International Requirements Engineering Board

Stan Bühne
Andrea Herrmann

Handbook

Requirements Management according to the IREB Standard

Education and Training for the IREB Certified Professional for Requirements Engineering Qualification

Advanced Level "Requirements Management"

Version 1.1.0

Terms of Use

Acknowledgements

We would like to express our gratitude to IREB e.v. for giving us the opportunity to write this book and for their continued organizational support throughout the process. The IREB "Requirements Management" working group has contributed significantly to the contents of the book, as the book is based on the syllabus created by the working group and the members of the working group have supported us with comments, notes, and additional material. We would therefore like to express our heartfelt thanks to the entire working group (in alphabetical order): Frank Engel, Sven Eselgrimm, Günter Halmans, Frank Houdek, Patrick Mäder, Alexander Rachmann, Thomas Schölzl, Amin Soesanto, Frank Stöckel, and Malik Tayeh. Special thanks are due to Professor Dr. Martin Glinz and Dr. Thorsten Weyer who, with their thorough appraisal and constructive comments, put the finishing touch to the book.

This handbook was produced by (in alphabetical order):

Stan Bühne, Dr. Andrea Herrmann. Translation to English by Tracey Duffy.

Table of Contents

The IREB CPRE Advanced Level Module "Requirements Management"

Anyone working as a requirements engineer, business analyst, consultant, demand manager, or project manager in system and software development projects knows that for a project to be implemented successfully, it is *by no means* enough to simply know the stakeholders of the project and to document their aligned requirements at the beginning of the project!

No, even if all requirements were well structured, aligned, and accepted at the start of the project, they will change by the end of the project or "go live"—and always at the worst times. Requirements also change during operation of the system (or software) concerned and should be kept up to date for documentation purposes until the system is decommissioned.

However, the fact that requirements change is due neither to the requirements engineer or poorly selected methods, nor to the stakeholders involved; it is usually simply due to the nature of things and constraints which change over time.

The more complex the project, the more indispensable **requirements management (RM)** is to avoid uncontrolled "fire-fighting" at such times and to enable you, as the requirements manager, to be able to provide information about the status of the requirements or about the effects of any change requests at any given time.

On one hand, requirements management includes the conscious management of requirements in the classic sense (e.g., by means of assignment of attributes, creation of views, traceability, etc.) as well as the management of changes to requirements. On the other hand, the prior planning and monitoring of the defined **requirements engineering (RE)** processes are also part of requirements management, in the sense of: "How do I elicit, document, and review my requirements to be able to continuously report on the status and to react to planned changes?".

In this handbook for the IREB CPRE Advanced Level Module "Requirements Management", we would therefore like to consider requirements management from both sides. To do so, we present the essential concepts of requirements management, but always also describe the necessary planning aspect which enables a conscious management of requirements.

To consciously manage requirements, the requirements manager must plan and define the following at the beginning of the requirements engineering process:

- The requirement types to be considered, the format in which they must be presented, and the level of detail to which they must be specified
- The questions the requirements manager must answer on the basis of his requirements and the views that are necessary for the different stakeholders
- The criteria to be used to evaluate the requirements to support prioritization
- Version control for requirements and requirements documents
- How and when changes should be handled
- The requirements and other development artifacts between which traceability must be achieved

- Whether and how to document requirement variants within the requirements specification
- The requirement status reports needed, the information they must contain, and the sources (for example, attribute documentation) that can be used to determine this information
- What the exact requirements engineering process (or sequence of activities) for the project should look like, and how the process can be monitored and potentially improved

The results of these considerations are documented in a **requirements management plan (RMP).** With this document, both the requirements engineering process as a whole and the reporting, prioritization, and changes (that are part of requirements management) can run in a structured manner according to plan.

Requirements management is planned and executed by the requirements engineer or by someone exercising the separate role of a **requirements manager**. Within the scope of requirements management, the requirements manager plans, manages, and monitors the requirements engineering process and its artifacts, reporting, for example, to the client or project manager. The requirements manager also coordinates changes.

The requirements management plan ensures that the requirements engineering process can be monitored actively and that subsequent decisions can be taken consciously and such that they can be traced. This does not mean that requirements engineering is not an iterative, incremental, and creative process: it merely means that the requirements engineering process should be planned creatively and consciously and should not be chaotic!

This handbook supports the requirements manager in the creation of the requirements management plan by explaining the concepts and terms required for the plan and presenting appropriate methods.

This book also shows how requirements engineering and requirements management can be implemented in agile projects—after all, requirements (e.g., user stories) are also documented in agile projects and such projects must also be able to handle changes, prioritization, etc.

In practice, it is difficult to imagine specifically implementing requirements management in complex projects without using tools. Therefore, in the last chapter of the book, we describe options for support from requirements management tools, as well as the limitations of these tools.

With these topics, the handbook for the Advanced Level module "Requirements Management" of the IREB Certified Professional for Requirements Engineering (CPRE) provides you with the know-how you need to manage requirements consciously and in a structured manner.

We hope you enjoy reading the book!

More information on the IREB Certified Professional for Requirements Engineering — Advanced Level "Requirements Management" can be found at: http://www.ireb.org.

Foreword

This *Handbook for Requirements Management according to the IREB Standard* supplements the syllabus of the International Requirements Engineering Board for the Advanced Level module "Requirements Management" Version 1.1.0 dated May 10th, 2018 and is based on the IREB Glossary [Glin2014].

The target audience for the handbook includes both training providers who want to offer seminars on requirements management according to the IREB standard, and training participants and interested practitioners who want a detailed view of the subject matter for this Advanced Level module and of requirements management according to the IREB standard.

This handbook is **not** a substitute for training or educational books on the topic. The handbook in fact represents a link between the compact syllabus (which merely lists and explains the learning objectives of the module) and the multitude of literature that has been published on the topic of requirements management in recent decades.

Together with the references to additional literature, the content described in this handbook is intended to support training providers in preparing training participants specifically for the certification exam. This handbook offers training participants and interested practitioners the opportunity to expand their knowledge in the area of requirements management and to work through the detailed content based on literature recommendations. Furthermore, the handbook is also intended as a reference—for example, for refreshing knowledge about the various topic areas in requirements management after successful certification.

In addition to the content that expands on the syllabus and is relevant for the exam, in each chapter, the handbook offers explanatory examples based on a continuous case study. The case study is identified by the icon shown here on the left. The content in the case study is not directly relevant for the exam but is highly recommended to allow a better understanding of the other content in the handbook.

We also offer interested readers information which goes beyond the exam and which is not relevant for the exam. Where this additional content fits within the flow of the material, it has been integrated in the respective chapter and flagged as not relevant for the exam with a red marking to the side (see right).

The additional content in Annexes A to C is also not relevant for the exam.

We are happy to receive any suggestions you might have for improvements or corrections.

E-mail contact: requirementsmanagement.guide@ireb.org

We hope you enjoy reading this handbook and wish you good luck with the certification exam for the IREB Certified Professional for Requirements Engineering - Advanced Level "Requirements Management".

Stan Bühne
Andrea Herrmann
Autumn 2015

Version History

Version	Date	Comment	Author
1.1.0	September 1, 2019	Initial English version based on German version 1.1.0	Stan Bühne, Andrea Herrmann, Frank Houdeck, Stefan Sturm

1 What Is Requirements Management?

This chapter initially defines why requirements management is important, for whom and for what purpose it is important, what tasks requirements management includes, and who performs these tasks.

1.1 Definition of Requirements Management

As is the case for many terms, there are different definitions for the term "requirements management". The relationship between requirements management (RM) and requirements engineering (RE) is also not defined uniquely. Sometimes, requirements management is deemed to be part of requirements engineering (as is the case for the CPRE Foundation Level [IREB 2015]); and in other cases, requirements engineering is deemed to be part of requirements management (e.g., in [Schi2001]). In contrast, the CMMI [SEI2011] awards both requirements management and requirements engineering equal value.

Definition 1-1: We define requirements management as part of requirements engineering. Requirements engineering is therefore more extensive.

- *Requirements engineering* is a systematic and disciplined approach to the specification and management of requirements with the following objectives:

1. Knowing the relevant requirements, establishing consensus among the stakeholders about the requirements, documenting the requirements in compliance with given standards, and managing the requirements systematically
2. Understanding and documenting the stakeholders' desires and needs
3. Specifying and managing requirements to minimize the risk that the system that does not meet the stakeholders’ desires and needs [Glin2014] and [PoRu2011].

