution proves to be the best solution, and then the design phase starts.

1.1.5 Requirements and Quality Issues

When dealing with requirements, one also has to define what makes up the quality of a requirement. The main question is how good quality requirements can be achieved.

Poor executed requirements engineering will affect the completion of the project. Bad managed requirements, poorly expressed requirements, requirements which are not properly organized and requirements which do not express what the stakeholders want are all factors which cause incomplete requirements. All these factors lead to project failures, missed deadlines and cost explosions.

A requirement that is fit for purpose has a high quality. Furthermore, a requirement can be of good quality if it is well specified and satisfies all the stakeholder needs. If development is based on a complete and consistent set of requirements, it will be less prone to difficulties.

Moreover, good requirements engineering is an addition to requirements management and aids to complete typical management tasks. It aids in the estimation of costs, the duration of development, schedules and provides a focus on the delivery of quality. As every management decision is one between cost, schedule and quality, good requirements are the best source to aid the management process.

Requirements of high quality can be achieved with a proper and intensive executed requirement engineering process, with well defined requirements documents and a good understanding of the system that is going to be developed. Moreover, with the aid of requirements management and with the metrics it provides, like impact analysis, derivation analysis and coverage analysis, the requirements can be checked for completeness and consistency. Only these

steps can provide a good basis for a successful completion of the development of a system.

1.2 Requirements Engineering Process

Before a system can be developed, developers and users need to gain an understanding of what the system should do and how it can support the goals of the users. The requirements engineering process is needed to gain this understanding. It is important to get to know the constraints of the system, the place where it will be used, the functional requirements and non-functional requirements, like performance, safety and dependency.

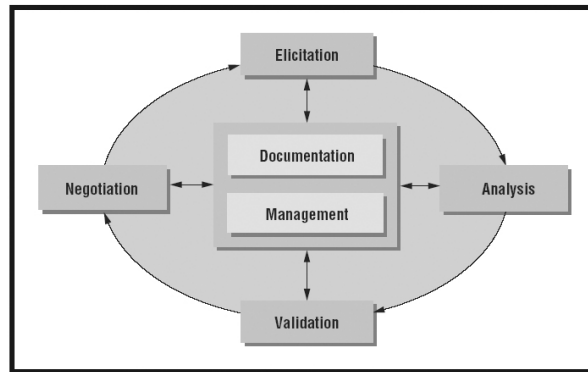


Figure 1.2: The Requirements Engineering Cycle. [12]

The requirements engineering process depends on the system that has to be developed and on the companies that develop the system. Safety critical, real time and military systems, and their developers will surely rely on a formal requirements engineering process, to cover the requirements fully and adequately. Moreover, the user and system requirements of these systems will be heavily documented to cover the full impact and function of the requirements.

On the contrary, small companies that develop innovative products do not have formal requirements engineering stages, and rely on a process that consists of brainstorming sessions. Moreover the user requirements are rather short, and may include a short vision statement of the supposed functionality of the system.

However, there are six fundamental activities that are common to all requirements engineering processes:

- Elicitation
- Analysis
- Validation
- Negotiation
- Documentation
- Management

The first four main activities will be covered in this section, requirements documentation and requirements management will be covered each in their own section, since they are very important, and are mainly management activities which support the other four main activities.

According to [2] and [12], it is a common mistake is to believe that the activities occur in sequence, where one starts with requirements elicitation and goes from one to the next activity and ends with a documented set of requirements which are then going to be implemented and which are adapted as changes occur.

On the contrary, requirements engineering is a cyclic iterative activity (Figure 1.2). The single activities are repeated as the system requirements are identified. Furthermore, the iteration continues during the development and is even active during system operation.

The outcome of the requirements engineering process is a requirements document and a statement of the requirements that describes and defines what should be implemented. It has been argued by the requirements engineering community that the more the requirements documentation is completed and consistent, the more likely the software is going to be developed and delivered in time. This argumentation goes hand in hand with the results of the Standish Group [13], which detected inconsistent and incomplete requirements specifications and requirements documents as a main factor for project failures.

Completeness and consistency of requirements can be achieved and analyzed by using various techniques such as special-purpose requirements specification languages, structured modeling and formal mathematical specification. However, in practice requirements are not complete and consistent. Moreover, requirements are usually written in natural language and consist of vague descriptions of desired functions, rather than detailed description. This approach might be correct in situations where requirements change very quickly, because the costs of maintaining detailed specifications are big. However, not defining requirements consistently may lead to difficulties between the stakeholders and the developers, because no clear statement was produced, and

developers may develop the wrong product, as both parties could not gain a complete and consistent understanding of the system.

The four main activities of the common requirements engineering processes are presented in sequel. As mentioned above, the requirements engineering process is cyclic and iterative. Each activity has associated methods that help to fulfill its role.

The result of each activity is documented; each decision and document has to be managed. Therefore, requirements documentation and requirements management are activities that are vital for the requirements engineering process.

Requirements Elicitation

The requirements elicitation needs good domain knowledge and good interpersonal skills, since this activity is executed together with the stakeholders of the systems. The requirements are discovered through consultation with stakeholders, through reading system documents, market studies, the systems environment, feasibility studies, market analyzes, business plan and analysis of competing products.

The main activity is to identify the requirements from these sources. The goal of this activity is to identify actors, scenarios, use cases and also the refining of the use cases. Moreover, relationships between use cases are discovered, participation objects are identified and non-functional requirements are documented.

Some techniques which aid this activity are interpersonal contact between developers and stakeholders, the developing of scenarios, the creation of prototypes and more.

Requirements Analysis

The requirements analysis discovers and solves conflicts between requirements. This aids in understanding requirements because here every activity is analyzed. The result of this analysis may be requirement problems like missing requirement, requirement conflicts, ambiguous requirements, overlapping requirements and unrealistic requirements. A requirement conflict occurs when two requirements negate each other. Ambiguous requirements are requirements that are not detailed enough. Overlapping requirements appear when at least two requirements have similar or equal meanings. Moreover, with the aid of the analysis the boundary of the system is detected. Detecting the system boundary means that it can be distinguished between the functions a system should provide and which functions should not be provided. Requirements analysis leads to a requirements classification and to a conceptual design of the system.

Requirements Validation

This activity is performed again together with the stakeholders to validate that the requirements which have been detected and analyzed are correct, consistent and complete. The developers check with the stakeholders if the requirements state what they need, and if both parties gain a common understanding.

Some techniques that aid requirements validation are prototyping, model validation and acceptance tests.

Requirements Negotiation

This activity resolves conflicts between viewpoints of stakeholders and viewpoints of developers. It relies on the output of the requirements analysis. Requirements might conflict, therefore the aim of requirements negotiation is to generate a consistent set of requirements. This can be achieved by meetings between stakeholders and developers. A formal negotiation process can be very useful in this activity.

These four activities are repeated until all requirements are consistent, complete and are accepted by the stakeholders. The result of these activities are requirements specifications which are represented through requirement documents and which can be understood by both the developers and the stakeholders.

A successful requirements specification can be supported easily with these activities, if they are used during the whole lifetime of a project. Since stakeholders requirements often change during a project, the requirement engineering process is needed to capture these changes, and to decide if the changes are feasible and useful for the development of the system.

Only a well executed requirements engineering process can guarantee the success of a project. The process takes market analysis and feasibility studies in account and is therefore the basis for a decision if the development of the system is useful. Moreover, a complete requirements specification may lead to a calculation of development time and may help in achieving this time.

1.3 Requirements Management

Requirements management keeps control of requirements changes and assesses the impact of the changes on the project. This consists of assessing the costs a change may cause, or the time the change may take. Therefore, change management is an important aspect of requirements management.

Like any other management activity, requirements management needs a development plan. However, as reality shows, it is hard to estimate the time

and effort to complete an activity, unless the people who are managing have enough experience. Moreover, during the development, difficulties in the project can arise as the work progresses. These influences deviate the process from the plan and may cause changes of the management plan. Furthermore, projects are constrained by three factors, namely the product capability, the cost and the timescale of the project. The goal of management is to generate a plan which works according to these factors, to control that the works progress according to the plan, and to change and adapt the plan when deviations occur.

Requirements management is more difficult than common management. Specific problems can arise while trying to establish a requirements management plan, as follows:

1. Only few people have extensive experience of requirements management. This lack of experience makes it hard to make estimations of the costs and timescale of a project. Moreover, it might not be known in advance which activities are necessary in order to fulfill a successful requirements engineering process and a development process.
2. People do not distinguish between stakeholder requirements and system requirements. Due to the lack of this distinction the design specification may aim in its requirements specification for a solution, rather than a description of what the system should do, and how it should be implemented based on this description.
3. Different companies manage requirements in different ways. A company that only acts as a supplier will have a different way to manage requirements than a company that develops products.
4. It is difficult to know if a requirement set is complete, so that the activities can stop. Moreover, it is much more difficult to monitor the progress when the generation of requirements has started and is nowhere near completion. And last, the quality of the generated requirements also has to be assessed, and it may be difficult to determine the uniqueness and necessity of all requirements.
5. It is not straightforward to deal with changes of requirements. Therefore, requirements management should have a great focus on change management. As one requirement change may also affect other requirements, its consequences have to be assessed.

1.4 Requirement Documentation

The requirement documentation consists of:

1. An official statement of the system requirements for customers, end-users and developers. This requirements document is the product of a requirement engineering process. The requirements document may have different synonyms like 'software requirements specification' or 'requirements specification'.
2. Requirements documents, delivered by stakeholders, which are usually very informal and are written in natural language. Most requirements documents consist of natural language sentences which may be supplemented with diagrams and tables of detailed information.
3. Requirements documents written by developers are usually much more detailed and can be more formal than the requirements documents from the stakeholders.

Requirements documents should have a layout specified by the company that is going to develop the system. A common requirements document structure leads to exchangeability and reusability. Developers which are used to a special document structure can easily read new requirements documents with the same structure. Moreover, a standard format can act as a checklist for document writers, and the chances that information is omitted can be reduced. Reviewers can use the format to control if sections have been left out.

Common known standards have been developed by the US Department of Defense and by IEEE. The most accessible of these standards is the IEEE/ANSI 830-1993 standard. This standard suggests the following layout:

- The document begins with an introduction, where the purpose of the document and the scope of the product are covered.
- The general description gives an overview of the characteristics of the product.
- The next chapter should cover the functional, non-functional and interface requirements.
- An appendix and an index are demanded.

This standard is defined in a generic format, therefore companies can adapt the standard according to their needs.

Chapter 2

Requirements Traceability

Requirements traceability is a main methodology that aids the requirements management process. Requirements traceability is helpful especially in the area of change management, particularly when evaluating the impact of changes. Requirements tracing makes it possible to follow the evolution steps of a requirement until it is implemented. Moreover, it also works the inverse way. For example it is possible to trace a design model back to the root requirements, on which the design model is based.

Furthermore, requirements traceability helps in evaluating the costs of changes, and makes it possible to evaluate the completeness of a product. Tracing also helps to validate and test if requirements have been implemented, and if they have been implemented correctly.

This chapter will first summarize the common definitions of requirements traceability and will then describe the need for traceability. Following that, problems that are encountered while using requirements traceability and problems pointed out by current research will be described. Concluding the chapter, current tracing methods will be presented, and a detailed description of current tracing research will be given.

2.1 Definition

Requirements Traceability refers to the ability to trace requirements forward and backward through the development cycle. An often cited definition of requirements traceability is:

”A software requirements specification is traceable if (i) the origin of each of its requirements is clear and if (ii) it facilitates the referencing of each requirement in future development or enhancement documentation” (ANSI/IEEE Standard 830-1984)

In current research, this definition has proven to be not sufficient to capture the viewpoint of all people involved in a software engineering process

and does not meet the different characteristics of requirements.

The research reported in [6] expanded the definition of the ANSI/IEEE Standard as follows:

”Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases).”

This definition puts more emphasis on the lifecycle of a requirement. Moreover, two types of requirements specification (RS) traceability are distinguished:

”Pre-RS traceability, which is concerned with those aspects of a requirement’s life prior to its inclusion in the RS (requirement production).”

”Post-RS traceability, which is concerned with those aspects of a requirement’s life that result from its inclusion in the RS (requirement deployment).”

The need for the separation of these two types was discussed in [6]. They discovered that the problems that are encountered while enforcing traceability are usually encountered in Pre-Requirements specification traceability.

Another research [5] focused on the natural occurrence of traces and their impact on requirements; therefore their definition is as follows:

”Requirements Traceability refers to the ability to define, capture and follow the traces left by requirements on other elements of the software development environment and the traces left by those elements on requirements.”

Clearly, this definition puts more emphasis on the influence of the requirements on other elements, therefore also on the lifecycle of the requirement itself.

2.2 The Need for Traceability

Traceability of requirements is a methodology by which high level requirements are linked to low level requirements, and furthermore are linked to design models and to the code. Requirements traceability aids both sides of

a project: the stakeholders and the developers. The stakeholders can verify if and how their wishes and objectives have been implemented and realized. The developers can verify how the stakeholder requirements are met by the system requirements and how the system requirements are realized in subsystems and components.

Traceability can contribute in various ways to the development of a product. The activities which can benefit from traceability are described below:

Verification

Requirements tracing helps to verify if all software requirements have been evolved to design, code and test cases. It verifies that all user needs have been implemented and adequately tested. Moreover, verification ensures that each function can be traced to a requirement. With this activity it is also ensured that all requirements have associated design components and test cases.

Cost Reduction

The more complete are the requirements allocated in the beginning of the development, the easier it is to keep track of them in the later development. If no tracing is established, and requirements and their implementation are altered in the last stages of a project, the consistency will not be able to keep up, which will lead to a cost explosion. This cost can be decreased by keeping track of the requirements with the aid of traceability, and with the aid of change management.

Accountability

If trace data is available, a higher success rate of audits is possible. Moreover, if traceability is used, it can be proven that requirements have been validated by their associated test cases. Traceability gives more insight if milestones have been reached, therefore it gives greater confidence in meeting objectives. This satisfies the stakeholders and improves their confidence. Furthermore, it can be assessed how suppliers attribute to the whole project. Therefore, their role can be assessed and the connections and roles become more visible.

Change Management

Traceability supports change management, as it can provide an analysis of the impact a change can cause. If tracing is established, it can be easily determined which design elements and associated requirements are affected.

Moreover, the documentation can be kept up to date. Managers can identify the test cases that will have to be rerun, and make the change consistent.

Progress Tracking

An important activity during the development phase is to keep track of the development progress. With established traceability, components and source code can be related to requirements, the progress can be measured, and costs can be assessed. Traceability can aid the development also in the form of traceability analysis. The traceability analysis aids the requirements engineering process, especially the change management and the analysis of the impact of a change. Three common forms of traceability analysis are:

- **Impact Analysis**

The impact analysis aids the change management process. It is used to determine the development artifacts that will be affected by one artifact changes. Moreover, it can be assessed which costs will be associated with a certain type of change.

- **Derivation Analysis**

The derivation analysis is more or less a cost and benefit analysis. It works in the opposite direction of the impact analysis. A low level artifact is chosen, and with the aid of the traces, it is determined which high level requirement initiated its implementation. Especially elements in design which have no corresponding high level requirement increase the cost without benefit.

- **Coverage Analysis**

The coverage analysis aids the development process and also supports the management reporting. The coverage analysis is used to determine if all requirements have been covered and correctly implemented. The analysis is performed by determining if all requirements trace to lower layers and to test plans. If a trace is not available, then either the requirement is not implemented or not tested.

The coverage analysis can also be used to determine the progress of the development.

2.3 Traceability Problems

When adding requirements traceability to a requirements engineering process, one is faced with a series of problems. These refer to the lack of a common definition of requirements traceability, the information which should be traceable, the tracing method, and the correct establishment and usage

of requirements traceability. These problems have been subject of intensive research.