- *Requirements management*: the process of managing existing requirements and requirements-based artifacts. In particular, this includes documenting, changing, and tracing requirements [Glin2014]. It also includes managing the requirements engineering process, which means planning, controlling, and checking the requirements engineering process.

In the following, we state some additional definitions to show the diversity that exists in professional literature. The IREB has defined a clear definition so that certified requirements managers and requirements engineers always mean the same thing when they use a particular term. Unambiguous terms not only simplify collaboration in professional life; when, as recommended in the introduction to this handbook, the role of the requirements manager is introduced in addition to the role of the requirements engineer, the definition of the relationship between the two disciplines (requirements engineering and requirements management) defines the division of tasks across these two roles.

Ebert [Eber2012] also defines the following: "Requirements management (RM) is a part of requirements engineering that is concerned with the maintenance, management, and further development of requirements in the lifecycle." This lifecycle is very important with regards to requirements engineering. Accordingly, it is not enough to simply collect requirements once. Changes must be managed over the entire lifecycle of the software/system. Note that not only the software/system itself but also each individual requirement goes through its own lifecycle.

According to Pohl [Pohl2010], requirements management can be divided into three main subactivities:

1. Observation of the system context to reveal context changes
2. Management and execution of the requirements engineering activities (i.e., requirements management as process management)
3. Management of the requirements and related artifacts during the development process

According to the ISO/IEC/IEEE 29148:2011 [ISO29148] standard, requirements management is defined as *"activities that ensure requirements are identified, documented, maintained, communicated and traced throughout the life cycle of a system, product, or service"*. Requirements management involves not only the management of the requirements, but also all information associated with the requirements: "*Maintain throughout the system life cycle the set of system requirements together with the associated rationale, decisions and assumptions*." ([ISO15288], 6.4.2.3 b)

To illustrate the definitions and recommendations in this handbook, we use the example of an online banking system. We assume that the system already exists. The system was developed based on a complete and quality-assured requirements specification according to the standard norms. However, this is not enough. The requirements for the system change: due to changes in law (e.g., the changeover to SEPA); through continuous efforts to make online banking more secure but keep the same level of user-friendliness; or to make online banking more user-friendly with the same level of security; through technical innovations; through ideas from product management for new functionality; or through changes to business processes at the bank that have an effect on online banking. The objective now, despite these many change ideas that come from all directions, is to keep an overview of the requirements, estimate in advance what the costs and other consequences of implementing an idea would be, and to justifiably implement, defer, or reject the changes. Furthermore, all stakeholders (such as the IT department, product management, the executive board, the data protection officer, and the Customer Advisory Board) must be included in the process. Even our requirements engineer is a stakeholder.

Our requirements engineer is Peter Reber. He is 35 years old and has been working at our example bank for 10 years. He knows all the stakeholders well and is happy to contact them for no specific reason, simply to find out if there is any news about anything. The online banking was successfully introduced before Peter started working at the bank. Since being trained by his predecessor, Peter has been solely responsible for conscientiously managing the requirements for online banking according to the rules of requirements engineering.

However, his peaceful days are ending as the entire online banking is being changed to a new corporate design, with new colors, new fonts, technical terms, and logos. The bank wants to take this opportunity to introduce new functionality to increase safety and user-friendliness. A group of experts will soon be collecting or defining change requirements, which means that they will be performing requirements engineering: several external business analysts are analyzing the business processes, a team of IT security experts is conducting risk analyses, the usability expert is designing alternative interface designs and improving accessibility, and a moderator is holding an ideas workshop with the Customer Advisory Board. All of these people are performing requirements engineering. Peter Reber's task is to organize and coordinate their work. This means that while they are performing requirements engineering, Peter is taking care of the requirements management.

1.2 Tasks in Requirements Management

Here we define the tasks that make up requirements management. This definition is therefore also a role description: the requirements manager is responsible for requirements management and either performs the tasks belonging to requirements management or monitors the performance of the tasks by other persons.

Three main constraints [RuSo2009] make the requirements management tasks more complex:

- Requirements have to be used by multiple persons
- Requirements are supposed to be reused.
- Requirements change.

Requirements management is responsible for providing the rules and techniques required so that requirements and other information can be stored in such a way that everyone involved can find what they need. This must be planned in advance. Therefore, before the requirements engineering process begins, the requirements manager creates the r**equirements management plan (RMP)**.

Definition 1-2: The requirements management plan covers:

- The requirements landscape—that is, the types of requirement artifacts to be managed and the level of detail they contain (see Chapter 2)
- Attributes and views of the requirements (see Chapter 3)
- Prioritization criteria and methods (see Chapter 4)
- Version management for requirements and the change process (see Chapter 5)
- Managing the traceability of requirements (see Chapter 6)
- Variant management (Chapter 7)
- Reporting (Chapter 8)

- The requirements engineering process and activities to improve this process (Chapters 9 and 10)
- Tools to be used (Chapter 11)

In the following chapters, this handbook looks at the content of the requirements management plan and proposes different methods for the respective activities to be performed. At the end of each section there is a summary of the content that the requirements management plan should contain.

During the requirements engineering process, within the scope of requirements management, these tasks are performed according to the plan: views and reports are created and updated, requirements are selected for releases, changes are managed and prioritized systematically, product lines are defined and managed, tools are introduced, and the requirements engineering process is monitored and improved.

Requirements management is planned and executed by the requirements engineer or by someone exercising the separate role of a requirements manager. The relationship between the requirements manager and the requirements engineer is like that of the relationship between the quality manager and a tester. This division of tasks makes sense when, in complex projects with a critical schedule, there is so much work for requirements engineering and requirements management that multiple persons have to collect, align, and manage requirements. A role is then required that sets up and monitors the process and merges and evaluates information.

Peter Reber's first task is therefore to create the requirements management plan. He could also organize the creation of the plan by somebody else. In our case, due to his vast experience, Peter creates the requirements management plan himself. He can, of course, request help from the many requirements engineering experts available and should in any case agree the plan with these experts. It may be the case that the experts have special requirements for requirements management. In this handbook, we accompany Peter and his team through their tasks step by step.

1.3 Goals and Benefits of Requirements Management

The goal of requirements management is to manage requirements and other artifacts related to requirements (e.g., interview logs and the customer requirements specification) in such a way that the requirements can be systematically scanned, grouped, evaluated, changed, and tracked with reasonable effort. Requirements management thus attempts to meet the needs of many different stakeholders simultaneously. The needs are essentially dependent on the specific project context. They differ, for example, in projects for customer-specific software or product development compared to internal projects performed by the IT department.

Amongst other things, requirements management provides answers to the following questions, based on the techniques given in parentheses:

- What different types of requirements are there? (Requirements landscape)
- To what levels of detail are requirements documented? (Requirements landscape)
- Which requirements have already been accepted? (Assignment of attributes)

- Which requirements come from which source? (Assignment of attributes)
- Which requirements are urgent and important and therefore candidates for the next release? (Evaluation and prioritization)
- Which requirement generates costs that are too high with too few benefits? (Evaluation and prioritization)
- Which requirements belong to a specific software baseline? (Version management)
- Which version of the requirement was implemented in the system? (Versioning)
- Who was the last person to change the requirement and why did they change it? (Versioning)
- Which technical component belongs to which requirement? (Traceability)
- Which test cases belong to which requirement? (Traceability)
- Which requirement is part of the system/product delivered? (Traceability)
- How do the two variants of the product differ? (Variant management)
- What proportion of the requirements has already been implemented and tested? (Reporting)
- How long does it take on average for a change request to be implemented? (Reporting)
- Has the requirements engineering process been improved by a specific measure? (Reporting)

In principle, requirements management is worthwhile not just for larger projects but also for small projects. However, for small projects with a low level of complexity, the core team often performs requirements management in their head, yet still knows exactly when which requirement was changed and why.