The paper [6] states as a big problem the definition of requirements traceability. It is illustrated that the definitions of traceability found in literature differ in emphasis and have a limited scope. The authors of [6] noticed that requirements traceability can be viewed as purpose-driven, solution-driven, information-driven and direction-driven. They explain that requirements traceability cannot be coherently and consistently provided if the definitions differ by much. They also note that this has negative impacts on the development and use of tools to aid requirements traceability, since individuals have all their own definition of requirements traceability.

They also suggest that requirements traceability consists of two basic types, the Pre-requirements specification traceability and the Post-requirements specification traceability. Pre-RS traceability covers the life of requirements before the inclusion in the requirements specification, and deals therefore with the production of requirements. It depends on the ability to trace requirements back and from their originating statements through the process of requirements production and refinement.

Post-RS traceability covers the development of requirements after they have been included into the requirements specification and therefore deals with requirements deployment. It depends on the ability to trace requirements back and from a baseline through the artifacts in which they are distributed.

This separation is suggested because many common problems of requirements traceability occur due to the lack of distinction between the Pre-RS and the Post-RS types of traceability. As mentioned in the definition above, both types depend on different emphasis of requirements tracing. It is argued that existing support mainly provide post-RS traceability, which can be captured in a formal way.

The authors of [6] see a need for improved pre-RS traceability. They believe that practitioners need techniques to record and trace the information that is related to requirements specification production and revision. They identified the conflicting problems and needs of the two main parties involved in requirements production: the developers and stakeholders. The problems involving the developers are the following:

- The developers regard pre-RS traceability usually with low priority. Very often the priority of pre-RS traceability even lessens when the specification has been signed off. It is also seen as an optional extra.
- Moreover, there is no management and allocation of the different roles that are needed to obtain and document the required information, to organise it and to maintain it.

- There is also an imbalance of the work that is involved to establish pre-RS traceability and the benefits that are gained.
- The developers are also orienting their tracing needs on the visible needs of the stakeholders, which is usually tracing of the post-RS traceability.
- Another problem is that the documentation of required information is no guarantee that the information is traceable, as there is no guarantee that the structured and traceable information is up to date.
- The developers may not be able to obtain all information, and if the information is obtained, its quality may vary.

The problems related to stakeholders are the following:

- There exists no stereotypical stakeholder, their requirements differ and are inconsistent. This makes it hard to determine a set of requirements that can be useful in the development.
- The inability to define the way how access to information and its further developed representation will be managed.
- The production of requirements depends on personal contact, because there is always something out of date, inaccessible or not usable.

The authors of [6] view the challenge in satisfying both Pre-RS and Post-RS traceability. The stakeholders need traceability that is sensitive to contextual needs and the developers have to identify and document the relevant information.

Another paper [5] deals with the informal and formal aspects of requirements tracing. Its authors state that the informal aspect is more dominant in the initial phases of software development. Following problems were identified with the informal aspects:

- Registering of the relevant information from the informal aspects to define the adequate reference is difficult. The question is which information is relevant? The reference is set by the methods and techniques used by the people who develop systems. These methods and techniques might determine to a great extent the kind of information which is needed for the further development.
- The automation of some activities can complicate traceability rather than help it.

- A given tracing model might not cover all traces correctly. One point is that the model regards some relations as important, whereas the people in charge of registering relations do not. It is not possible to cover all necessary relations for the future traceability needs. This may be due to the fact that the information that should be traceable is very often informal. The transformation of this information to some formal structure may not fulfill the traceability needs.

In [5], the current support of requirements traceability is regarded to be not sufficient. The existing tools do not provide full support. Still, traceability might be achieved due to personal contact. It is stated that the traceability techniques are restricted and therefore the current support of traceability is limited to a great extent to the extraction of already traceable information.

Other problems might arise while establishing a requirement management process. The book [1] points out various guidelines for process improvement. It also deals with problems that might arise while establishing these guidelines.

The problems cover the way requirements are identified and how to apply a unique identifier. Furthermore, establishing traceability policies might need months to be introduced, established and performed. The main problem is that doubts about the value of tracing occur, especially when people have already had experience with traceability problems.

The book contains a pragmatic view onto the way and the need of requirements tracing. However, it contains a restricted view on requirements tracing. It gives aid in how to maintain requirements traceability, but does not deal with problems which occur when trying to create traces to very informal requirements.

Recapitulating, problems in tracing usually occur when capturing requirements. Requirements which are already specified can be formalized and can be traced easily and automated. However, problems typically appear during the process of requirements development, the informal part of requirements tracing, where a lot of change happens and a lot of people are involved. Furthermore, it is difficult to match the views of the providers and the end-users, to identify the relevant information and to create requirements which satisfy the end-user and which are understandable by the provider. Another problem is the low priority of tracing as its impacts might not be visible at the beginning.

These problems need to be solved to achieve consistent, efficient and usable requirements traceability models.

2.4 Traceability Methods

This section deals with common tracing methods. It gives an overview of the information that can be recorded with tracing, an overview of various kinds of traces that can be covered. It also provides explanation of the most common known tracing methods and gives a summary of current research on tracing methods.

2.4.1 Traceability Types

Tracing of too much information can lead to big costs and also to misunderstanding rather than being helpful. Therefore, the needed tracing information should be chosen according to the developers and stakeholder needs.

The most common traceability types are described below:

Requirements-sources traceability

This covers the traces which link the requirement to the people or the documents which have created and have specified the requirement.

Requirements-rationale traceability

This covers the links which create a trace to the motivation behind the specified requirement.

Requirements-requirements traceability

This deals with the traces of the relation of requirements. If one requirement is dependent on another one, then this is a type of requirement-requirements traceability.

Requirements-architecture traceability

This type of traceability links the requirement with the subsystems where the requirement is implemented.

Requirements-design traceability

This type covers the traces which link requirements to a specific component in the subsystem which is used to implement the requirement.

Requirements-interface traceability

This type deals with the links that relate requirements with external interfaces.

2.4.2 Requirement Relations

Traceability types can lead to a well chosen traceability model. But to regard traces between requirements, a more fine-grained distinction can be made. The various relation types of requirements are enumerated in the sequel.

- specifies/is-specified by
Two requirements are in this relation if one of them adds detail to the other one.
- requires/is-required by
Two requirements are in this relation if one requirement requires the result of the other one.
- constrains/is-constrained by
Two requirements are in this relation if one requirement is constrained in some way by the other one.

2.4.3 Common Traceability Methods

This section describes several basic established traceability methods.

Tracing tables

Tracing tables may be used to follow the traces between requirements, or between requirements and design components. The requirements are listed along the horizontal and the vertical axes, the relationships between them are marked in the table cells. This method of requirement tracing can easily be implemented using spreadsheet tables or a word processor. The tables should be defined such that the names of the columns and rows are the names of the requirements. In the simplest form of marking a relationship, a simple mark like '*' can be put in the corresponding cells. For example, if requirement in row A depends on requirements in column F, a mark should be put in cells (A, F).

The table can be read and understood easily. When reading down one column, all requirements which depend on the column requirement are listed, when reading across a row all requirements on which the requirement, if this row depends are identified.

This simple method can be expanded by adding requirement relations, as mentioned above, instead of using a simple mark.

Traceability tables can easily be managed when only a small number of requirements are used. Traceability cannot be managed when hundreds and thousands of requirements have to be traced with this method. A way to deal with large numbers of requirements is to split requirements into groups and to keep requirement tables of these groups and requirement tables of the

relations between these groups. Figure 2.1 shows an example of a tracing table.

Depends-on

	A	B	C	D	E	F
A			*	*		
B					*	*
C				*	*	
D		*				
E						*
F						

Figure 2.1: Example of a Requirements Table.

Tracing Lists

Tracing lists are more compact forms of traceability tables. Along with a requirement, one or more identifiers of related requirements are kept. This view can be represented in a table with two columns. All the requirements are listed in the left column. The right column lists the requirement on which each requirement in the left column depends on.

Relationships can be identified easily in this way. For example it is easily seen on which requirements a given requirement depends on. But it is hard to see what requirements depend on given requirements.

Tracing lists are easy to manage, and probably less prone to errors than tracing table. Furthermore, they can be used with more requirements than in the case of tracing tables.

Tracing lists can for example be easily implemented via hyper-linked documents. The requirements specifications can be created as a html-file and the hyperlinks link to the requirements the requirement is related to. An example of a requirements list is given in Figure 2.2.

Requirements Database

This method is based on an existing requirements database, where the tracing information is stored with the requirement. With the aid of various SQL queries, the tracing paths can be extracted. Forwards and backwards tracing can be done easily.

Keeping a requirements database needs more costs and people who administer the database. It is easier to create tables and lists than to create databases.

Requirement	Depends-on
A	C,D
B	E,F
C	D,E
D	B
E	F
F	A

Figure 2.2: An Example of a Requirements List.

Rich Tracing

Rich tracing is an advanced tracing method, as it extends the relations between requirements. This tracing method aids to verify the implementation of a user requirement by its corresponding system requirements much easier. It adds a 'satisfaction argument' for each user requirement. This is done by putting another statement between the user requirement and the corresponding system requirement.

The system requirements are combined in the statement with a propositional operator:

- The conjunction (and) means that all system requirements which are combined with the operator are necessary for the user requirement satisfaction argument to hold.
- The disjunction (or) means that any one of the system requirements which are combined with this operator is necessary for the user requirement satisfaction to hold.

By adding these statements and operators to the traceability, it can easily be identified which requirements are necessary to satisfy a user requirement. Moreover, this method is helpful in assessing the requirements for validity and completeness; it makes the structure more precise. An example of rich tracing is given in Figure 2.3, the rectangles with rounded edges represent the propositional operators and connect the requirements, which are represented as rectangles. Furthermore, the rectangles with rounded edges represent the way how to satisfy the connected requirements.

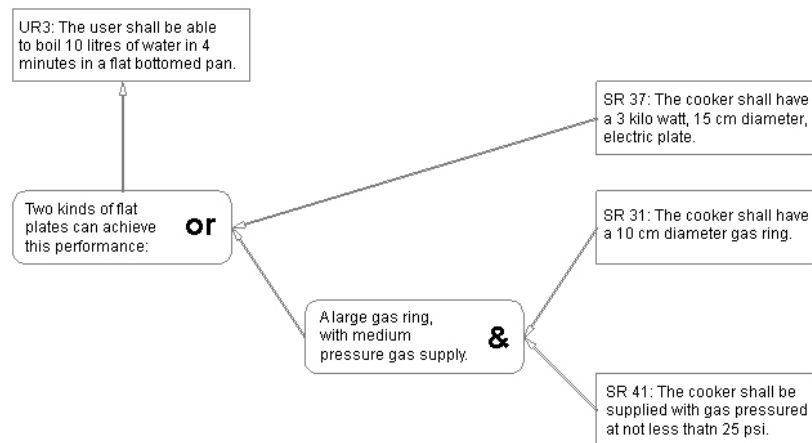


Figure 2.3: Example of Rich Tracing. [2]

2.4.4 Traceability Methods in Research

This section describes methods of tracing that are subject of current research and their associated problems. Research is done on tracing that is performed during development and tracing that is established after development. Researchers argue that improvements in requirements tracing are needed. This section describes first research of tracing after the development, then shows a formal tracing model and concludes with presenting a more in depth requirements relation research.

Scenario Driven Model

The research reported in [7] proposes a method of registering trace information after the development is finished. It generates traceability information by observing test scenarios which are executed on the running software system.

This method matches development models against the implementation and also the execution of the final piece of software. On this software, the test scenarios, which are usually defined before and during the development, are executed and the internal and external behavior of the system is observed. This leads to defining a footprint, which covers the code that is executed while running a specific test scenario. Moreover, based on these footprints, the following types of traces can be generated and validated.

1. Traces between scenarios and system

2. Traces between model elements and system
3. Traces between scenarios and model elements
4. Traces between model elements

The method consists of four activities: Hypothesizing, Atomizing, Generalizing and Refining. These activities will be explained in the sequel.

- **Hypothesizing**

This activity determines what type of traces may exist in the analyzed system. These traces are usually elicited from systems documentations or corresponding models. If no documentation is available, the hypothesizing may have to be done manually, but since the method can be iteratively used. Traces created by an iteration of the method, can be used as hypothesized traces.

The trace information created in this activity does not have to be extensive nor does the information have to be correct. Inconsistent and faulty trace hypotheses may result in contradictions during subsequent activities and therefore they are detectable.

- **Atomizing**

During atomizing the footprint graph is generated from the observable information that is gathered while running the test scenarios on the system. The aim of this graph is to illustrate the common atomic footprints of two scenarios.

The graph builds the foundation for the next two activities.

- **Generalizing**

During generalizing the trace information is propagated or generalized by traversing the footprint graph from leaves to parents.

- **Refining**

Refining is the reverse activity to generalizing. During this activity the graph is traversed from the roots to the leaves to propagate and refine the trace information to the leaves.

Traces of type 1 are usually identified with Atomizing. Generalizing, Refining and Hypothesizing generate traces of the types 2, 3 and 4.

Formal Model

This section describes a method that allows automated formal tracing, based on the abstract traceability tool TOOR presented in [5].

TOOR implements an abstract traceability model, which can be viewed as consisting of three languages: a language to define structures of related objects, a language to express relationships and a language to define module structures.

The model consists of three main parts: The trace definition, the trace production and the trace extraction.

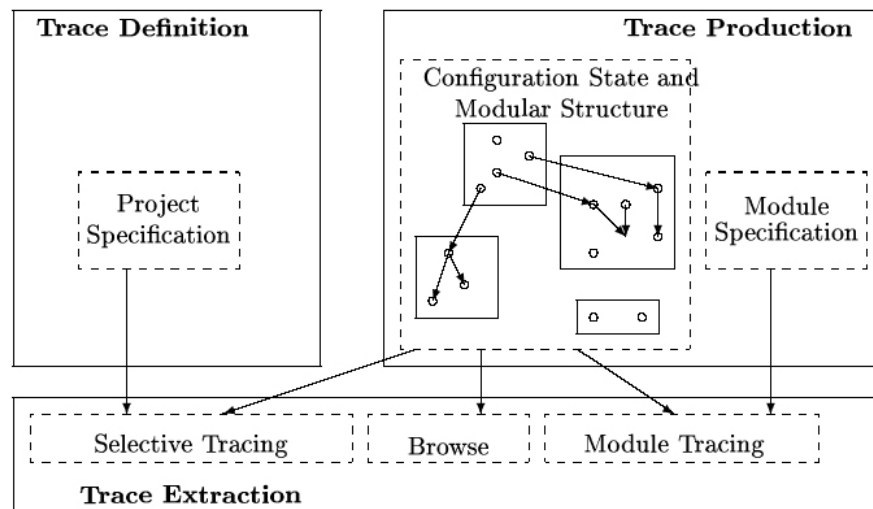


Figure 2.4: The Model Components of TOOR. [5]

Figure 2.4 display the model components of TOOR. Trace definition consists of the project specification, which has to be defined beforehand. The trace production consists of the module specification and the configuration state and modular structure. The module specification is part of the trace production to express that the project structure does not need to be specified in advance. The configuration consists of the objects, also called trace units and relations and the traces, which are registered by using TOOR. The trace extraction consists of selective tracing, browse and module tracing. Browse is used to inspect the configuration state, module tracing is used to trace according to the project structure and selective tracing is employed by using regular expressions to express the patterns of related objects.

TOOR is initiated by defining a project specification. The specification instantiates TOOR's abstract traceability model with a concrete one. The project specification is a formal specification which consists of the definitions of the objects that can be traced during the development and of the rela-

tions between traces of distinct objects. In TOOR, a project only refers to a tracing project not an actual development project. As soon as the project specification is written, the user can start a project by creating objects and relating them. Usually, the project specification evolves during project development. During development, new relations might be needed or specified relation might become obsolete.

Contribution Structures

Contribution structures are a solution of the traceability problems according to [4]. As presented in [8], contribution structures expand the traces to the people who requested requirements, therefore tracing the contribution.

This approach is limited to the pre-requirements specification traceability, as described in [4], and deals with the situations where the information about the participants in requirements production process is absent or badly managed. [3] suggests that artifact-based and personnel-based requirements traceability provides a comprehensive approach to handle requirements traceability.

Contribution structures provide an extension to the conventional requirements tracing by adding information about the persons involved in creating and specifying the requirements. Conventional requirements specifications usually have a name and author label, which is expanded as the requirements specification evolves, but this information is not traceable. Another drawback is that the author label only refers to the person who has written the specification, but not to the one who requested the requirement. The conventional requirements tracing is also known as artifact based tracing, it captures the relations between requirements, their dependencies and the evolution of the requirements. Personnel based tracing expands artifact based tracing and allows tracing requirements to the authors, the relations between the authors that created the requirement and makes it possible to view the structures of the people associated with requirements engineering. It also adds details about how and why a person was involved in the production, if the working arrangements have changed, and how the loss of a person in the development team affects the requirements knowledge. Figure 2.5 displays the artifact-based and personnel-based requirements traceability. The rectangles represent requirement artifacts and the oval shaped circles represent people. The white shaded rectangles represent the first versions of the requirement artifacts whereas the grey shaded rectangles represent the versions based on the original version. The white shaded circles represent personnel which was originally responsible for the artifact whereas grey shaded circles represent the personnel which took over the accountability from the original personnel.

The contribution structure method suggests following the three roles a person can have when being involved in the requirements production and specification process:

- **Principal**

The principal is the person who has initiated the creation of the requirement artifact. The principal has provided the information, is engaged to what it expresses and is the person responsible for the consequences the requirement artifact has.

- **Author**

The author organized the information that the requirement artifacts phrase. This person is liable for the content and the structure of the artifact.

- **Documentor**

The documentor is the person responsible to document and record the information for the requirements artifact's data.

Furthermore, the approach regards specific relations between requirements and can therefore draw conclusions about the relations of requirements to persons, and the position a person can take while creating a requirement, and the importance of a person that is involved in the requirements production process.

The approach mainly distinguishes between three categories of artifact-based relations.

- **Temporal relations**

These relations cover the chronological order in which requirement artifacts have been produced and provides a tracing history.

- **Development relations**

These relations cover the logical order in which requirement artifacts have been produced and provide requirements flow-down.

- **Auxiliary relations**

These relations add more orders into the relations between requirement artifacts and add more subtle forms of requirements traceability.

The auxiliary relations were expanded in the context of the contribution structure approach with containment and connectivity relations.

- **Containment Relations**

Containment relations are based on recording information between a composite artifact and the artifacts it contains. By maintaining this information, the contribution areas are getting more delineated as the time advances.

– **Connectivity Relations**

The contribution structure approach mainly deals with two kinds of connectivity relations, the function to reference and the function to adopt.

When the contents of two related artifacts does not overlap, then one speaks of a function to reference. When the contents of two related artifacts overlap in some way, one speaks of function to adopt.

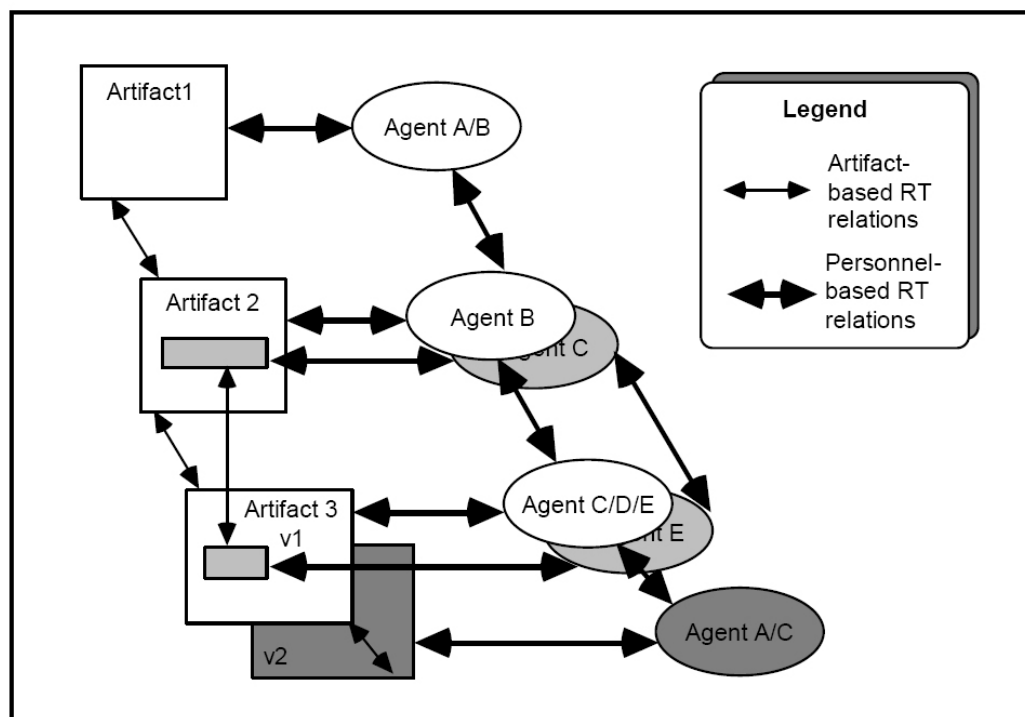


Figure 2.5: Contribution Structures: Traces between Artifacts and Agents. [8]

The contribution structure approach was implemented as a tool. Its main advantages are dealing with the missing parts of required information and providing information on the side of the people involved with the development. The approach provides the ability for process improvement. The main drawback mentioned in [8] is the people's resilience to adopt the method, due to the clearer view on accountability generated by the approach.

Chapter 3

Requirements Engineering and Embedded Systems

This chapter will describe and define embedded real-time systems. Furthermore an overview of requirements engineering techniques and additions will be given.

3.1 General Real-Time Systems

Real-time systems and especially embedded real-time systems are gaining more and more attention of the developers. However their construction is nowadays a field of intense research. Due to the fact that real-time systems need a more formal definition new rules for their development and tools aiding the development have to be created.

Real-time systems are strongly connected to embedded systems and usually they are used as synonyms for each other. This report deals with embedded real-time systems therefore an introduction into real-time systems will be given. Kopetz [3] defines real-time systems as followed:

”A real-time computer system is a computer system in which the correctness of the system behavior depends not only on the logical results of the computation, but also on the physical instant at which these results have been produced.”

Furthermore, a real-time computer system is only part of a larger system, that can be partitioned into clusters. The clusters are usually the *controlled object*, the *real-time computer system* and the *human operator*. The human operator and the controlled object are also known as the *environment* of the computer system. There exist two interfaces in this clustered system, the man-machine interface which is the interface between the human operator and the real-time computer system and the instrumentation interface, which

Characteristic	Hard Real-Time	Soft Real-Time
Response Time	Hard-required	Soft-desired
Peak-load Performance	Predictable	Degraded
Control of Pace	Environment	Computer
Safety	Often critical	Non-critical
Size of Data Files	Small/Medium	Large
Redundancy Type	Active	Checkpoint-recovery
Data Integrity	Short-term	Long-term
Error Detection	Autonomous	User assisted

Table 3.1: Differences of Hard Real-Time Systems and Soft Real-Time Systems. [3]

is the interface between the real-time computer system and the controlled object. The instrumentation interface comprises of sensors and actuators to transform physical signals into digital ones, and digital signals into physical ones.

A real-time system contains tasks. Each task reads sensors and reacts on their output. This means, that tasks calculate the stimuli and values given from the sensors. The tasks have to calculate the values in the *response time*, which is the time span between the input from the sensors and the output provided from the task. The instant at which an result has to be produced after the sensors have been read is called *deadline*. This deadline can be soft, firm or hard. It is called soft if the processed output is useful, even after the deadline has passed. A firm deadline refers to a deadline which should be met in order to calculate an useful output. A deadline is classified as hard, if missing the deadline leads to a system failure.

This classification leads to two main types of real-time systems, the soft real-time systems and hard real time systems. A general overview of the differences between hard and soft real-time systems is given in Table 3.1.

Soft Real-Time Systems

In soft real-time systems, tasks do not always need to meet the deadlines. The deadlines can be missed occasionally. However the performance of the system may suffer if a deadline is missed. The response time is often in the order of seconds. This is can be the case if the systems suffers under a peak load. This means, that the number of tasks have increased so much that the system is slowed down. In soft real-time systems the average performance of

the system is important. The pace, which is the time interval in which the system progresses, is given by the computer system. Soft real-time systems are used for non-critical tasks

Hard Real-Time Systems

Hard real-time systems are designed to meet a guaranteed temporal behavior under all specified load and fault conditions. Hard real-time systems are used for critical tasks, therefore all deadlines have to be met. This kind of real-time systems has a response time of in the order of milliseconds or less. Furthermore, the peak-load performance should never drop, which means that it has to be guaranteed by design that the deadlines are met in every situation. The pace of hard real-time systems is defined by the environment, therefore it is paced according to the state changes in the environment. Autonomous error detection should be ensured and recovery actions should be initiated within time intervals given by the system specification.

Real-time systems differ in their usage from other systems, by the fact that they are often used in areas in which a failure can lead to a threat to human life. For example a failure in a car's braking mechanism can lead to a dangerous situation for the people in the car.

Real-time systems have stern non-functional requirements, which have to be met in order for the system to fulfil the desired service.

According to [1], such systems can also be classified as RAMSS systems, which have following critical attributes:

1. Reliability: The system's behavior must be consistent and must conform to its specification.
2. Availability: The system must be available for service whenever required.
3. Maintainability: It must be possible to maintain and update the system, without necessarily taking it out of service.
4. Safety: The system's behavior must never cause damage to people or the system's environment.
5. Security: The system and its data must be protected from unauthorized access.

These attributes are not independent, as for example a system has to be secure, in order to be safe.

Due to the fact that these systems have those stern non-functional requirements, they are very expensive to develop. The specification and the development should avoid errors and the systems have to be evaluated carefully and thoroughly. Very often expensive testing has to be used for checking a real-time specification.

Another important point in developing real-time systems is that at one point, those systems might have to be certified. For example systems on airplanes need to be certified by national aviation authorities.

3.2 Requirements Engineering and Real-Time Systems

The requirements engineering of real-time systems needs to be done very carefully and with explicit formal methods. Especially requirements tracing and its associated tracing analysis are an important methodology in this respect. This section will describe points in which developing real-time systems differs from developing other types of systems.

The engineering of real-time systems adds special types of requirements into the requirements engineering process. According to Kopetz [3], the requirements can be categorized in three main groups: functional requirements, temporal requirements and dependability requirements. These three types will be described in the sequel.

- **Functional Requirements**

Functional requirements deal with the functions a real-time system has to perform. This can be for example an final state machine specification of the state of a real-time system.

- **Temporal Requirements**

Temporal requirements have their origin in control loops. For example a control loop can be a thermostat for the control of radiators. The temperature of the thermostat is measured in constant time intervals. If the temperature rises above a user defined threshold, the heaters will be deactivated. If the temperature falls below a another threshold, the heaters will be activated. Temporal requirements deal with variables that are involved in a control loop. Example variables are the sampling period or the computer delay. The impose a timing constraint onto the control loop.

- **Dependability Requirements**

This type consists of the requirements that deal with the quality of

service a system provides in a time interval. It is important to satisfy the critical attributes of RAMSS systems.

With respect to these requirements, real-time systems can be developed similar to general systems. However due to the safety-critical nature of real-time systems, improved engineering methods are topic of intensive research.

An interesting approach towards modeling safety-critical systems with aid of model-based requirements engineering is the *Feature Net* [16]. The approach addresses two needs: the systematical acquisition of functional dependencies and the necessity for focused reuse. Feature Net is a methodology to capture dependencies and to detect inconsistencies between requirements in the early stages of requirements. Moreover, it aims to simplify the change from requirements engineering to design. Furthermore it targets systematic reuse. The feature net is composed of the Feature Tree and the Semantic Net.

- **Feature Tree**

This is a domain specific classification tree. Every node in this tree represents a feature and is associated with a template. A template consists of meta-information like an identification tag, a short description and other specific information. The tree edges represent classification relations between different features. This creates a hierarchical classification of features into different classes.

- **Semantic Net**

This net depicts the dependencies between the features. It structurally collects dependencies between features. There exist different types of relations. These are for example hierarchical relations or causal relations.

The Feature Net is the result of a formalizing process of the requirements engineering because it captures all requirements that appear in the requirement engineering phase. Furthermore, the information acquired in the Feature Net forms a basis for the design.

Traceability has more importance in developing real-time systems than in developing other common systems. Requirements traceability is for example practiced by the US Department of Defence and its contractors for safety-critical systems. Moreover, real-time systems need to be verified and validated very carefully with respect to all requirements and functions.

There is no need for a special traceability method. Even presently available tools like RequisitePro can provide a traceability mechanism which suffices the demands for tracing in real-time systems. [9]

However the traceability problems that have been pointed out in Chapter

2, also apply to the development of real-time systems. Therefore new approaches should be considered for tracing requirements of real-time systems. The Feature Net [16] provides a promising research approach to this need.

Chapter 4

CMMI - Capability Maturity Model Integration

This chapter describes the basic principles of CMMI, starting from its history, its levels and models.

CMMI roots in several capability maturity models (CMM). These CMMs aid in process improvement and management. Many available CMMs only address a specific part of the business, and therefore only provide aid in this single focus. What is missing is a model which provides a systematic approach on problems most organizations are facing. Focusing only on one area can lead to barriers within the companies.

CMMI can avoid these barriers and helps to eliminate them through integrated models that traverse between the different disciplines within a company. A special emphasis is placed on the importance of CMMI within traceability.

4.1 The history of CMMI

The Software Engineering Institute (SEI) of the Carnegie Mellon University found several dimensions in which a company can focus on to improve its business. These critical dimensions are: people, procedures and methods, and tools and equipment. They are kept together by the processes used in the organization.

Since 1991 numerous CMMs have been developed, which have focused only on one of the following business areas:

- Systems engineering
- Software engineering
- Software acquisition

- Workforce management and development
- Integrated and process development.

Companies were able to use these models successfully, however problems arised when models were used concurrently in an organization. Moreover, companies wanted to focus their improvement on more than one area, but with CMMs which were not able to be integrated, this was not possible. The CMM Integration project was formed to sort out the problem of using multiple CMMs. The CMMI Product Team's mission was to combine three source models into a single improvement framework for use by organizations pursuing enterprise-wide process improvement. These three source models were the following:

1. Capability Maturity Model for Software (SW-CMM) v2.0 draft C
2. The Systems Engineering Capability Model (SECM)
3. Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98

These models were chosen, due to the fact that they were in widespread use in software and systems engineering communities and due to their approaches on process improvement in organizations. The CMMI team used those three models as their source material and developed a set of integrated models which can be used by those that already used the source model and by those that are new to CMM.

CMMI can be regarded as the successor of the three source models. The first draft of CMMI was released 1997. Since then, CMMI has gone through a lot of reviews and currently exists in version 1.1.

4.2 The disciplines of CMMI

As the main goal of CMMI is to provide a maturity model that covers development and maintenance improvement. Furthermore CMMI has the intent to provide an extensible framework to allow the integration of new disciplines. As of now, there are four disciplines supported by CMMI:

- Systems engineering
- Software engineering
- Integrated product and process development
- Supplier sourcing

These are described below:

Systems Engineering

This discipline covers development of general systems, which can, but don't have to include software. System engineers focus on transforming customers' needs, expectations and constraints into products. Moreover, they aim to support the product during its lifetime.

Software Engineering

This discipline deals with the development of software systems. The development of software systems should be done by systematic, disciplined and quantifiable approaches. Moreover, a focus should be kept on the maintenance of the software.

Integrated Product and Process Development (IPPD)

The IPPD is a systematic approach to manage a timely collaboration of relevant stakeholders throughout the life of a product to satisfy the needs of the stakeholders, their expectations and requirements. The processes that support IPPD are integrated within other processes in the organization.