Requirements management is more important and more difficult [RuSo2009]…

- ... the greater the number of requirements that exist
- ... the longer the estimated lifetime of the product is
- ... the greater the number changes that are expected
- ... the larger the number of participants in the requirements engineering process is
- ... the more difficult it is to reach or involve the stakeholders
- ... the higher that the quality demands on the system are
- ... the greater the level of reuse that is to be performed
- ... the more complex the development process is
- ... the more inhomogeneous stakeholders' opinions are
- ... the greater the number of releases that will be developed
- … the more important the use of standards is for the project

Good requirements management ... [RuSo2009]

- ... increases the quality of requirements, products, and processes
- ... reduces project costs and project duration
- ... makes it easier to monitor complex projects during all phases
- ... improves communication within and among teams
- ... increases customer satisfaction
- ... reduces the project risk

Requirements management is a complex task: every stakeholder should be able to access up-to-date information at all times and should also be informed about changes that affect them, but without overloading them with unnecessary information. This should also apply even if the stakeholders are spread around the world and when contact persons change. At the same time, however, data protection regulations must be complied with and each person must be able to access only that information that they need to do their job. The data collected by requirements management leads to a certain complexity—not solely as a result of the pure quantity of requirements and associated information, but also from mutual dependencies between requirements and the temporal dimension of versions and requirements baselines.

Requirements management simplifies requirements engineering:

- Structuring of requirements and the requirements document (e.g., via assignment of attributes, sorting, and filtering)
- Standardization of terminology (e.g., via a glossary)
- Definition of clear processes and work steps to be performed (e.g., in the change process)

For Peter Reber, therefore, there is no question of the benefits of requirements management. The expectation is that the large number of stakeholders and requirements engineers will produce a large number (perhaps even too large) of change requirements. The requirements must be aligned with one another and the most relevant requirements selected for the first release. Of course, like most projects, this project is under pressure from a time perspective and has only a fixed budget that has been defined in advance.

1.4 The Requirements Management Plan

The necessity of a project management plan [PMI2013] is something that has been known in project management for years. This plan describes how a specific project is to be executed. In addition to the project schedule, the plan contains the planning, details of how, for example, risk management is to be performed, how communication and discussions should take place, who is responsible for what, etc. The plan enables the project manager to bring all project team members to the same level of information about how work is to be performed within the project. The plan also provides the opportunity to control the process.

The requirements management plan (RMP) is very similar to the project management plan. Amongst other things, the requirements management plan describes the planning for how the requirements engineering process is to be set up, who is responsible for what tasks, which requirements are to be documented and how they are to be documented, how these requirements are to be managed, whether tools are to be used and, if so, which tools. In brief, the requirements management plan describes all aspects that must be considered in requirements engineering and requirements management for a new development or for the continued further development of a product, for example. The requirements management plan thus describes the framework for the entire requirements engineering process.

In the following chapters, we describe the main content of requirements management and in each chapter, we point out which of the aspects described should be included in a requirements management plan. Annex A also contains a template for a requirements management plan for your own use.

Note: In practice, the requirements management plan is often not an independent document but rather part of the project plan, the configuration management plan, or other specification documents for the development process. The structure of the requirements management plan in the annex should essentially give you a framework for this topic.

1.5 Relevant Standards

To ensure that software and technical systems are developed to a high quality in such a way that they can be traced and repeated, various standards have been developed. These standards describe the activities to be performed, the artifacts to be created, and the techniques to be applied. The standards universally recognize requirements management as making an important contribution to ensuring the quality of results. The standards therefore also make statements about the execution of requirements management. However, the statements of the various standards are not necessarily consistent and compatible with one another. For example, the standards use different terms for one specific artifact and suggest different chapters respectively.

Some important standards that cover the entire software or system development process, and thereby also make statements about requirements engineering and requirements management, are the following:

- The process capability maturity model integration, CMMI (Version 1.3) [SEI2010], considers, amongst other things, the processes "requirements development" and "requirements management", whereby some of the assigned objectives differ significantly from the definitions of the IREB.
- ISO 9000/ISO 9001 [ISO 9000] is a standard for quality management in organizations. ISO 9001:2008 ("Quality Management Systems - Requirements") defines minimum requirements for a quality management system and describes, for example, requirements for product realization as well as measurement and improvement and thus addresses topics such as identifiability or traceability of requirements (see Clause 7.5.3, *"Identification and Traceability"*).

- ISO/IEC 12207:2008 [ISO12207] and 15288:2008 [ISO15288] (*"Software life cycle processes"* and *"Systems and software engineering - Systems life cycle processes"*) define all processes for systems and software development. The tasks "*System requirements analysis*" and "*software requirements analysis"* from ISO/IEC 12207 and "*Stakeholder Requirements Definition Process"* and "*Requirements Analysis Process"* from ISO/IEC 15288 cover the activities of requirements engineering and requirements management.

  IEC 61508 [DIN61508] ("Functional safety of safety-related electrical/electronic/programmable electronic systems") deals with the definition of requirements for the functional safety of systems and their implementation, including quantitative safety assessments. Particular attention is given to the topic of traceability. The standard defines safety integrity levels (SIL) 1 to 4, which describe the risk.

  The higher the level, the greater the potential risk, and thus the greater the requirements for system reliability.

- SOX (Sarbanes-Oxley Act) [USCo 2002] is a US federal law in response to accounting scandals which is intended to improve the reliability of reporting by companies listed on the public capital market in the USA. In essence, the core of the Sarbanes-Oxley Act is about knowing who made what changes when, and thus also relates to the core tasks of requirements management.

Requirements engineering and requirements management can be found in the following standards, for example:

- VDI guideline 2519 sheet 1 - The procedure for the creation of customer requirements specifications/system requirements specifications [VDI2001] is the German standard for describing customer requirements specifications and system requirements specifications.
- IEEE 830-1998 (*"Recommended Practice for Software Requirements Specifications"*) [IEEE830] defines terms for requirements engineering and requirements management—in particular, quality properties of requirements and the chapters involved in a specification (*"software requirements specification"*). Many of these definitions have also been included in the CPRE Foundation Level [IREB2015].
- ISO/IEC/IEEE 29148:2011 ("*Systems and software engineering – Life cycle processes – Requirements engineering"*) [ISO29148] defines quality properties and attributes of requirements and recommends iterative handling of requirements over the entire lifecycle.
- IEEE Standard 1233 "*Guide for Developing System Requirements Specifications"* [IEEE1233] describes the development of requirements and specifications and the management of these in the entire product development. The standard describes the collection and definition of requirements, change management, and the organization of requirements in a project.
- ISO/IEC 14102:1995 ("*Evaluation and Selection of CASE Tools"*) [ISO14102] describes requirements for CASE tools—that is, tools for computer-aided software development—but can also be used to select requirements engineering and requirements management tools.

- ISO/IEC 25010:2011 ("*Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*") [ISO25010] describes two quality models for non-functional requirements: one for "*Quality in use*", and one for the product quality. These quality models can be used to collect and specify non-functional requirements of software and computer systems in a standardized way.
- ISO 29110 ("*Lifecycle process standard for Very Small and Medium Entities (VSME)*") [ISO29110] describes a system lifecycle, including requirements engineering, for small and medium-sized units, so for projects with less than approx. 25 persons.
- The European standard ISO 9241 [ISO9241], which is also recognized as a DIN standard, describes guidelines for human-computer interaction—specifically, both a list of quality requirements that a user-friendly software must contain, and the development and testing process for such software.

There are also industry-specific standards, such as DO 178 B/ED-12B and DIN EN 14160 for aviation, IEEE/EIA Std. 12207:1998 for the military, FDA-535, FDA-938, and EN62304 for medical engineering, EN 50128 for railway technology, or ITU X.290-X.296 (ISO/IEC 9646-x) or ETSI ES 201 873-x (TTCN-3) for telecommunications.

These standards do not apply automatically and above all, they do not apply simultaneously, as they are not completely compatible with one another. Each company and each project selects the appropriate standards which they then apply in the original or an adapted form. Sometimes, the customer requires compliance with a specific standard.

In addition to the standards and guidelines referred to above, company-specific standards of the software manufacturer or the customer must also be observed. In turn, these can be developed based on public standards and can contain aspects of requirements engineering and requirements management.

Due to its comprehensibility and the fact that the contents are closely related to practice, for his work, Peter Reber uses this IREB Handbook for Requirements Management, which has been developed from standards. The company guidelines for the implementation of IT projects and the new corporate identity also apply.

1.6 Literature for Further Reading

For further reading, we recommend the standards detailed in the previous section.

Further definitions can be found in the IREB Glossary [Glin2014].

2 Requirements Information Model

In this chapter, we look at how you can define the different aspects of your project-specific requirements landscape and describe them with a requirements information model.

When documenting requirements, we repeatedly encounter a number of basic questions that have nothing to do with the specific content of requirements, but which must be defined at an early stage, for example:

- What are the different types of requirements that exist and that have to be considered?
- How can requirements be classified according to their solution dependency?
- How should these requirements be documented and presented?
- To what level of detail must the requirement be described?