If IPPD is chosen, then it has to be used concurrently with other disciplines to produce output. Therefore one or more disciplines in addition to this one have to be selected.

Supplier Sourcing (SS)

The SS can be employed when project managers use suppliers which are essential for the project to perform functions or add modifications to a product. In the case that these activities are critical, the project benefits from the improved source analysis and from the monitoring of the activities of the suppliers before they deliver the product. Therefore, SS covers the acquisition of products from suppliers.

SS is similar to IPPD in the way that SS has to be used in conjunction with best practices used to produce products.

These disciplines consist of several process areas, which are clusters of associated best practices. As of now, there are twenty-five processes areas in CMMI. Example processes areas are: measurement and analysis, requirements management and configuration management. *Process areas* consist of components. To make improvements in one are the associated best practices have to be implemented collectively.

4.3 Components of Process Areas

The components of the process areas can be grouped into several categories.

Required Components

These components define what an organization has to fulfill in order to satisfy a process area. The satisfaction of these components has to be visibly implemented. Goal satisfaction is used to decide if a process area has been reached.

Expected Components

These components define what an organization has to implement to achieve a required component. The expected components aid people that are implementing the improvements or are performing appraisals. An appraisal is an examination of one or more processes by using an appraisal reference model to determine strength and weaknesses. To consider goals as satisfied those practices or alternatives have to be planned and implemented processes of the organization.

Informative Components

These components supply details that aid an organization to get a starting point in approaching the required and expected components.

An enumeration and description of process area components follows. An overview of the components is given in Figure 4.1.

- **Purpose Statements**

The purpose statement is an informative component describing the aim of the process area.

- **Introductory Notes**

The introductory notes are also comprised as informative components. They declare the concepts which are covered in the process area.

- **Related Process Areas**

These informative components enumerate references to other related and similar process areas. Moreover they represent the relationships between the process areas.

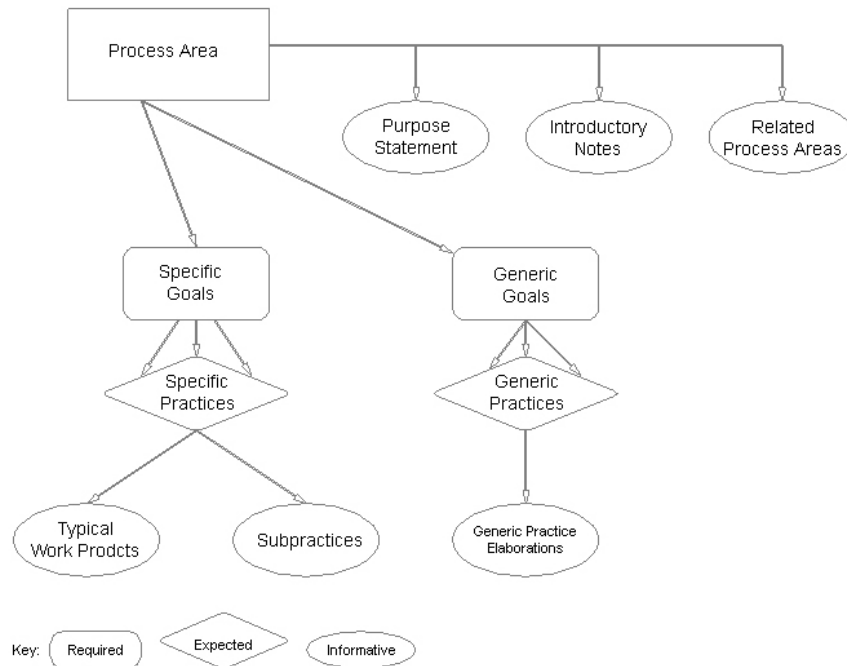


Figure 4.1: CMMI Process Area Components. [11]

- **Specific Goals**

The specific goals are part of the requirements components, and these goals depict unique characteristics that have to be met, in order to fulfil a process area. Moreover, the existence of a specific goal can be used to determine the satisfaction of a process area. Furthermore, only the statements of the specific goals are required, other information like title and notes are considered as informative components.

- **Generic Goals**

The generic goals are required components and are called generic because the same goal statement can appear in various process areas. The generic goals depict characteristics which have to be met, in order to institutionalize the process that make up the process areas. *Institutionalization* means that an organization follows routinely its way of doing business as part of its corporate culture. Generic goals help to

determine the satisfaction level of a process area.

- **Practice-to-Goal Relationship Tables**

These tables are informative components. They describe the relationships between practices which are expected components and the goals, which are required components. These relationships aid to determine when a goal has been satisfied. Furthermore, the table represents a summary of all the goals and practices.

- **Specific Practices**

These practices are expected components and they describe activities which are important to achieve an associated specific goal.

- **Typical Work Products**

The typical work products are informative components. They comprise exemplary outputs from specific practices. These samples are declared as "typical work products" because there are usually other work products that can be as effective as those that are listed in the typical work products, but are not mentioned there.

- **Subpractices**

Subpractices are informative components and aid the implementation of specific practices by giving a detailed description that will give guidance for interpreting and implementing specific practices. Subpractices are meant to supply ideas that could be helpful for the process improvement.

- **Generic Practices**

Generic practices are expected process area components and like generic goals they appear in the end of a process area. Generic practices provide the institutionalization to ensure that the improved processes will be effective. Generic practices are defined as "generic" since the same practices can appear in more than one process area. As its counterpart, the "specific practices", the generic practices provide descriptions of activities that are important for achieving an associated generic goal. Generic practices can be grouped in common features:

- Commitment to Perform (CO)
- Ability to Perform (AB)

- Directing Implementation (DI)
- Verifying Implementation (VE)

- **Generic Practice Elaborations**

Generic practice elaborations are considered as informative components. They aid the generic practice, by providing guidance in how to apply the generic practice to the process area.

4.4 The CMMI Representations

CMMI comes in two flavours: staged and continuous. These representations are rooted in the history of CMMI. CMMI is based on different models which had all different representations. The representation, which both had disadvantages and advantages, were included into CMMI. This was done, to support communities which were using the base models and to encourage them to use CMMI. An overview of the main differences between the staged and continuous representation is given in Table 4.1.

Staged Representation

The staged representation uses predefined sets of process areas to define an improvement path for organizations, which is described by maturity levels. This representation was used by the Software CMM.

The staged representation offers a systematic and structured way to improve processes within an organization. The staged representation offers a step by step improvement with given milestones and given process areas that have to be implemented to reach the next level. Implementing CMMI with the staged representation offers a good improvement, since it is visible what needs have to be fulfilled to reach the next level. The staged representation demands a given order for implementing process areas.

The process improvement with staged representation is measured with maturity levels.

Continuous Representation

The continuous representation allows the organization to choose a specific process area for improvement. The improvement is characterized with capability levels. This representation bases in the SECM and the IPD-CMM.

The continuous representation offers a flexible approach to reach a certain capability level in an organization. The continuous representation is useful

Criteria	Continuous Representation	Staged Representation
Improvement	Grants explicit freedom to select the order of improvement that best meets the organization's business objectives and mitigates the organization's areas of risk.	Enables organizations to have a predefined and proved improvement path
Capability	Enables increased visibility of the capability achieved in each individual process area	Focuses on a set of processes that provide an organization with a specific capability that is characterized by each maturity level.
Representation	Provides a capability-level rating that is used primarily for improvement in an organization and is rarely communicated externally.	Provides a maturity-level rating that is often used in internal management communication, statements external to the organization, and during acquisitions as a means to qualify bidders.
Improvement	Allows improvements to different processes to be performed at different rates.	Summarizes process-improvement results in a simpler form - a single maturity-level number.
Approach	Reflects a newer approach that does not yet have the data to demonstrate its ties to return on investment.	Builds on a relatively long history of use that includes case studies and data that demonstrate proved return on investment.
Migration	Provides an easy migration from the SECM to the CMMI.	Provides an easy migration from the Software CMM to CMMI.
ISO/IEC 15504	Affords an easy comparison of process improvement to ISO/IEC 15504 because the organization of process areas is derived from 15504.	Allows comparison to 15504, but the organization of process areas does not correspond to the organization used in ISO/IEC 15504.

Table 4.1: The main differences between the staged and continuous representation. [11]

when only one trouble spot and several of its associated areas should be improved. Moreover, it allows different processes to improve at different rates. This approach makes it possible for organizations to deal with the areas which provide most of the problems.

The process improvement is measured with the capability levels.

4.5 CMMI Levels

Both the continuous and staged representation have different levels to measure the progress of the process improvement. Within the continuous representation this is done with capability levels. The progress of the staged representation is measured with maturity levels.

This section will describe the capability levels and the maturity levels.

4.5.1 Capability Levels

Capability levels measure the process improvement of one organization in individual process areas. They measure the incremental improvements of processes which belong to a single process area. There exist six capability levels:

Level	Capability
0	Incomplete
1	Performed
2	Managed
3	Defined
4	Quantitatively Managed
5	Optimizing

The levels consist of related specific and generic practices for a chosen process area. When the specific and generic practices of a single process area at one level are satisfied the benefit of the process improvement becomes visible. Figure 4.2 gives an overview of the continuous representation.

The capability levels are maintained in a capability level profile. This is represented by a list of process areas and their associated capability levels. In this way, an organization can keep track of the capability levels. This profile can be regarded as an achievement profile as it represents the progress of the processes of the organization. It can also be seen as a target profile when it states the improvement objectives of an organization. The improvement can be tracked by comparing the two profiles.

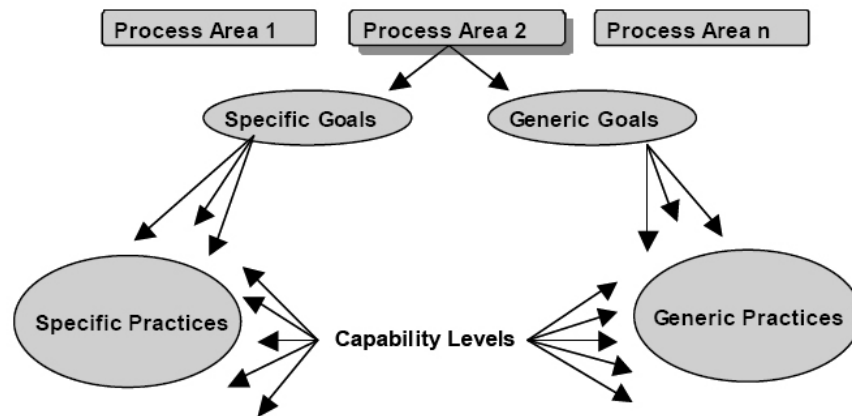


Figure 4.2: CMMI Continuous Representation. [14]

Level 0 - Incomplete

At level 0 a process is regarded as incomplete when it is either not performed or only partially performed. This means that either one or more specific goals of one process have not been met, and that no generic goal has been established.

Level 1 - Performed

A performed process satisfies the specific goals of the process area. It therefore aids in creating work products.

Level 2 - Managed

A managed process is a performed process which consists of a simple infrastructure to support the process.

The difference between a managed and a performed process is that a managed process is planned, and furthermore the performance of the managed process is evaluated in accordance with the plan. Corrections to the plan are only made when the results and the performance differ too much from the plan.

Level 3 - Defined

A defined process is a managed process that is made up from the set of the organizations' standard processes and is established according to the guidelines of the organization. These processes contribute to process assets with work products, measures and other information about process-improvement.

The standard processes which are the basis of the defined process are improved over time. They declare the process elements which are expected in the defined process. Moreover, these standard processes can describe the relationships between process elements. According to [14] a defined process has to state the following points:

- Purpose
- Inputs
- Entry criteria
- Activities
- Roles
- Measures
- Verification steps
- Outputs
- Exit criteria

Furthermore, a big difference between a managed process and a defined process is their scope of process descriptions standards and procedures. For a managed process they apply only to a project group or an organizational function. In the defined state the organizations' goal is to deploy standard processes that have to be proven as useful. This takes less time than writing and deploying new ones. Therefore these processes are consistent within the organization. Moreover, defined processes are described in more detail and are performed more rigorously than managed processes, which means that the improvement information can be understood easier, can be analyzed and used. With these improvements, an understanding of the relationships between the process activities, measures, work products and its services can be gained.

Level 4 - Quantitatively Managed

A process is quantitatively managed when it is a defined process which is controlled by using statistical and quantitative techniques. Therefore, quantitative objectives for quality and process performances are used as a criterion when managing the process. The quality of a process and its performances is expressed in statistical terms, and with the aid of these terms the process is managed through its life.

The main difference between a defined process and a quantitatively managed process is the predictability of the process performance. As the process

is quantitatively managed, statistical and other quantitative techniques are used to evaluate the performances of one ore more subprocesses whereas a defined process can only provide qualitative predictability.

Level 5 - Optimizing

An optimizing process is a process that is quantitatively managed and is improved to meet relevant, current and projected business objectives. The main goal of the optimizing process is to improve the process performance with both incremental and innovative enhancements. Moreover, process improvements which can deal with the process variations are identified. At this level, the performances of the organization's processes are continually improved. The difference between a quantitatively managed process and an optimizing process is that the optimizing process aims to continuously improve the process by addressing the reasons for process variation, whereas a quantitatively managed process only aims to address special causes for process variation and provides statistical predictability for the results.

4.5.2 Maturity Levels

Maturity levels consist of specific and generic practices for a predefined set of process areas which improve the performances of the whole organization. For each level a set of process areas is given. If this set process areas is satisfied the organization reaches the next maturity level. There are five maturity levels:

Level	Maturity
1	Initial
2	Managed
3	Defined
4	Quantitatively Managed
5	Optimizing

When a level is reached, the corresponding part of the organization's processes is stabilized, forms a base on which the next level can be achieved. The levels are measured by the satisfaction of specific and generic goals within their predefined set of process areas. The maturity levels measure organizational improvement relative to a set of process areas.

Level 2 - 5 use the same terms as the capability levels, because they are complementary. Figure 4.3 gives an overview of the staged representation.

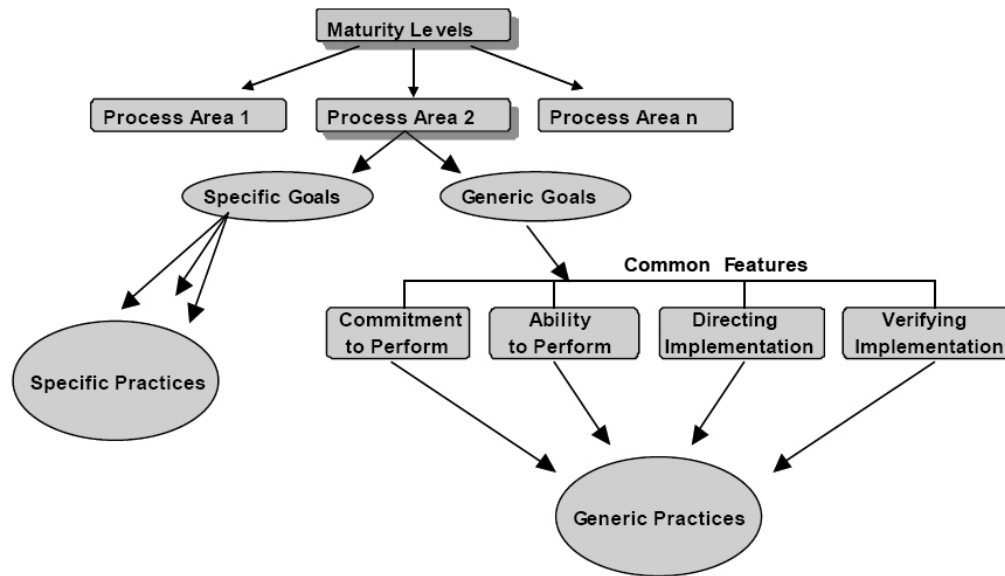


Figure 4.3: CMMI Staged Representation. [15]

Level 1 - Initial

Processes at level 1 are regarded as ad-hoc and chaotic. The organization does not support an environment that is stable enough to support the processes. If the organization has success, this may be due to the people in the organization and not due to the use of proven processes. Moreover, products and services that are offered by the organization may function, but they regularly exceed the budget and the schedules.

Furthermore, organizations at level one may abandon processes during crisis and they may not be able to repeat earlier successes.

Level 2 - Managed

An organization has level 2 when it has fulfilled the specific and generic goals of the set of process areas needed to reach level 2. This means that the process are managed, planned, performed, measured and controlled. Also the status of the work progress is visible to the management.

With the aid of the established control and planning it is ensured that practices are retained during stress. When the practices are executed the projects are performed and managed according to the documented plan.

Level 3 - Defined

An organization at this level has achieved all the specific and generic goals which are needed to reach maturity level 2 and maturity level 3. Within level 3, the processes are characterized and understood and are supported by standards, procedures, tools and methods.

The goal of maturity level 3 is to improve and establish the standard processes of the organization. With the help of these processes, an organization-wide consistence of the processes shall be established.

Level 4 - Quantitatively Managed

When an organization has reached maturity level 4, it has achieved all the specific goals of the process areas associated with maturity level 2 to 4 and the generic goals associated to maturity levels 2 and 3. At level 4 subprocesses are chosen and controlled by using statistical and other quantitative techniques. With this information, they can contribute to the overall process performance.

As in capability level 4, statistical and quantitative objectives are used to measure the process performance. With the aid of this information, the reasons for process variation can be identified.

Level 5 - Optimizing

An organization that has reached maturity level 5 has fulfilled all the specific goals of the process areas associated to maturity levels 2 - 5 and the generic goals associated with maturity level 2- 3. At maturity level 5 the main goal is the continual improvement of the process according to a quantitative understanding of the reasons for process variation. The main focus of maturity level 5 lays on improving the process performance with the aid of incremental and innovative technological improvements.

4.6 CMMI and Requirements Engineering

One of the engineering process areas in the staged representation at maturity level 2 is the process area "requirements management". This area's specific goal 1 is called "manage requirements" and the specific practices (SP) associated with this goal are the following:

- SP 1.1: Obtain an Understanding of Requirements
- SP 1.2: Obtain Commitment to Requirements
- SP 1.3: Manage Requirements Changes

- SP 1.4: Maintain Bidirectional Traceability of Requirements
- SP 1.5: Identify Inconsistencies between Project Work and Requirements

Summarizing, each specific practice deals with a specific topic in requirements management. They deal with the criteria for obtaining requirements, with assessments of the impact requirements may bring in, with the way requirements can be managed, the establishment of tracing and the detection of inconsistencies. All these specific practices ensure that the requirements are fully covered, correct and consistent.

The interesting specific practice for this report is SP 1.4 "Maintain Bidirectional Traceability of Requirements". The SP 1.4 states the following [11]:

"SP 1.4 Maintain Bidirectional Traceability of Requirements
Maintain bidirectional traceability among the requirements and the project plans and work products.

The intent of this specific practice is to maintain the bidirectional traceability of requirements for each level of product decomposition.

When the requirements are managed well, traceability can be established from the source requirement to its lower level requirements and from the lower level requirements back to their source. Such bidirectional traceability helps determine that all source requirements have been completely addressed and that all lower level requirements can be traced to a valid source. Requirements traceability can also cover the relationships to other entities such as intermediate and final work products, changes in design documentation, test plans, and work tasks. The traceability should cover both the horizontal and vertical relationships, such as across interfaces. Traceability is particularly needed in conducting the impact assessment of requirements changes on the project plans, activities, and work products.

Typical Work Products

1. Requirements traceability matrix
2. Requirements tracking system

Subpractices

1. Maintain requirements traceability to ensure that the source of lower level (derived) requirements is documented.
2. Maintain requirements traceability from a requirement to its derived requirements as well as to its allocation of functions, objects, people, processes, and work products.
3. Maintain horizontal traceability from function to function and across interfaces.
4. Generate the requirements traceability matrix.”

This specific practice demands at least a simple traceability tool. It requires horizontal tracing to be established and a traceability matrix. Consequently this method can be of disadvantage in big systems.

CMMI suggests what and how it can be done to achieve a certain specific practice. The suggestion is kept very general, which means, the way how a specific practice can be achieved is very abstract defined. When analyzing how bidirectional tracing should be maintained in SP 1.4, only simple tracing methods are suggested. However, a traceability matrix and requirements tracking system can be simple or sophisticated according to the type of the system that is going to be developed. The requirements of a small up to medium system can be traced, however this gets more difficult when dealing with safety-critical and real-time systems, have numerous and formal requirements and consist of numerous non-functional requirements. Moreover bidirectional tracing is necessary for checking functions against requirements especially within real-time systems. It is obvious that the depth of different systems cannot be covered when the main goal of CMMI is process improvement, this can be only handled within abstract definitions. The way the solutions are implemented depend on the system’s needs.

Chapter 5

Requirements Engineering Tools

This chapter describes three tools that are deployed to aid requirements engineering. The tools in this chapter were chosen because they are used by AVL.

5.1 IBM Rational RequisitePro

RequisitePro is a requirements management tool. It has been developed by Rational Software, now owned by IBM, which currently maintains and deploys it.

In this section, the basic features of RequisitePro will be described, with a strong emphasis on the tracing support, the coverage analysis and the integration with other products of the IBM Rational tool family.

RequisitePro is based on two main features: a database and Microsoft Word. The database can be chosen at start-up and can be built in Microsoft Access, but also optional databases such as the IBM DB2, Microsoft SQL Server or Oracle can be used.

Usually, companies use Word documents to keep track of their requirements, however this becomes problematic when requirements are changed and adapted. There is no overview if the requirements are already changed. RequisitePro addresses this problem, as the information in the database and in the Word documents is kept consistent. Moreover, if a requirement is created in the RequisitePro application, it will be visible in the associated Word document. If a requirement is updated in the Word document, its changes will be reflected in the database.

Requirements can be added and altered directly within the database when the changes are applied with aid of the RequisitePro application. To apply

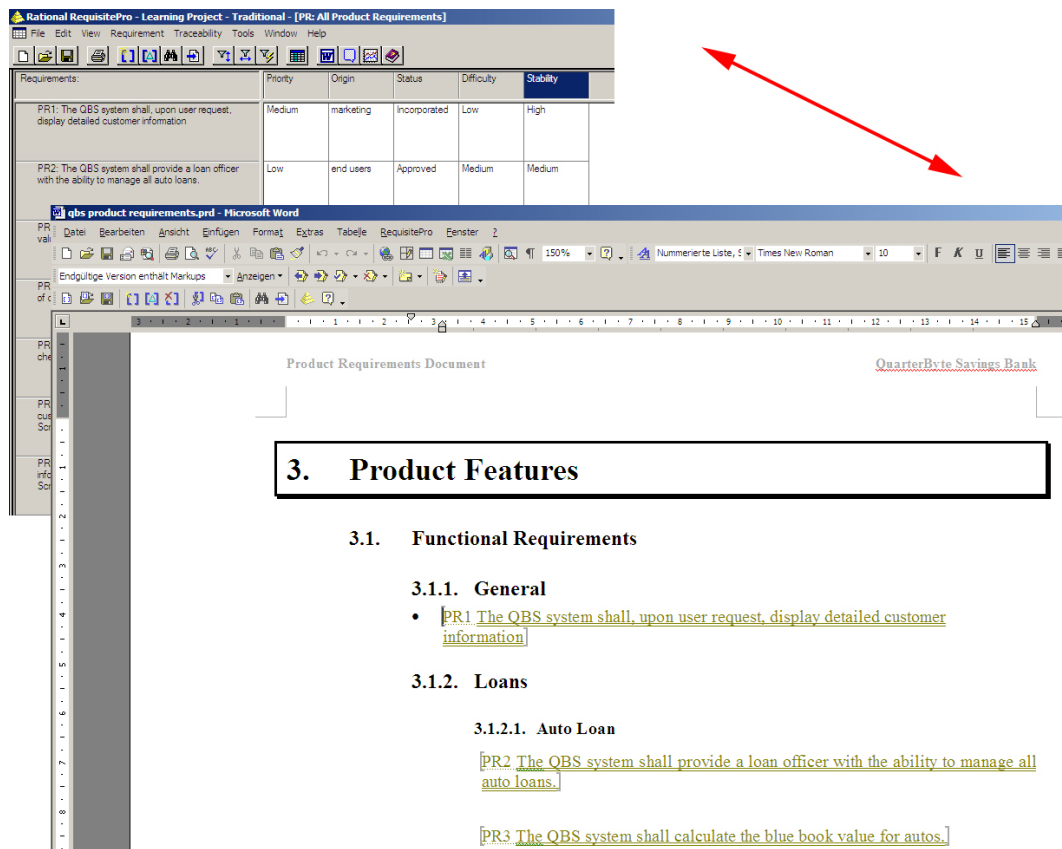


Figure 5.1: The information in Microsoft Word and RequisitePro is kept consistent.

changes of the Word document into the database, an add-in to the Word application is used. The application adds a menu and buttons to the Word tool bar, as seen in Figure 5.2.

All functions on the tool bar are also accessible through the RequisitePro menu in Figure 5.2 The following list describes the functions of the buttons:

1. Button  creates a new requirements document.
2. Button  opens an existing requirements document.
3. Button  saves the current requirements document.
4. Button  creates a new requirement.

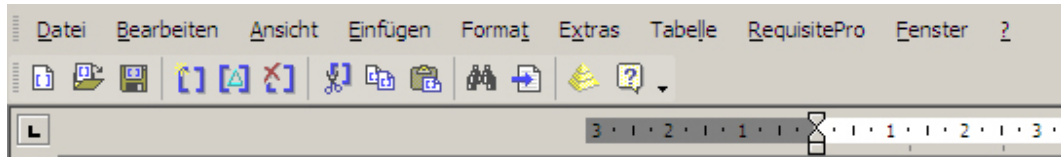


Figure 5.2: RequisitePro provides an add-on for Microsoft Word.

5. Button  adds attributes to a marked requirement.
6. Button  deletes a requirement.
7. Button  cuts a requirement out of the document.
8. Button  copies a marked requirement.
9. Button  pastes a requirement into the document.
10. Button  searches a requirement.
11. Button  goes to a requirement.
12. Button  opens the RequisitePro Workspace.
13. Button  opens the RequisitePro help.

When creating requirements documents with the aid of Word, it is advisable to save the document via the "save requirements document" button and not the "save document" function included in Word. Otherwise the requirements document would not be saved within the project of RequisitePro.

Moreover RequisitePro does not only come as a standalone application for every workstation, it also is distributed as a web solution, named "IBM Rational RequisiteWeb". This allows developers to access the RequisitePro workspace with an internet browser, which has all the functions that the RequisitePro application offers.

As seen in the RequisitePro application in Figure 5.1, the Matrix overview also allows applying attributes to each requirement. Commonly used attributes are automatically suggested by RequisitePro. The attributes can be

Requirements	Priority	Status	Contact	Difficult
FEAT1 Secure Payment method.	Medium	Validated	Fran	Medium
FEAT2 Easy Browsing for available titles.	High	Proposed	Fran	Low
FEAT3 Search capabilities	Medium	Validated	Fran	Medium
FEAT4 Ability to check the status of an order.	High	Validated	Fran	Low
FEAT5 E-mail notification for customers when new titles are added that may be of interest to them.	Medium	Validated	Fran	Medium
FEAT6 Highly Scaleable to include many titles and effective searching through those titles.	Medium	Proposed	Fran	Medium
FEAT7 Customer should be able to customize the web site.	Medium	Validated	Fran	Medium
FEAT8 Customer should be able to register as a user for future purchases without needing to re-enter personal information.	Medium	Validated	Fran	Medium
FEAT9 Ability to add/remove offerings.	Medium	Validated	Sandy	High
FEAT10 Ability to check on customer orders.	Medium	Validated	Sandy	Medium

Figure 5.3: RequisiteWeb is the web application of RequisitePro. [19]

also chosen according to their company's needs and specific attributes which are only used in the company can be added.

Moreover, user-defined views can be generated. In the example of Figure 5.4, a view was created which only lists medium and high priority requirements. This, for example aids people to figure out the requirements which have to be specified and implemented first.

Common views offered by RequisitePro are the following:

Attribute Matrix

The attribute matrix allows creating user defined views. For example with the aid of this view, a view can be generated in which requirements are only displayed according to their priority. The requirements will be displayed according to a given criterion that meets a certain attribute.

Requirements:	Priority	Origin	Status	Difficulty	Stability
PR4: The QBS system shall allow only maintenance of current savings accounts.	High	partners	Approved	Low	Medium
PR9: The QBS system shall provide the following reports:	High	end users	Proposed	High	Medium
PR13: The QBS system shall offer a comprehensive on-line help system.	High	marketing	Approved	Medium	High
PR18: The QBS system shall use no more than 1 megabyte of disk space to maintain customer data.	High	marketing	Approved	Medium	Medium
* <Click here to create a requirement>	Medium		Approved	Medium	Medium

Figure 5.4: A user defined view.

Traceability Matrix

This view creates a traceability matrix between two kinds of requirement types.

Traceability Tree (traced into)

This view creates a traceability tree which displays a tree structure in which the root nodes are traced to leaves.

Traceability Tree (traced out of)

This view creates a traceability tree which displays a tree structure in which the branch nodes are traced to the root nodes.

As by basic installation and provided by the built-in templates, RequisitePro comes with five requirements types:

- FEAT: Feature
- NONE: Default for documents without requirements
- STRQ: Stakeholder Request

- SUPL: Supplementary
- TERM: Glossary Item
- UC: Use Case

However, the requirement types are variable and can be defined for each project. For example, a company can define the requirement types it needs on their own.

5.1.1 Tracing Support in RequisitePro

Traceability can be established in RequisitePro very easily. Each requirement has a property pane, and with aid of this pane requirements traceability can be managed.

Managing requirements traceability is very easy, as the traceability can be created in simple steps. Moreover the traceability analysis is supported with the aid of various views. Following, the views associated with requirements traceability will be shown by examples.

Tracing Matrix

A traceability matrix, as seen in Figure 5.6, displays all the traces between two kinds of requirements. As seen in the image, an arrow which looks downwards means that the requirement in the column is traced from the requirement in the associated row.

An arrow looking upwards indicates that the requirement in the row is traced to the associated requirement in the column.

An arrow which is crossed out indicates that the trace is only a suspect and has to be verified. A suspect is a trace relation that has to be verified.

Traceability Tree (traced into)

This kind of traceability tree, as seen in Figure 5.7 covers all traces which originate from the given kind of requirement.

Crossed out signs mean that the traces are suspects and have to be verified. This kind of tree can be used for coverage analysis, because it can help in proving that each requirement has a successor in a high level requirement.

Traceability Tree (traced out of)

This kind of traceability tree, as seen in Figure 5.8 displays all the requirements traced from the requirements on which the tree is based on.