These questions should generally be clarified at the beginning of the requirements engineering process. However, projects do not always run as ideally as desired. Sometimes, projects are taken over from third parties who have not made any specifications of this type. Sometimes, the constraints at the beginning of a project do not allow you to create a requirements management plan or to think about the structure of requirements, and therefore, initially you merely "collect" requirements without classifying them. What is important, however, is that at a given time—as early as possible—you plan your requirements landscape. As the requirements manager, as well as being responsible for managing the requirements artifacts, you are responsible for managing the activities in the requirements engineering process, which means planning, monitoring, and controlling these activities appropriately (definition 1-1).

Definition 2-1: Requirements artifact according to [Pohl2010]: "A requirements artifact is a documented requirement".

Therefore, whenever we refer to requirements artifacts in the subsequent chapters, we are referring to a documented requirement. In contrast, when we use the more general term "artifact", we are referring to documented artifacts at different development levels: for example, test cases, architecture descriptions, etc. (see IREB Glossary [Glin2014]).

Definition 2-2: Requirements landscape*: A requirements landscape is a specification of the:*

1. *Classification to be used for the types of requirements*
2. *Classification to be used for the independence of the requirement from a solution*
3. *Required levels of abstraction (detailing levels) for each type of requirements artifact*
4. *Forms of presentation to be used for each type of requirements artifact*

In the following sections, we look at the different dimensions of the requirements landscape and provide information about the points in the landscape that a requirements manager has to think about in order to create a **"requirements information model"** (RIM) to describe the requirements landscape.

2.1 Basic Principles (Classification of Requirements)

When we talk about requirements, we are often talking about the different forms of a requirement. A requirement can be differentiated, for example, by its level of detail. Requirements can be very detailed and describe a specific function, but they can also be a very abstract requirement for the overall system. Apart from the level of detail, requirements can also differ in the type of content they contain. For example, a requirement can describe the quality of a system (e.g., correctness) or require functional properties. Requirements can also differ in their independence from a solution; they can be generic product objectives, for example, in contrast to requirements for the data structure. In principle this is nothing new, and it has already been mentioned in the Certified Professional for Requirements Engineering – Foundation Level [IREB2015].

Therefore, for better orientation and to enable you to construct your requirements landscape more specifically, we want to classify requirements according to the following dimensions:

- the type of requirement
- the independence of a requirement from a solution
- the level of detail (or abstraction level) of a requirement

The dimensions referred to above are orthogonal to one another, which means that even at a level with a high level of detail, there are goals, for example—that is, requirements that are independent of a solution.

2.1.1 Classification by Type of Requirement

The following question is often raised in requirements management: "What types of requirements have to be considered during collection and documentation?". This question can be answered with a quick look back at the Certified Professional for Requirements Engineering – Foundation Level [IREB2015].

- **Functional requirements**: Functional requirements describe the functionality that the planned system should provide. These requirements describe what the planned system should be able to do—for example, which data a customer needs for authorization at an ATM to authorize a withdrawal of cash.

Definition 2-3: Functional requirement according to [PoRu2011]: *"A functional requirement is a requirement concerning a result of behavior that shall be provided by a function of the system."*

- **Quality requirements**: Quality requirements describe desired qualities of the planned system and thereby influence the system architecture. This class describes, for example, requirements with regard to the reliability, security, scalability, or performance of the planned system or individual functions.

Definition 2-4: Quality requirement according to [PoRu2011]: *"A quality requirement is a requirement that pertains to a quality concern that is not covered by functional requirements."*

- **Constraints**: Constraints are organizational, legal, or technical specifications (usually limitations) for the realization of the planned system. They can be a wide variety of conditions, starting with time-based specifications for implementation, through to specific technology specifications for implementation.

Definition 2-5: Constraint according to [PoRu2011]: *"A constraint is a requirement that limits the solution space beyond what is necessary for meeting the given functional requirements and the quality requirements."*

The characteristics of the different types of requirements and their further categorization are discussed in detail in literature (see [Pohl2010]; [PoRu2011]; and [Eber2012]).

In addition to the classification of requirements introduced above, requirements engineering literature contains further classifications for which an association to the three types of requirements referred to above can be established, for example:

- [RuSo2009]: functional requirements, technology requirements, quality requirements, requirements for the user interface, requirements for other components delivered, requirements for activities to be performed, legal contractual requirements
- [WiBe2013]: business requirements, business rules, constraints, interface requirements, features, functional requirements, non-functional requirements, quality requirements, system requirements, user requirements
- [RoRo2014]: functional requirements, non-functional requirements, constraints
- [Youn2014]: business requirements, user requirements, product requirements, environment requirements, system requirements, functional requirements, performance requirements, interface requirements, etc.

However, it is not the classification you use to differentiate between requirements that is the decisive factor for good requirements management, but rather the awareness of the existence of different types of requirements that have to be considered to describe the desired change or the planned system completely.

None of the requirement type classifications presented in this chapter is a generally valid standard. The point of this information is merely to illustrate the wealth of types of requirement that have become established in recent years.

2.1.2 Classification according to the Dependence of Requirements on a Solution

Regardless of the type of a requirement, requirements often demonstrate very different levels of dependence on a solution. Therefore, in descriptions of requirements, we often find a mix of:

- Goals to be achieved with a system (usually an almost solution-independent description of the goal to be achieved—for example, easier and more secure access to cash for all bank customers)
- System processes to be supported by a system (usually only an indirect reference to a solution by means of technical specifications for the desired system behavior or process flow—for example, description of a process for authentication at an ATM)

- Specific properties and characteristics that a system should fulfill (usually direct dependence on a solution by means of technical and operational specifications for the desired system—for example, unique specification of the relevant data for authenticating a user at ATMs) .

To consciously differentiate between the requirements with a different solution dependency, we recommend that you explicitly classify requirements artifacts dependent on the reference to a solution or the dependence on a solution—for example, as goals, scenarios, and solution-dependent requirements (see [Pohl2010]).

Goal-oriented descriptions (goals):

- Goal-oriented descriptions document (by means of goals) the intention of the system without addressing the implementation (solution). They are therefore the most abstract form of a documented requirement and require, for example, that a customer must be able to withdraw money from their account in any city—regardless of how that can be implemented.
- Goals can be described, for example, in natural language in pure text form, with and-or trees, or with independent notations such as i*. Regardless of the form of presentation, the main point of goals is to achieve a system understanding to thus recognize the required added value of the planned system.

Scenario-oriented descriptions (scenarios):

- By way of example, scenario-oriented descriptions document (using scenarios) the desired process to be supported from the user perspective (sometimes also from the system perspective). Scenarios thus describe possible sequences of interactions to fulfill one or more goals. They often supply the context required for the requirement by, for example, describing the process of withdrawing money at an ATM. Scenarios therefore usually cover several atomic requirements.
- Scenarios can be described, for example, with structured templates (e.g., use case templates) in pure text form as a type of story, or based on models using activity diagrams, business process models (e.g., BMPN), sequence diagrams, etc.—see IREB CPRE Advanced Level "Requirements Modeling" [CHQW14].

Solution-oriented descriptions (solution-based requirements):

- Solution-oriented descriptions document (using solution-based requirements) specific requirements for, for example, the functionality or performance of a system or individual components. They describe the data, functions, system behavior, statuses, and quality required to fulfill the goals and to implement the scenarios. They must therefore be understood as "classic" requirements which must result in a solution—for example, "After successful authorization, the system must give the customer the opportunity to withdraw cash amounts of between €50 and €500. The selected amount must be divisible by €10 to enable disbursement."
- Solution-oriented requirements can be described in either natural language as classic textual requirements, or via model-based notations (e.g., UML). Solution-oriented requirements generally cover all requirements for the classic system views: data, functions, and behavior of the planned system. They are therefore the most specific descriptions, see IREB CPRE Advanced Level "Requirements Modeling" [CHQW14].

2.1.3 Levels of Detail for Requirements—Twin Peaks Model

In practice, detailing requirements is only rarely a strict, sequential (waterfall) process that begins with rough requirements artifacts and then turns these step by step into requirements artifacts at a fine level of detail which, in the next step, are then used as the basis for the creation of a system architecture. In real life, at the beginning of the process, there are usually requirements with very different levels of detail on the one hand, and on the other hand, an early interaction between requirements and system architecture, which means that there are mutual influences between the system architecture or solution decisions and the requirements.

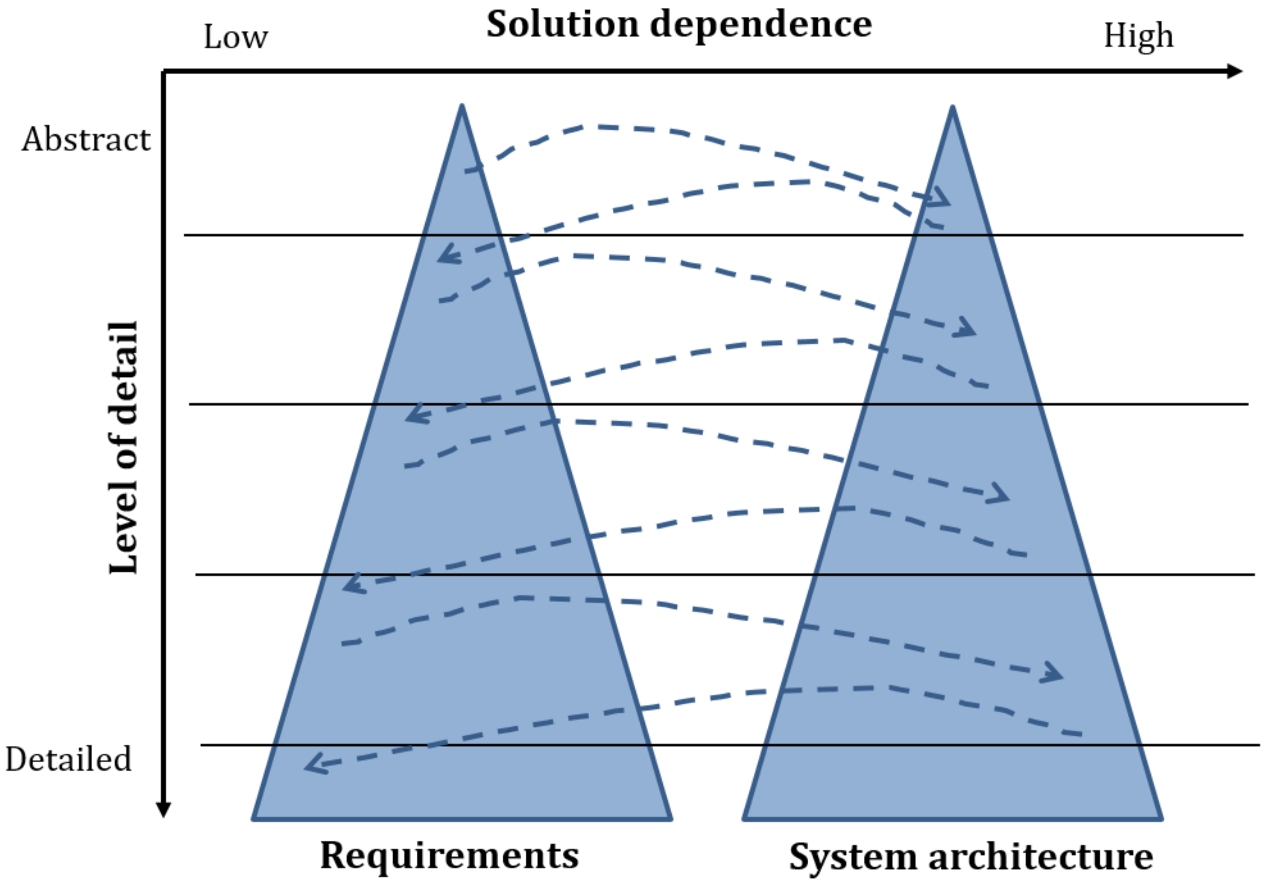


Figure 1: Twin peaks model

Figure 1 illustrates this relationship in the twin peaks model [Nuse2001]. The vertical axis represents the level of detail of the requirements or the system architecture, while the horizontal axis represents the solution dependency, that is, the increasing alignment from problem description to implementation. The figure shows that an increasingly detailed requirement description (on the left) is developed iteratively in parallel with an increasingly detailed system architecture (on the right), and that the description and the architecture supplement each other. Although the figure shows, for simplification purposes, the same levels for the detailing of the requirements and for the system architecture, different levels of detail are in fact possible. The intention behind the figure is essentially to show the necessity for different levels of detail for documenting requirements (see also [BBHK2014]).

Unfortunately, there is no general agreement on the number of levels of detail that are required and useful on the requirements side. The required level of detail for requirements depends on many factors, as the following examples show:

- **System context and domain**: If only a small change is to be executed within a well-known system environment, a lower level of detail may be necessary than for a completely new development.
- **Expertise and proximity of the stakeholders**: In a project environment with experienced and skilled requirements engineers, architects, developers, and testers, a lower level of detail is often required than would be the case for a distributed project team and development team with supplier relationships.
- **Accepted levels of freedom in the implementation**: In a project environment in which the client and stakeholders are thinking purely in terms of results, and the way in which the result is achieved is irrelevant (e.g., representation of account movements), the solution space has to be less restricted with requirements details than in an environment where security is critical (e.g., authorization on login).

For the reasons listed above, the number of detail levels or the level of detail of the requirements must be defined on an individual basis (e.g., on a project-specific basis). This level of detail can differ even within a project depending on the specific system object under consideration. In principle, a requirement should be detailed to the extent that:

- All stakeholders have reached a **common understanding** of the requirements and it is clear to everyone exactly what is required. This is particularly true for the group of stakeholders who have to implement the requirement.
- The remaining degrees of freedom for the design of the solution are so small that further precision would generate more costs than benefits. This means that the requirements must be detailed to an **accepted residual risk** that, due to the remaining degrees of freedom, an undesired solution will arise.
- The requirements are specified to the extent that the subsequent solution is **clearly verifiable** (testable) by means of the requirements—that is, the solution can be accepted based on the specification.

Note: Three levels of detail have proven to be worthwhile in many projects—even though these levels often bear different names, this has proven to be a practicable level of detail (e.g., product requirement level, user requirement level, system requirement level).

Literature also offers a number of suggestions for structuring requirements at different levels of detail.

- [WiBe2013] suggests classifying requirements as *"business requirements"*, *"user requirements"*, and *"system requirements"*. [Eber2012] proposes detailing requirements via a classification according to market, product, and component requirements.
- [RuSo2009] describes five levels of detail, from the rough overall intention with its goals, through to technical specification and separation into hardware, software, and other components.
- [PHAB2012] defines three levels of details for the domain of embedded software-intensive systems (embedded systems): *"functional layer"*, *"logical layer"*, and *"technical layer"*.

- [BBHK2014] also describes three levels of detail, *"system layer"*, *"function group layer"*, and *"hardware/software layer"* for detailing requirements for software-intensive embedded systems.

2.2 Forms of Presentation for Documenting Requirements

There are a number of different forms of presentation (or rather, forms of description) for documenting requirements. The form of presentation used to document requirements depends on various factors, for example:

- Purpose of the documentation (e.g., formal check, review, discussion)
- Recipient of the information (e.g., product manager, architect, tester, developer)
- Classification of the requirement (e.g., use case, performance requirement)

The form of presentation is also dependent on the experience and the personal "preferences" of the person documenting the requirement.

In the following, we differentiate between the following forms of presentation for documentation:

- **Textual presentation of requirements using natural language:** Natural languages (e.g., German, English, Spanish) are languages which are used on a daily basis to document and exchange information. In textual descriptions, we find the following forms of presentation for requirements, for example:
  - Pure prose
  - Phrase templates (e.g., "THE SYSTEM must/should/will PROCESS VERB")
  - Structuring templates (e.g., to describe use cases)
- **Model-based presentation of requirements using modeling languages**: In comparison to natural languages, modeling languages are languages created artificially. Modeling languages for documenting requirements include:
  - Unified Modeling Language (UML)
  - Business Process Model and Notation (BPMN)
  - Event-driven Process Chain (EPC)
  - System modeling language (SysML)
  - Entity relationship model (ERM)
  - Petri nets
- **Formalized presentation of requirements with formal languages**: Formal languages are also artificially created languages. With formal languages, the focus is on a description that is free of contradictions, rather than on communication. Formal languages include:
  - Mathematical-algebraic descriptions
  - Set theory forms of description
  - Logical descriptions and operators

To make the documentation in the requirements engineering process—and the management and maintenance of the documentation as part of requirements management—manageable, as the requirements manager, you should define, as early as possible, which type of requirement is to be persisted, with what solution dependency, at what level of detail, and with what form of presentation.