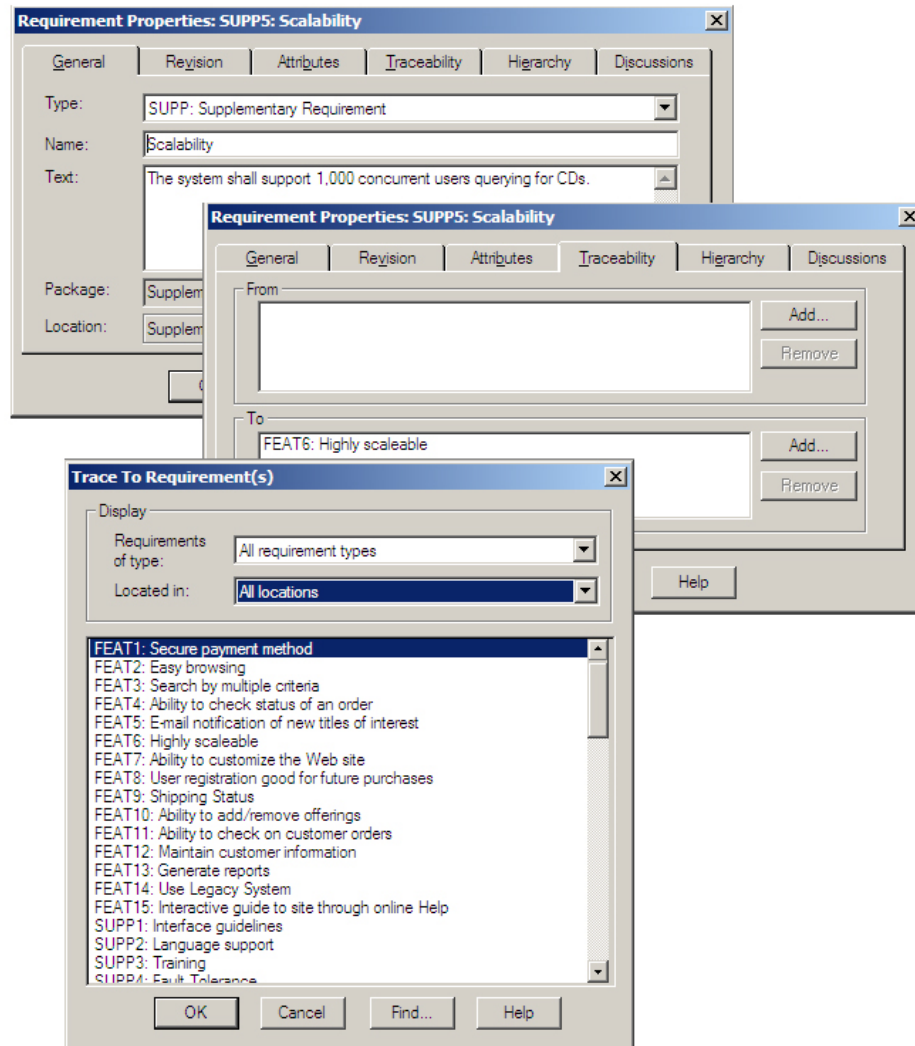


Figure 5.5: Applying of the traceability information with the property pane.

Summarizing, one can say that RequisitePro is a very powerful requirements management tool. It supports basic bidirectional traceability, only on requirements level. With the aid of the views a tracing analysis can be carried out. Therefore, the impact of changing requirements can be fully covered. To

Relationships - direct only	UC1: Arrange... Arrange Shipment	UC2: Check... Check Order Status	UC2.1: Brief... Registered ClassicCD.com Shoppers can view the status of their...	UC2.2: Basic... BEGIN: The use case starts when the Shopper chooses to check...	UC2.3: TRACK... TRACK PACKAGES. A BF VIEW ORDER DETAILS, the...	UC2.4: USER... SHOPPER ALREADY LOGGED ON. A BF IDENTIFY...	UC2.5: INVAL... INVALID CUSTOMERID OR PASSWORD. A BF IDENTIFY...	UC2.6: OPTIO... OPTIONS FOR DISPLAY. A BF SHOW EXISTING ORDER, the...	UC2.7: Pre... The Shopper has previously purchased CDs.	UC3: Purchase... Purchase CD	UC3.1: Brief... A Shopper has already selected CDs he/she is considering for...	UC3.2: Basic... BEGIN: The use case starts when the Shopper chooses to...	UN KN ST ST
FEAT1: Secure payment... Secure payment method													
FEAT2: Easy browsing Easy browsing for available titles													
FEAT3: Search by multiple... Ability to search for CDs by multiple criteria													
FEAT4: Ability to check... Ability to check the status of an order													
FEAT5: E-mail notification... E-mail notification for Shoppers when new titles that may interest them are added to the collection.													
FEAT6: Highly scalable Highly scalable to include many titles and effective searching through those titles													
FEAT7: Ability to... Shoppers should be able to customize the Web site													
FEAT8: User registration... Shoppers should be able to register once for all purchases; they should not be asked to...													
FEAT9: Shipping Status Shoppers should be able track any package that has been shipped to them.													
FEAT10: Ability to... Ability to add/remove CDs available for sale													
FEAT11: Ability to check... Ability to check on Shopper orders													
FEAT12: Maintain... Maintain Shopper information													
FEAT13: Generate reports Generate reports													
FEAT14: Use Legacy... Use existing legacy Warehouse System for shipping orders													
FEAT15: Interactive guide... The Web site will include an													

Figure 5.6: Traceability Matrix.

gain even further traceability into the source code, working with an additional product of the IBM Rational product family is suggested like IBM Rational ClearCase. Moreover, IBM Rational RequisitePro can cooperate with any of the IBM Rational software testing application. With the aid of this whole product family, a fully established requirements engineering process can be supported.

5.2 IBM Rational ClearQuest

IBM Rational ClearQuest is a customizable defect and change tracking tool used for dealing with the intensive change request during software developing. ClearQuest does not only keep track of the change requests, it also offers the creation of charts for a better analysis and overview of the changes and to create a summary of the changes. These change requests comprise enhancement requests, defect reports, and documentation modifications.

Rational ClearQuest bases on a database, which can be Microsoft Access, Sybase SQL Anywhere, Microsoft SQL Server, Oracle relational databases,

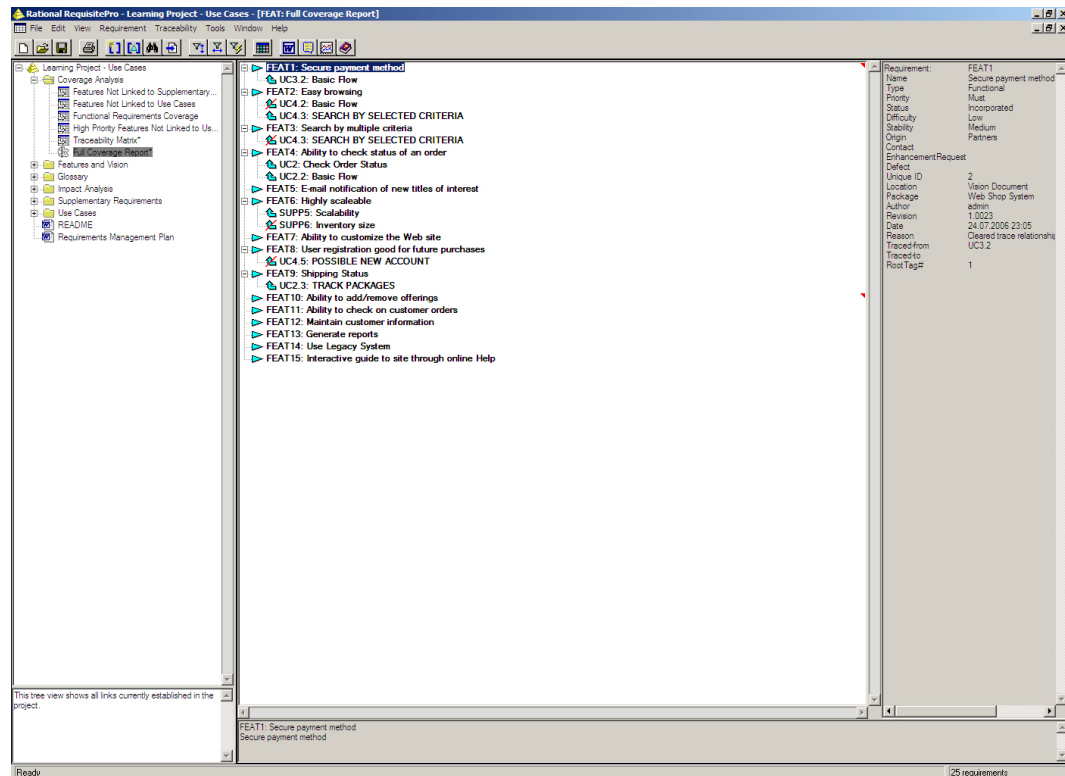


Figure 5.7: Traceability Tree (traced into).

or IBM DB2. Moreover, ClearQuest interacts with other tools of the IBM Rational product family. To associate a change of the software product directly with a change request, an interaction with Rational ClearCase should be established. To control the change requests which have been detected by tests, ClearQuest should interact with one of the Rational test tools. These are Rational TeamTest, VisualTest, Purify, PureCoverage, and Quantify. In this way a change request can directly be submitted to ClearQuest. Moreover, ClearQuest interacts with RequisitePro. When an interaction is established, it is possible to associate change requests with a requirement in RequisitePro and the requirement can be adapted coherently according to the change.

The following section describes an example treatment of a change request: Every time a new change request is submitted, it gets a new identification tag. At the beginning the change request record has the status 'submitted'. The next step is to assign the change request to a responsible person. When this is done, the change request changes its status to 'assigned'. When the responsible person is working on the change request, the status is changed to 'opened', when the problem is solved the status is changed to 'resolved' and

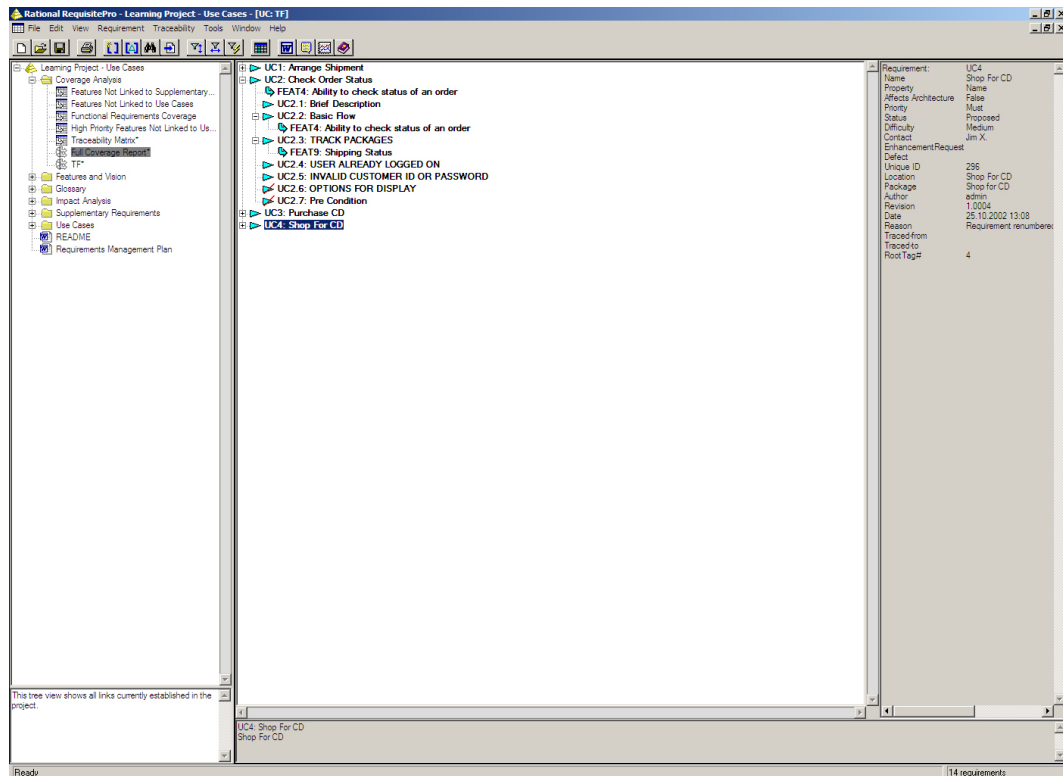


Figure 5.8: Traceability Tree (traced out of).

the change request is closed.

Like RequisitePro, ClearQuest also comes with a web interface. Moreover, ClearQuest is also very flexible, the record contents can be edited and extended as the company needs, and the charts and views can be created and designed as needed.

Another main feature is the integration into RequisitePro. A change request can be associated with a request in RequisitePro and can be viewed in RequisitePro. The change request can be then applied to the requirement and the requirement can be altered according to the change request.

5.3 Harvest Change Manager

Harvest Change Manager is a software change tracking system and software version control application.

The Harvest Change Manager is a client/server application and is therefore scaleable for large systems and is designed for heavy concurrent use of the

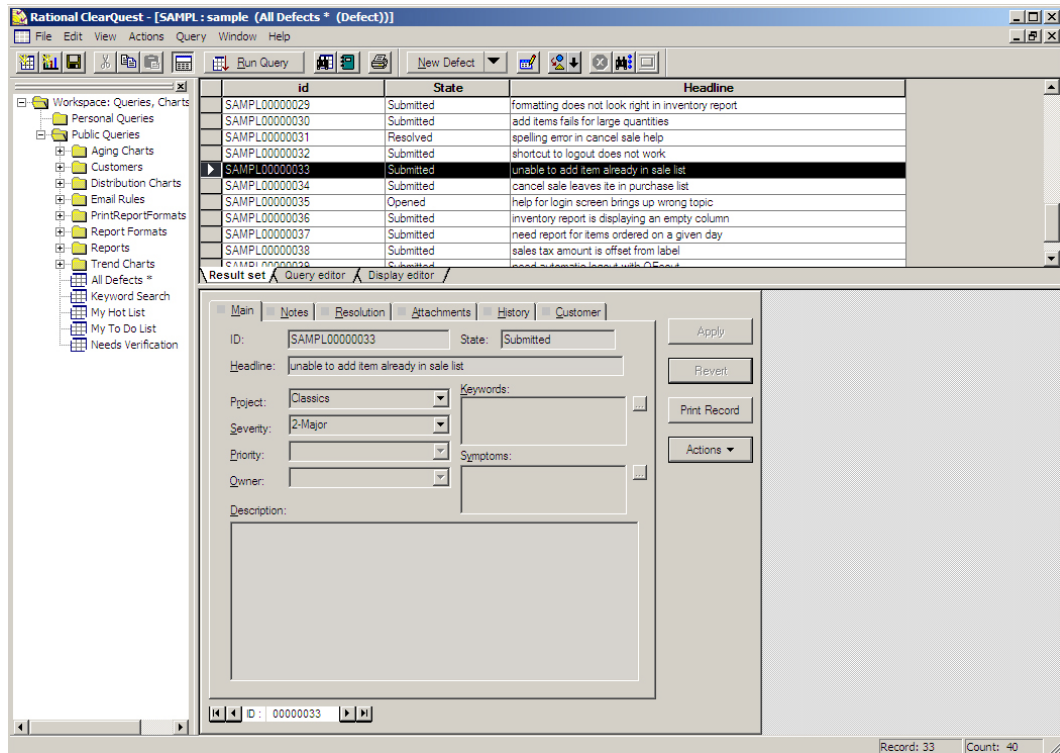


Figure 5.9: The ClearQuest Workplace.

application.

Harvest Change Manager has a lot of strong features that aid in supporting development teams. Harvest Change Manager is very flexible in the means of development life cycle processes. Moreover due to its GUI, the Harvest Change Manager has a high usability. The main features of the Harvest Change Manager are the concurrent development support, which means, that more than one developer can work on the same piece of code simultaneously. The second main feature is the parallel development support, which means that multiple releases of one application can be maintained.

Furthermore, Harvest Change Manager tracks all changes and testing defects in their associated change packages and forms. Therefore, the Harvest Change Manager keeps a history of the development process which includes changes and the change of the reasons.