You should also define, at an early stage, the language in which textual and model-based requirements are to be documented to avoid unnecessary duplicated effort for subsequent translation.

Note: The language of the requirements is usually determined by the project language or the national language of the stakeholders and suppliers involved. However, you can also define, for example, that requirements from departments are documented in the national language (e.g., in German for projects in Germany) to achieve greater involvement and acceptance, and that system requirements that have to be implemented by a supplier are documented in English—that is, the documentation language can be different depending on the level of detail.

2.3 Describing a Requirements Landscape with a Requirements Information Model

In this chapter, we explain how you can describe a requirements landscape and document it with a **requirements information model** (RIM). As explained in definition 2-2, a requirements landscape defines the following dimensions:

- *Classification to be used for the types of requirements*
- *Classification to be used for the dependence of requirements on a solution*
- *Required levels of detail for each type of requirements artifact*
- *Forms of presentation to be used for each type of requirements artifact*

You can use a tabular list to describe the requirements landscape. Here, you document the different dimensions of the requirements landscape (type of requirement, solution dependency, levels of detail, form of presentation). Table 1 shows an example of a requirements landscape.

		Solution Dependency		
Level of detail	**Requirement type**	**Low (Goal)**	**Medium (Scenario)**	**High (Solution-Based Req.)**
Level of detail 1: Business level	Constraint	Business goal (textual)	Not relevant	Not relevant
	Quality requirement	Service quality (textual)	Not relevant	Not relevant
	Functional Requirement	Not relevant	Business process (BPMN)	Business rule (textual)
Level of detail 2: User level	Constraint	Usability goal	Not relevant	Not relevant
	Quality requirement	Not relevant	User interface (mock-up)	Not relevant
	Functional Requirement	Not relevant	User use case (use case diagrams, templates)	User requirement (textual, ER models)
Level of detail 3: System level	Constraint	Not relevant	Not relevant	Interface guidelines (textual)
	Quality requirement	System quality goal (textual)	Not relevant	System quality (textual)
	Functional Requirement	Not relevant	System use case (MSC, AD)	Interface requirements (textual, MSC)

Table 1: Example definition of a requirements landscape

Column 1 contains the description of the levels of detail (here, Level of detail 1: Business level, Level of detail 2: User level, and Level of detail 3: System level). Column 2 contains the classification by requirement type (here, constraint, quality requirement, and functional requirement). Columns 3–5 describe the dependency of the requirement on the solution by classification into goals, scenarios, and solution-based requirements. This table therefore describes all combinations theoretically possible for types of requirements artifacts.

Via the cells, you can select which types of requirements artifacts are relevant or not relevant for your specification. For the relevant requirements candidates (**requirements class**), you can now define the desired forms of presentation for each artifact type for your requirements landscape (e.g., user level, solution-based description for functional requirements is via entity-relationship models or in text form). Here, you can also assign a dedicated, company-specific designation (e.g., user level, solution-based description for functional requirements = user requirements) to the selected requirements candidates.

When defining the requirements landscape, you always have to balance the benefits that more extensive requirements documentation would provide against the costs that would be incurred (see [Glinz 2008], [Davis 2005]). It may well be the case, for example, that you describe requirements only at two levels of detail, based on scenarios and solution-based requirements.

However, the requirements landscape should be defined explicitly—and not just by chance—and should, for example, be documented in the requirements management plan so that it is clear to all the stakeholders which types of requirements artifacts are to be documented and at what level of detail. A tabular description of the requirements landscape makes sense here (see Table 1). In addition to the tabular description, it also makes sense to create an information model in the form of an entity relationship diagram or a class diagram to describe relationships between types of artifacts at one or different levels of detail. In the following, we refer to this descriptive information model as the **requirements information model** (RIM).

In addition to the specifications made above with regard to which type of requirement is documented, with what solution dependency, at what level of detail, and in what form, further aspects can be added to the requirements information model:

- Which attributes are used for which types of artifacts? (See Chapter 3)
- Which views are supported? (See Chapter 3)
- Which evaluation criteria are planned for requirements? (See Chapter 4)
- Which roles are responsible for maintenance and change? (See Chapter 5)
- Which traceability relationships between requirements artifacts and upstream and downstream artifacts are documented? (See Chapter 6)
- How are variants of requirements documented? (See Chapter 7)

With this information, the requirements information model makes up a significant part of the requirements management plan (RMP). Therefore, the requirements information model must be accessible for all stakeholders to view at any time.

Figure 2 shows the requirements information model of Peter Reber. Based on Table 1, Peter has divided his requirements information model into three levels of detail (business level, user level, system level). From the 27 types of requirements artifacts theoretically possible, Peter has selected 13 to specify the requirements for the new bank system.

Goals at the business level (level of detail 1) are differentiated into business goals, business rules, service quality, and business processes.

Here, business goals usually describe constraints that have to be taken into account in the implementation—for example, the planned start date, company guidelines, etc.

Business processes can be assigned to the category of scenarios and mainly describe functional requirements. Peter uses BPMN to describe the business processes.

Business rules describe, at a high level, functional requirements that have to be considered in the subsequent work steps and that restrict the solution space. For Peter, business rules include, for example, limits for the amount for online transfers per day.

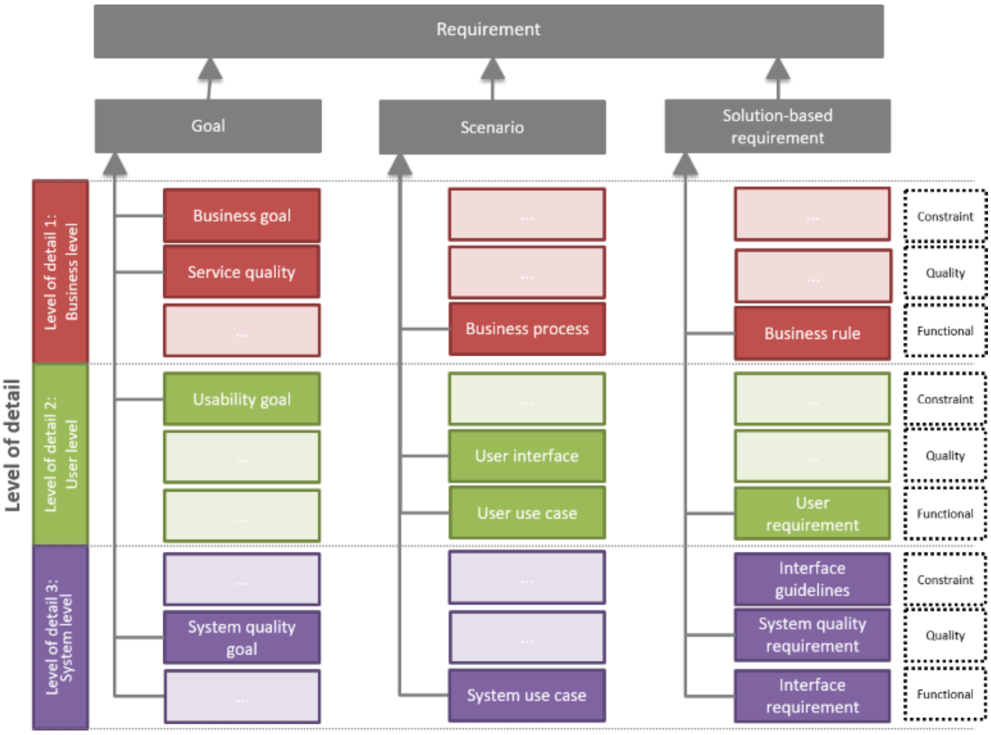


Figure 2: Example of a requirements information model

At the user level, Peter wants to document usability goals, requirements for the user interface (GUI), as well as user use cases and user requirements. User use cases should be described with use case diagrams and templates, for example.

User requirements and user interface requirements are described from a solution-based view. Mock-ups, textual descriptions, or ER models should be used here.

At the IT level there are interface guidelines, system quality goals, system use cases, interface requirements, and system quality requirements. The forms of presentation at system level are based more strongly on IT development, which means that here, in addition to textual requirements, model-based notations such as activity diagrams and message sequence charts (sequence diagrams) are used.

In the present model, only the solution dependency dimension has been described at different levels of detail. The requirement type dimension (constraint, quality requirement, functional requirement) is presented only structurally on the right-hand side in this model. The corresponding form of presentation has been completely ignored in the requirements information model because, when more than two dimensions are presented, the model soon becomes unclear. If a modeling tool is used, here you may be able to offer different views of the information model to avoid overloading a model view.

A further option for including more details in a requirements information model is to use annotations for a class in order to describe the different dimensions of the class (see Figure 3).

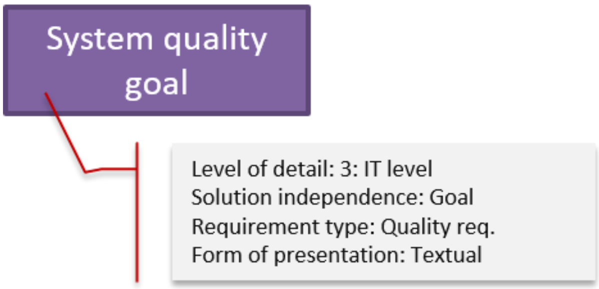

System quality goal
Level of detail: 3: IT level
Solution independence: Goal
Requirement type: Quality req.
Form of presentation: Textual

Figure 3: Example requirements information model with annotation of the dimension details

As an alternative to the requirements information model, you can use a tabular description (see Table 1) for the form of presentation or to assign the levels of detail.

To check the requirements information model, you can apply the following control questions:

- **Check for formal completeness**: Does the requirements information model clearly show, for each class of requirements, which requirement type it contains, how dependent this requirements class is on a solution, what level of detail the requirements class exists at, and with which form(s) of presentation it is documented?
- **Check for content relationships**: Does the requirements information model clearly show which levels of detail exist and how they are connected? Is it clear how requirements at the different levels of detail are dependent on one another?
- **Check for adequacy**: Are all the selected requirements classes appropriate to document sufficiently detailed, complete, and consistent requirements so that the subsequent activities (e.g., development and testing) can fully complete their tasks?

2.4 Content for the Requirements Management Plan

When you create the requirements landscape, you create a significant first part of your requirements management plan. With the requirements landscape, you define which types of requirements artifacts you want to consider, the number of levels of detail that you want to define requirements on, and the forms of presentation that you want to use to specify types of requirements artifacts (see Table 1 and Figure 2). By describing the requirements landscape (e.g., with a requirements information model), via the requirements management plan, you can ensure that all stakeholders involved in the project have a shared understanding of the types of requirements artifacts to be used to document requirements, as well as the levels of detail and forms of presentation to be used when documenting requirements.

2.5 Literature for Further Reading

[BBHK2014] Braun, P.; Broy, M.; Houdek, F.; Kirchmayr, M.; Müller, M.; Penzenstadler, B.; Pohl, K.; Weyer, T.: Guiding requirements engineering for software-intensive embedded systems in the automotive industry. Computer Science - R&D 29(1): 21–43 (2014).

[Eber2012] C. Ebert: Systematisches Requirements Engineering. Dpunkt, 4th edition, 2012 (available in German only)

[CHQW14] Thorsten Cziharz, Peter Hruschka, Stefan Queins, Thorsten Weyer: Handbook of Requirements Modeling According to the IREB Standard, Education and Training for IREB Certified Professional for Requirements Engineering Advanced Level "Requirements Modeling" IREB, Version 1.0-3, March 1, 2014.

[PHAB2012] Pohl, K., Hönninger, H., Achatz, R., Broy, M. (Eds.): Model-Based Engineering of Embedded Systems - The SPES 2020 Methodology, Springer 2012.

[Pohl2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, 2010.

[WiBe2013] K. Wiegers and J. Beatty: Software Requirements. 3rd Edition. Microsoft Press, 2013.

3 Assigning Attributes and Views for Requirements

In this chapter, we look at how requirements management defines requirements attributes and which requirements attributes have to be used in projects. The chapter also looks at how we create, use, and change attribute schemas and views.

Definition 3-1: An attribute is a characteristic property of a unit. (From the IREB Glossary [Glin2014])

The standard ISO/IEC/IEEE 29148:2011 [ISO29148] adds the aspect of how attributes are evaluated to the definition of an attribute: "an inherent property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means."

In connection with requirements management, attributes are therefore properties of the requirements, for example, the processing status (attribute "Status"). As meta-information, attributes are not usually mixed with the requirement description. Instead, they are documented and managed separately—for example, as a separate column in a tabular list of requirements and as a separate field in a requirements database. It is not only textual requirements that can have attributes, but also elements of a UML model [CHQW14]. Despite having the same name, requirements attributes are not synonymous with the attributes of a class in a class diagram. These latter attributes are part of the content of the requirement but they are not meta-information—that is, they are not requirements attributes in the sense of this chapter. Requirements of all types and levels of details can have attributes, but sometimes they do not have the same attributes. Change requests also have attributes (see Chapter 5). Entire documents can be characterized with attributes, such as a status or a version number.

3.1 Objectives of Assigning Attributes and Examples of the Use of Attributes in Management Activities

As the definitions above indicate, in requirements management, attributes are used to categorize requirements, specifically with regard to meta-information required for release planning or management, for example. Attributes allow you to get an overview of the requirements. For extensive projects in particular, nobody has an overview of all of the requirements. In this situation, for each requirements-based activity to be performed in software engineering, attributes help you to concentrate on the important information—on the requirements defined and their relevant properties. Of course, depending on the activity, you will be interested in different extracts from the information.

Attributes of a requirement typically answer a number of important questions, for example: "Who was the last person to change a requirement and when did they do so?" or "Which requirements are planned for Release 1?" or "How much effort is Release 1 likely to incur overall?" From a practical perspective, you do not enter all of the meta-information for a requirement in one single free-text field. Each attribute is managed in a separate field.

Value lists are often specified here, and these are standardized for all requirements. For example, it is easier to evaluate the attribute "Priority" if it permits only the values "Low", "Normal", and "High" or another grading or values list. If this were a free-text field, comments such as "Quite important", or "Mr. Miller said the requirement is important" could be entered here. This type of content does not particularly support the differentiation between the most important and the less important requirements. This differentiation is much easier if, by simply using filters, you can display a list of all requirements categorized as "High". Which format and which attribute values make most sense for the priority, for example, depends, amongst other things, on which views and decisions are to be supported by the priorities (see Chapter 4).

The objective of assigning attributes to requirements is to enable team members and other stakeholders to document and evaluate information on requirements in a structured manner as part of the requirements engineering process [Pohl 2010].

You should think carefully about which attributes are required and the values permitted for the attributes at the very beginning of a project, as it is not easy to change an attribute schema retrospectively (see Section 3.5). Unfortunately, there is no one single attribute schema that fits everywhere and every situation ideally. The decisive factor is always which attributes you want to subsequently evaluate and how you want to do this.

Various authors propose different compositions of attributes which, in their experience, have proven practical. According to the CPRE Foundation Level, the attributes in Table 2 and Table 3 are some of the important requirements attributes [PoRu2011].

Attribute Type	**Meaning**
Identifier	Short, unique identification of a requirements artifact in the set of requirements under consideration
Name	Unique, characteristic name
Description	Describes the content of the requirement in compact form
Version	Current version of the requirement
Author	Designates the author of the requirement
Source	Designates the source or sources of the requirement
Justification	Describes why this requirement is important for the planned system
Stability	Designates the probable stability of the requirement here, stability is the scope of changes expected for this requirement in the future; possible differentiation: "Stable", "Volatile"
Criticality	In the sense of an estimation of the level of damage and probability of occurrence
Priority	Designates the priority of the requirement with regard to the selected features for prioritization—for example, "Importance for acceptance on the market", "Order of implementation", "Damage or opportunity costs of non-realization"

Table 2: Frequently used attribute types [PoRu2011].

Attribute Type	**Meaning**
Owner	Designates the person, stakehold-er group, or organization(al unit) responsible for the content of this requirement
Requirement type	Designates the type of requirement (e.g., functional requirement, quality requirement, or constraint) dependent on the differentiation schema used
Status of the content	Designates the current status of the content of the requirement—for example, "Idea", "Concept", "Detailed content"
Status of verification	Designates the current status of the validation—for example, "Unchecked", "In evaluation", "Checked", "Failed", "To be corrected"
Status of agreement	Designates the current status of the agreement—for example, "Not agreed", "Agreed", "Conflicts"
Effort	Forecast/actual implementation effort for this requirement
Release	Name and/or number of the release in which the requirement is to be implemented
Legal liability	Indicates the degree of legal liability of the requirement—for example, "Must", "Recommended", and "Optional"
Cross-references	Designates the relationships to other requirements: for example, if you know that the prerequisite for realizing this requirement is the prior realization of another requirement (see Chapter 1)
General information	In this attribute, you can document any information considered relevant for this requirement; for example, if agreement to this requirement is planned for the next meeting with the client

Table 3: Frequently used attribute types [PoRu2011].