A description of the most important units in the Harvest Change Manager hierarchical tree, as seen in Figure 5.11, is given next:

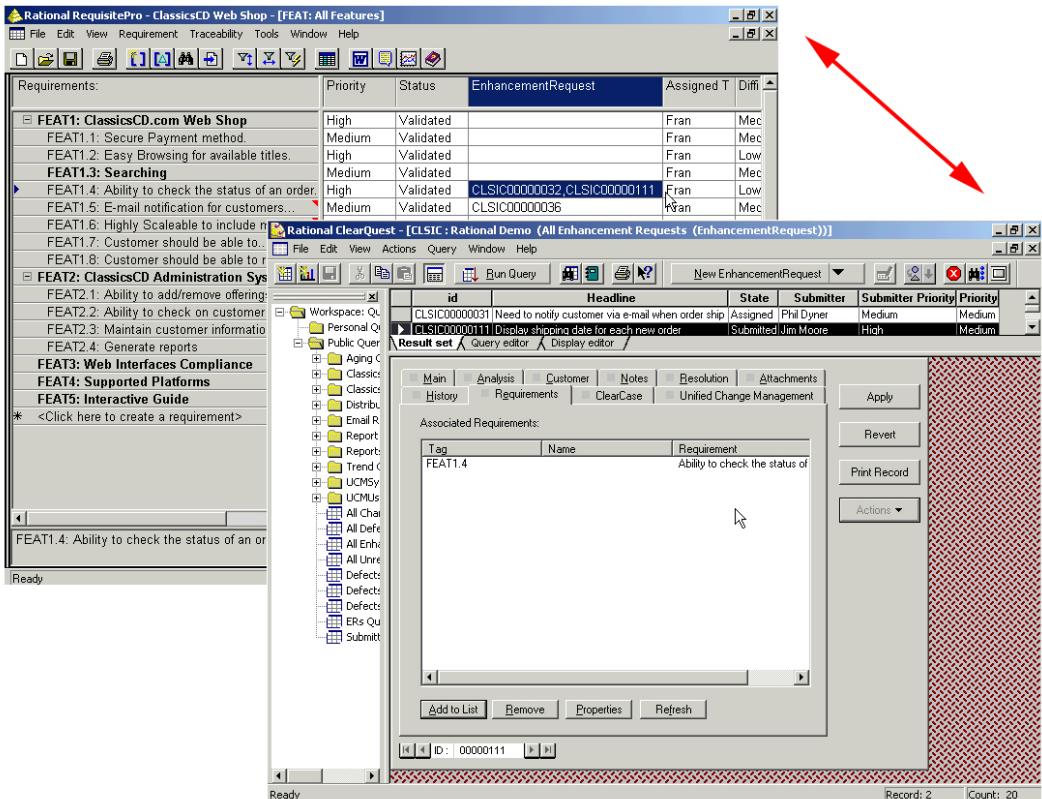


Figure 5.10: An example of the integration of RequisitePro and ClearQuest.

Projects

A project is the control structure and represents the biggest unit in Harvest Change Manager. It encompasses all users, data and object that are required to finish an activity. Moreover, a life cycle is part of a project and it determines what activities should take place, when they should take place and by whom they should be carried out.

Repositories

Harvest maintains the data in one or more repositories. These repositories can be associated with more than one project.

Forms

To track changes and issues and to document the test defects and arising problems, forms are used as a structured means of communication.

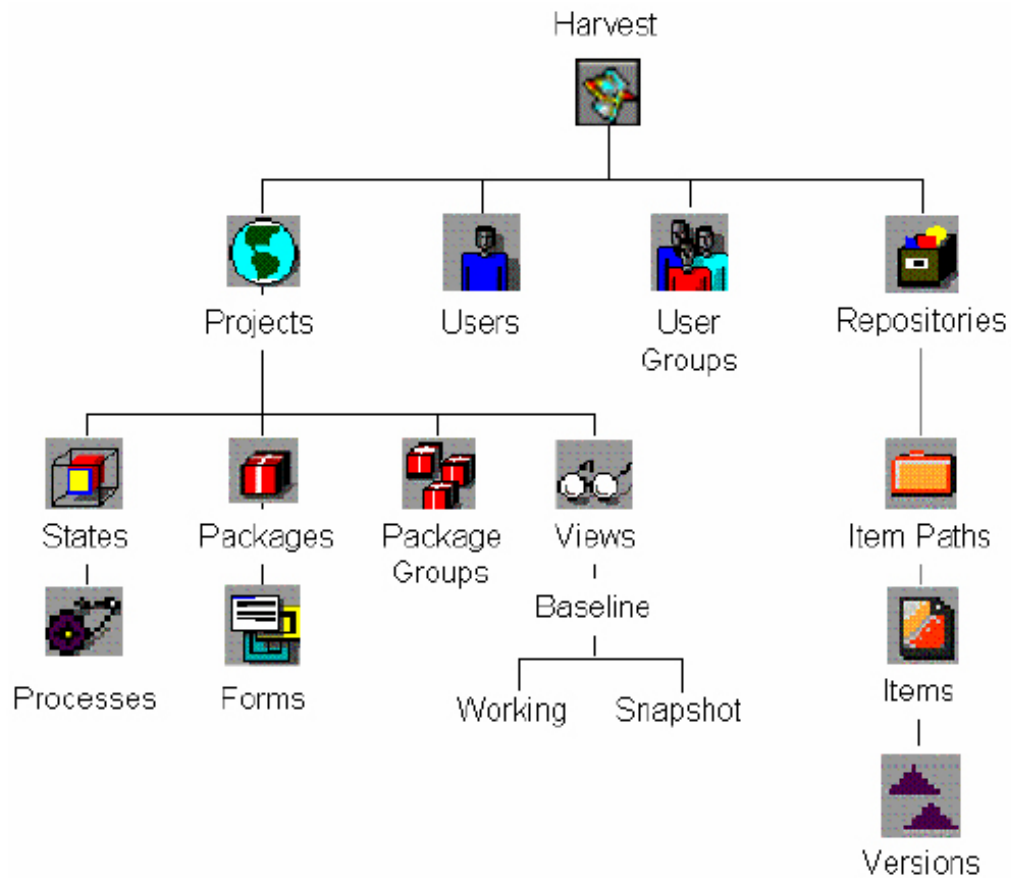


Figure 5.11: Harvest Change Manager Hierarchical Tree. [21]

Package

The basic unit of work that moves throughout the life cycle is called package. A package typically represents a problem which needs to be tracked and also all associated changes which have been made to solve the problem.

States

A state is a work area in which activities take place when packages move from identification to completion. A life cycle can encompass more than one state.

Processes

Actions that can be executed on objects in Harvest Change Manager are called processes. For each state there exist associated processes which can only be

executed in the state. An example process is "check-in", which creates a new version of an item by adding the changes which were made in a file into Harvest Change Manager.

Package Groups

The logical grouping of related packages is called package groups. A package can belong to one, several or no package groups.

Items

An item consists of data located in a repository that is therefore under control from Harvest Change Manager. An item is the same as a file in a file system.

Versions

A new version of an item is created whenever an item is changed.

The Harvest Change Manager offers a very sophisticated solution for tracking software changes. Very important are the clustering of problems as packages and package groups which aid in tracking the changes made to items. This clustering can aid in verifying the implementation of requirements, of course only by hand. Verifying can be achieved by querying the reasons for change and creation of a package and then checking the package against the requirements and change requests.

Chapter 6

The Tracing Implementation at AVL

This chapter describes examples of requirements traceability at AVL. AVL has been certified according to CMMI and therefore has to have established requirements traceability.

This chapter describes the methodology of traceability at AVL with the help of examples.

6.1 Requirements in AVL Projects

The basis for each project forms the requirements specification from the customers. From this specification enhancement requests are deviated. These enhancement requests form the basis of the requirements pyramid. (Figure 6.1) From the top of this pyramid way to its bottom the requirements are getting more and more fine grained.

Following three main dependencies between the pyramid requirements types can be identified:

- Traces between customer requirements specification and product requirements.
- Traces between product requirements and use-cases.
- Traces between functional requirements, which are documented in the product requirements specification, and architecture design

The documentation of these traces is not easy because in all cases it is not a *1 on 1* relationship, but rather a *n on n* relationship. This relationship results on the methodologies used to document the traces.

REQ.ID	Requirement Text	Enhancement Request ID	Product Req ID
7.4.14_R2	The Automation System shall provide the capability to online modify controller parameters of the PID Controllers in order to control tuning parameters in the application.	43998	PR40.9.11.4
7.6.1.1.2_R1	The Automation System shall provide an online display for the PID Controllers containing the following info: controller name, controller state, controller channels and controller parameters.	43998	PR40.9.11.2
7.6.1.1.2_R2	The Automation System shall provide commands for the PID Controllers to online modify controller parameters (constants) in order to control tuning parameters.	43998	PR40.10.1.5.2
7.6.1.1.2_R3	The general purpose PID Controllers should provide online functions available to apply changes, save changes to the parameter loadset and to discard changes.	43998	PR40.9.11.4.1
7.6.7.2_C1	The Automation System shall provide the capability to define up to 12 PID Controllers (Closed Loop Devices Controls), which use the actual value from an input channel and a demand value to calculate a control value for a designated output channel. The user shall be able to define the controller parameters.	43998	PR40.9.12.2
7.6.7.3_R1	The Automation System shall include general purpose PID Controllers for definition of device controls.	43998	PR40.1.1
7.6.7.3_R2	The general purpose PID Controllers shall support proportional, proportional/integral and proportional/integral/derivative algorithms.	43998	PR40.1.1
7.6.7.3_R3	The PID Controllers shall provide the ability to specify gain and a time interval parameter.	43998	PR40.9.5
7.6.7.3_R5	The PID Controllers shall execute control loops with a frequency of up to 100 Hz.	43998	PR40.9.5.10

Table 6.1: Example requirements from the customer requirement specification. [24]

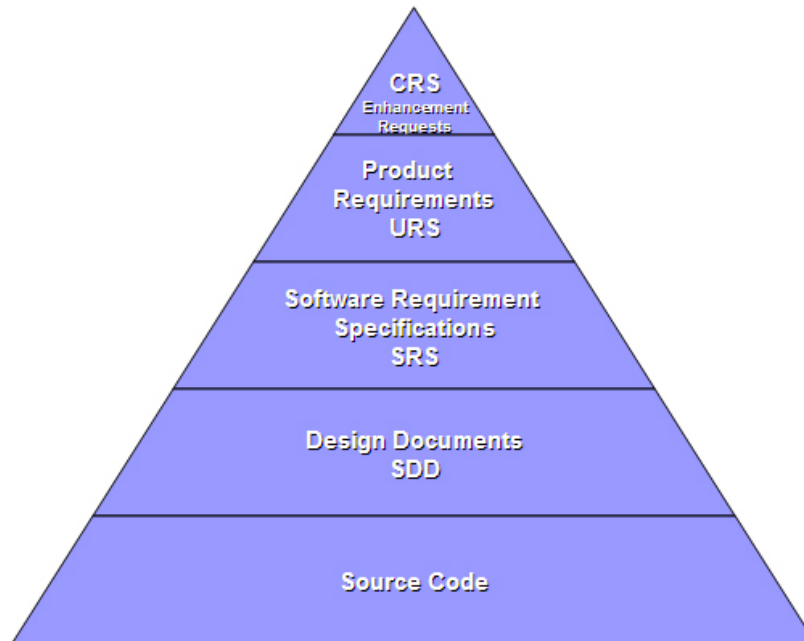


Figure 6.1: Requirements Pyramid. [22]

6.2 Traceability between User Requirements and Product Requirements

The basis for a project is the customer requirements specification (CRS) created by the customers. Nine example requirements are shown in Table 6.1. To fulfill the wishes of the customers the product manager has to create enhancement requests. These enhancement requests are stored in the product enhancement requests database. An enhancement request has following setup as seen in Table 6.2.

However, neither the customer requirements nor the product enhancement request in this example (Table 6.1, Table 6.2) are enough fine grained to present the requirements for the product enhancement with sufficient details. Therefore more detailed product requirements are defined. In this example the requirements can be detailed up to 83 requirements. Every single product requirement has its unique ID, which can be used for its traceability. Furthermore the requirements also contain a reference to the enhancement requests the product requirement is based on.

In this stage the product manager also has to unify the requirements of the customers which may be similar or even conflicting. In this example the

Enhancement Request	
Case ID	PE00043998
Headline	GeneralPurposeControllers
Customer	xxxx, yyyy, zzzz
Description	<p>As part of the system board removal, the task for the PID's and 2/3point controllers must be replaced. With the new implementation it shall be possible to:</p> <ol style="list-style-type: none"> 1. Parametrize the controller frequency from 0.1 to 100 Hz 2. Parametrize Up to 32 controllers 3. Online change of the controller parameters

Table 6.2: Example Enhancement Request. [24]

functions for the GeneralPurposeControllers were also implemented in an old version. Therefore, the old functions also have to be supported in the new version.

The result of this stage is a product requirements specification (URS). Examples for an URS are given in Table 6.3. These requirements are a subset from the 83 product requirements which were detailed out of the customer requirements specification shown in Table 6.1.

Establishment of Traceability

The customer requirements (Table 6.1) and the product requirements (Table 6.3) are referenced by a unique ID. Furthermore, the customer requirements contain an ID of the enhancement request which leads to the realization of these requirements. This ID is also present in the product requirements.

With the aid of this information the customers can control which product requirement fulfills which customer requirement. Furthermore, the developers can control which product requirement fulfills which customer requirement.

Req.ID	Requirement Text	Enh.Req-ID	SRS	Traced from
PR40	General Purpose Control	43998	General Purpose PID & PT3 Control	
PR40.1	The automation system shall handle the following types of controllers for general purpose applications	43998	General Purpose PID & PT3 Control	
PR40.1.1	PID Controller	43998	General Purpose PID & PT3 Control	7.6.7.3_R1
PR40.1.2	23 -Point Controller	43998	General Purpose PID & PT3 Control	

Table 6.3: Example Requirements from the Product Requirements Specification. [24]

6.3 Traceability between Product Requirements and Uses Cases in the Software Requirements Specification

The next step in the establishment of the traceability is to analyze the product requirements. The functional requirements are represented as use cases and the non-functional requirements are described in detailed form according to their category in the software requirements specification (SRS).

The Use Cases are detailed as a text describing the use cases and are also documented as use cases in the use case model. The use cases are referenced by name identity in the SRS and in the use case model.

In the example described above (Table 6.1, Table 6.3) two use cases were identified: "Use Case: Define PID and PT2PT3 Controller" and "Use Case: Operate PID and PT2PT3 Controller" (Figure 6.2).

The SRS forms the basis for the creation of the test plan, the concept of the user documentation and the software design.

Establishment of Traceability

The traceability from product requirements to the use cases is established with the aid of a traceability matrix (Figure 6.3). For the verification of

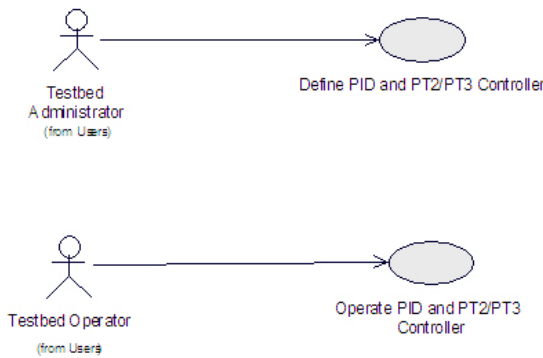


Figure 6.2: Use Cases. [24]

Relationships: direct only	UC33: General Purpose PID & PT3 Control	
	UC33.1: Define PID and PT2/PT3 Controller	
	UC33.2: Operate PID and PT2/PT3 Controller	
PR40: General Purpose Control		
PR40.1: The automation system shall handle the following types of controllers for general purpose applications:		
PR40.1.1: PID Controller		
PR40.1.2: 2/3-Point Controller		
PR40.9: The following specific features have to be provided for all types of controllers:		
PR40.9.1: All controller types can be parametrized in TFP, SYS, PAL and UUT parameter sets		
PR40.9.3: All controller types can be started automatically upon the following system state transitions and system events:		
PR40.9.3.1: Reaching state MONITOR		
PR40.9.3.2: Reaching state MANUAL		
PR40.9.3.3: Engine is started		
PR40.9.4: All controller types can also be started via activation objects from within a testrun or interactively via the diagnostic console, i.e. the 'System'.		
PR40.9.5: All controller types must operate synchronous to the data acquisition and have to support the following execution frequencies:		
PR40.9.5.1: 0.1 Hz		
PR40.9.5.2: 0.2 Hz		
PR40.9.5.3: 0.5 Hz		
PR40.9.5.4: 1.0 Hz		
PR40.9.5.5: 2.0 Hz		
PR40.9.5.6: 5.0 Hz		
PR40.9.5.7: 10.0 Hz		
PR40.9.5.8: 20.0 Hz		
PR40.9.5.9: 50.0 Hz		
PR40.9.5.10: 100.0 Hz		
PR40.9.6: All controller types support the following reference value sources (i.e. 'demand value sources'):		
PR40.9.6.1: Constant value		
PR40.9.6.1.1: In case of 'Constant Value' the reference value can be adjusted via activation objects		
PR40.9.6.2: System channel		
PR40.9.6.3: Lookup table		
PR40.9.6.3.1: In case of 'Lookup Table' the reference value is calculated by interpolating a given lookup table with the controller frequency.		
PR40.9.6.3.2: The input channel definition for the interpolation is defined along with the lookup table definition, i.e. it is not part of the controller		
PR40.9.7: All controller types can be operated in the following modes:		
PR40.9.7.1: 'Dynamic Reference Value', i.e. the reference value is either given by a specific 'reference value system channel' or a 'reference value'.		
PR40.9.7.1.1: In case of invalid 'dynamic reference values' the controller automatically is switched to the 'manual reference value' mode.		
PR40.9.7.1.2: If both a 'reference value system channel' and a 'reference value lookup table' are defined it is possible to switch between these two...		
PR40.9.7.2: 'Manual Reference Value', i.e. the reference value is adjusted via activation objects.		
PR40.9.7.2.1: Upon switching from 'dynamic' to 'manual' mode the current dynamic reference value is used as initial manual reference value.		

Figure 6.3: Traceability Matrix. [24]

the complete implementation all the product requirements and the software requirements have to be fulfilled.

Furthermore, with the aid of queries the traceability matrix can validate if all requirements have been covered.

6.4 Traceability between Use Cases and Architecture Design

The transition of the use cases to architecture design is the most complex step. Only small parts of the architecture are dependent on the functional requirements, instead, the non-functional requirements have the bigger influence. Non-functional requirements can be clustered into following categories:

- Availability
- Modifiability
- Performance
- Security
- Testability
- Usability

The architecture design is based on use case realization via sequence diagrams. With sequence diagrams (Figure 6.4) the dependence of the components become visible.

Establishment of Traceability

For each use case in the SRS a use case realization folder is created which contains at least one sequence diagram. If a complete system model is established, components of the model can be traced from use cases and use cases can be traced to components. The representation of the sequence diagrams was chosen, to gain an overview which test cases have to be rerun when components are changed. With sequence diagrams the way how components participate in the interaction can be identified. Therefore AVL uses an UML Tool, otherwise it could not be determined which components are used by which use cases and furthermore it could not be determined which use cases are affected by the change of a component. The traceability is therefore established with the UML Model of the software system.

6.5 Transition of the System Model to the Source Code

The sequence diagrams located in the use case realization folders provide candidates for the classes. The design maps the system model on the classes and is partially used to generate the code framework. Therefore the classes in the design model are identical with the classes in the source. The design model is layered, this layer structure is also used in the source code tree (Figure 6.5).

Figure 6.5 depicts the dependency of all steps in the traceability solution. The main inputs of this solution are the customer requirements specification, which results in the enhancement request initiated by the product manager. The next step is to identify the product requirements based on the CRS and enhancement requests. Following that, the use cases are identified and the SRS is written. Based on the use cases a system model is developed and detailed sequence diagrams are created. The next step is the transition to the design model, based on the classes identified in the system model. The classes in the class model provide the framework for the code.

With the mapping of components to a source code file, it can be determined which test cases have to be rerun when software components are changed. Based on the given model following scenario could appear: A hotfix package affects certain source code files. However, due to the mapping of components to the files it can be determined which components are affected. Furthermore, in the system model the use cases which use the changed components can be

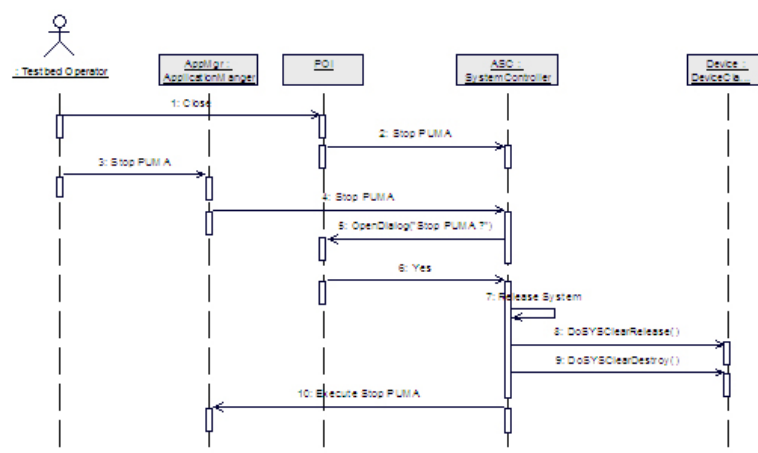
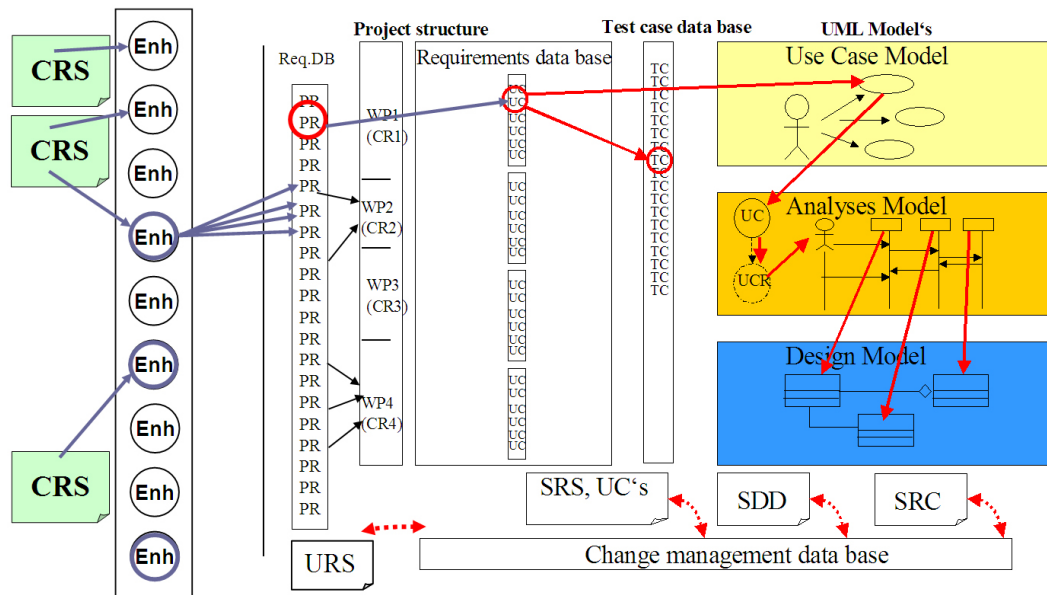


Figure 6.4: Example of a Sequence Diagram. [24]



6.6 Traceability of Change Requests

Change requests are stored in the change request database which is handled by Rational ClearQuest. Each change request is assigned a unique ID. Furthermore, if a change request has to be integrated the affected requirements artifacts like URS or SRS have to be adapted according to the change. To verify that the change request has been implemented, for each change request a new package is created in Harvest, which contains the change ID and a detailed description. With the aid of the Harvest find tool, it can easily be verified that a change request has been implemented. The change impact analysis identifies the requirement artifacts that have to

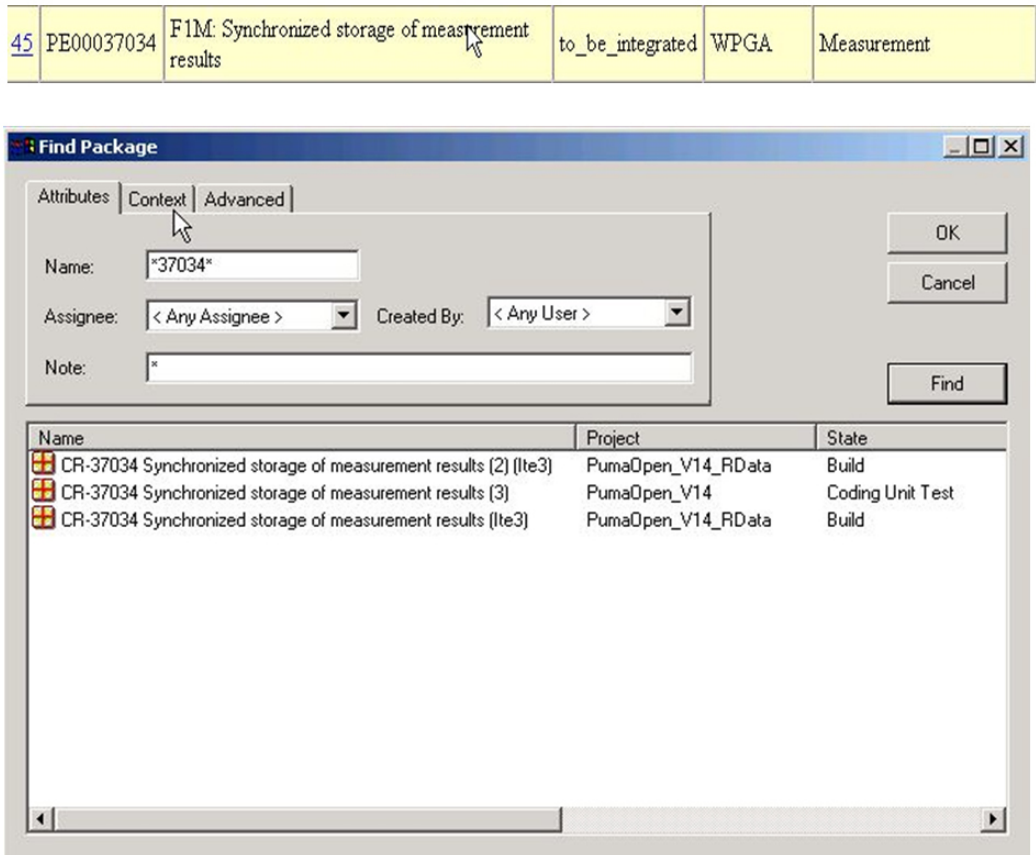


Figure 6.6: Change Request in ClearQuest and the associated Packages in Harvest. [23]

be changed. In a final review it has to be verified that the items have really been changed.

6.7 Analysis

The example presented in this chapter describes a method of how to trace requirements from the customer requirements specification to the source and rudimentary from the source to the customer requirements. As the relationship between each level of the requirements artifacts is n to n tracing one requirement may lead to results which may not be detailed enough. The following points can be criticized in the implementation of traceability at AVL:

- The whole traceability chain is not supported by one tool. Each stage

has its own tool.

- There is no integration of the tools which are included in the traceability chain.
- There is no data exchange between the tools possible. Furthermore, in the worst case the requirement artifacts are referenced by name identity and not a unique ID.

To deal with all these flaws, the implementation depends on a detailed discipline of the developers. They have to establish traceability by hand. They also have to ensure that the IDs are correct and that, in the realm of use cases, name identity is given. It is hard to manage the consistency of the traceability. Also, especially tracing by name identity is not resilient to faults. Furthermore, when changes occur it can get difficult to keep track of the changed references.

Possible improvements maybe the integration of the tool chain, which can deal with changes of the tracing structure. It would also lead to a simplification of the traceability. Furthermore it would be more resilient towards faults.

Chapter 7

Conclusion

The report was focused on requirements engineering, particularly the traceability of requirements. It points out multiple deficiencies of requirements engineering. Various scientific publications argue that requirements engineering has to be integrated in the whole development cycle as requirements drive the whole project. This goes hand in hand with new approaches for traceability methods. As requirements change in the course of a project, methods and tools have to cope with this situation.

The flaws in requirements engineering need to be addressed because surveys show that wrong and inconsistent requirements engineering can lead to project failures. [13]

The growing complexity of embedded real-time systems puts a further challenge into requirements engineering. Embedded real-time system development methodologies have not been significantly improved for a long time. Therefore, the existing requirements engineering methodologies with all their flaws may not be that helpful for developing embedded real-time systems. Promising approaches like FeatureNet [16], which cover functional and non-functional requirements and furthermore the dependencies between the requirements, may be the future for new requirements engineering methodologies for embedded real-time systems.

As shown in the example of traceability at AVL in Chapter 6 it is very hard to establish traceability. It works fine, however, with certain restrictions. Changes which affect not manageable numbers of use cases cannot or can only be dealt with restrictions. Furthermore, the traceability is not automated. Between each level of requirements granularity the traceability is established by hand, in the worst case by name identity. This results in a lot of work and maintainability to keep the traceability intact and the requirements consistent.

Requirements engineering and especially requirements engineering of embedded real-time systems definitely needs to be improved. The report points out how traceability methodologies should be improved. New methodolo-

gies and tools have to be developed to support the rising complexity of the requirements engineering of embedded real-time systems.

Bibliography

- [1] P. Sawyer I. Sommerville. *Requirements Engineering. A good practice guide*. WHILEY, 1997.
- [2] J. Dick E. Hull, K. Jackson. *Requirements Engineering*. Springer, 2002.
- [3] H. Kopetz. *Real-Time Systems. Design Principles for Distributed Embedded Applications*. Kluwer, 1997.
- [4] M. Neal R Walkins. *Why and How of Requiremens Tracing*. 1994.
- [5] F. Pinheiro. *Formal and Informal Aspects of Requirement Tracing*.
- [6] A. Finkelstein O. gotel. *An Analysis of the Requirements Traceability Problem*. 1993.
- [7] A. Egyed. *A Scenario Driven Approach to Traceability*. 2002.
- [8] A. Finkelstein O. Gotel. *Extending Requirements Traceability Through Contribution Structures*.
- [9] L. Murray et al. *Requirements Traceability for Embedded Software - An Industry Experience Report*.
- [10] M. Broy. *Requirements Engineering for Embedded Systems*.
- [11] M. Chrissis et al. *CMMI Guidelines for Process Integration and Product Improvement*. Addison-Wesley, 2003.
- [12] I. Sommerville. *Integrated Requirements Engineering. A Tutorial*. 2005.
- [13] Standish Group. *The Chaos Report*. 1994.
- [14] CMMI Product Team. *CMMI - Capability Maturity Model Integration Version 1.1 - Continuous Representation*. March 2002.
- [15] CMMI Product Team. *CMMI - Capability Maturity Model Integration Version 1.1 - Staged Representation*. March 2002.

- [16] J. Hartman et al. Feature Net - ein Ansatz zur Modellierung von automobilespezifischen Doänenwissen und Anforderungen. Technical report, GI Jahrestagung, 4. Workshop Automotive Software Engineering, Dresden, Germany, 5. Oktober 2006.
- [17] http://www.it-cortex.com/Stat_Failure_Cause.htm. *Failure Cause*. Snapshot August 2006.
- [18] Rational. *Rational RequisitePro - Users Guide*. 2003.
- [19] IBM. *IBM Rational RequisitePro*. 2004.
- [20] IBM. *IBM Rational ClearQuest*. 2004.
- [21] Computer Associates. *AllFusion Harvest Change Manager - Tutorial Guide*. 2005.
- [22] Gerhard Hauser. *Unifying Requirement in PE*. 2005.
- [23] Gerhard Hauser. *Tracability*. 2005.
- [24] Gerhard Hauser. *Verfolgbarkeit von Anforderungen*. 2006.
- [25] http://en.wikipedia.org/wiki/Waterfall_model. *Waterfall Model*. Snapshot August 2006.