The list above contains all of the attributes named in the standard ISO/IEC/IEEE 29148:2011 [ISO29148], as well as further attributes.

To document traceability, in addition to documenting the source, it would be useful to document in a further attribute the technical component in which a requirement is implemented (attribute "Technical component") and the test cases used to test the requirement (attribute "Test cases") (see Chapter 1).

[Pohl2010] recommends a further attribute schema with seven attribute categories. This schema differentiates between the following categories: identification, context relationships, documentation aspects, content aspects, agreement aspects, validation aspects, and management aspects. Each of the categories mentioned contains a number of possible attributes.

- *Identification:* these are the attributes that allow an attribute to be identified. They include the ID and the name, which should describe the content of the requirement as meaningfully as possible.
- *Context relationships:* these attributes document the relationships between the requirements and the context—for example, the source, the justification, the person responsible, and any stakeholders affected and with whom changes to the requirement must be agreed.

- *Documentation aspects:* here you define the form of presentation for specifying a requirement (free text, UML model, text template, etc.), a link to a document that contains the specification rules, and the validation status of the requirement (documentation) (e.g., unchecked/in evaluation/partially checked/checked/to be corrected/released).
- *Content aspects:* these attributes document and classify the content of the requirement. In particular, this includes the description of the requirement, but also the type of requirement, comments from the person who created the requirement, the status of the content (idea/rough content/detailed content) and cross-references to other development artifacts (traceability relationships).
- *Agreement aspects:* these attributes document the agreement amongst the stakeholders—for example, an agreement status (not known/conflicts/in agreement/agreed), a validation status for the agreement (unchecked/in evaluation/partially checked/checked/to be corrected/released), and one free-text field each for recognized conflicts and decisions.
- *Validation aspects:* the validation checks the quality of a requirement with regard to the three dimensions of content, documentation, and agreement. Here you can document the following: compliance with initial criteria for validation (i.e.: can validation start?), techniques for validation, the current validation step, and the overall status of validation (unchecked/in evaluation/partially checked/checked/to be corrected/released).
- *Management aspects:* these attributes document the status of a requirement and other management information. This information includes stability, criticality and priority, legal liability, and further status information. It also includes the author of the requirement, the version, change history, system release, and expected and actual effort.

Practical tip: To avoid subsequent changes as far as possible, think precisely at an early stage about which attributes your attribute schema should contain. Add only those attributes to the attribute schema that fulfill two criteria: (1) You are sure that the person responsible will maintain this attribute during the course of their work; (2) It is clear who benefits from this attribute by evaluating it, when they benefit, and how they benefit. For the effort involved in documenting this meta-information to be worthwhile, both criteria must be fulfilled.

3.2 What Is an Attribute Schema?

Definition 3-2: "The set of all defined attributes for a class of requirements (e.g., functional requirements, quality requirements) is called an attribute schema." ([PoRu2011], Section 8.1.2)

An attribute schema describes the relevant requirements attributes for a project and/or a company [RuSo2009]. In addition to the name and definition of the attribute, the attribute schema also includes the format of the attributes (Text or number? How many characters are permitted for the attribute?) and the specification of the permitted values or value ranges.

In requirements management, providing an attribute schema (template-based) for requirements brings the following advantages [Pohl2010]:

- *Accurate and consistent definition of the required information*: A predefined schema defines which information or attributes for requirements must be entered and which values are allowed for this information.
- *Gap detection*: It is possible to detect gaps in the elaboration of requirements if certain attributes are empty.
- *Support for employee training*: Employees who have already worked with the same, or similar, attribute schemas in a previous project, for example, can quickly find the necessary information and where particular information on the requirement should be documented.
- *Finding the same information in the same place*: As all requirements within a project are documented on the basis of the same attribute schema, there is a clear specification of where which information—such as the author—can be found for a requirement.

The attribute schema is part of the requirements management plan, but not the requirements landscape. Sometimes company standards have to be observed, or the use of an official standard or industry standard is stipulated. These standards usually specify the attributes to be used and their values. This standardized attribute schema then allows cross-project comparable evaluations via requirements management (see Chapter 8).

In addition to the requirements manager, other roles in the development process and in the company use the meta-information about requirements that is contained in the requirements attributes. The project manager, for example, is regularly interested in the processing status of requirements. Therefore, when defining an attribute schema, the information needs of other stakeholders in the requirements engineering process must also be taken into account. We will look at the different roles and the information they need again later on in connection with views (see Section 3.6). Ultimately, the views required are the basis for defining the attribute schema.

3.3 The Benefits of an Attribute Schema

Attributes support a number of requirements management tasks as well as other management tasks:

- *Views:* Attributes are the basis for the definition and implementation of views in a tool (see Section 3.6).
- *Prioritization*: The respective priority of a requirement is documented in one or more corresponding attributes. Multiple attributes can be defined for different prioritization criteria. In turn, these priorities support decisions that are based on the priorities—for example, the release planning. Usually, you want to implement the most important requirements first. In turn, this importance can be dependent on multiple attributes—for example, when benefits and costs are weighed up against one another. You can find more information about prioritization in Chapter 4.
- *Project management:* The project manager is interested in the forecast (or actual) realization effort involved with each requirement. This is recorded in a corresponding attribute.

This attribute allows you to create totals, and you can therefore use it to determine the total effort for a group of requirements (for example, all important requirements or the requirements for one release). It also supports project monitoring during the course of the project. Project management reports that contain only the pure number of requirements with a specific status do not give a true picture of the status of the project because not every requirement is of the same size. However, if, in these reports, you weight the requirements according to their realization effort, you get a very good picture of the status of the entire project. This is particularly true if the status attribute of the requirements covers the entire lifecycle—that is, not just whether a requirement has been accepted and handed over to development, but also whether the requirement has been realized, whether the implementation has already been tested, whether the requirement has been released by quality assurance and deemed free of errors, whether the customer has accepted the associated functionality, and whether the requirement has been delivered.

- *Release management:* The priorities support the release definition and release management—that is, the management of the different software statuses delivered to customers. Release management is supported by a corresponding "Release" attribute. This attribute documents which requirements are implemented in which release. In many cases, a distinction will be made between the desired and the planned release in order to reflect the difference that often occurs between these two realities (see Chapter 5).
- *Risk management:* The attributes "Criticality", "Stability", and "Legal liability" support the identification and evaluation of risks associated with a requirement. In turn, this risk evaluation is relevant for the project manager and the release planning. The decisive factors for the definition of attributes to support risk management are the criteria used in risk management and the evaluations required.
- *Traceability:* Being able to trace requirements is important if, during change management, you want to be able to foresee the effects of changes to requirements or resolve conflicts between contradictory requirements. Traceability should be achieved in both directions: to the source of a requirement and to later artifacts such as technical components and test cases. However, dependencies between requirements of the same type and refinement relationships between requirements at different levels of detail should also be documented, provided the benefits of doing so justifies the effort involved. For more information about the traceability of requirements, see Chapter 6.
- *Variant management:* As part of variant management (see Chapter 7), attributes can be used to assign requirements to specific variants and product configurations.
- *Reporting:* Attributes form the basis for reports, such as an evaluation of the respective number of requirements with a specific status (e.g., "Released" or "Tested"). You can find more information about requirements management reports in Chapter 8.

3.4 Designing an Attribute Schema

The attribute schema is part of the requirements management plan. It should be defined before the documentation of requirements begins and should be agreed with all stakeholders of the requirements engineering process. Subsequent enhancements and changes are usually only possible with great effort.

To design an attribute schema for use in a specific project, we recommend the following steps, which are discussed in more detail in the subsequent sections:

1. Identify sources of attributes
2. Select the attributes
3. Define permitted attribute values and properties of attributes
4. Define dependencies between attributes and their values
5. Provide support for recording data
6. Document the attribute schema

3.4.1 Identifying Sources of Attributes

To select attributes, you have to first identify the relevant sources for attributes. Sources that can be used to select attributes include:

- An attribute schema from a similar project (for example, similar in scope, number of employees involved, etc.)
- A reference schema of the organization or another standard, as described in Section 3.1
- Organizational rules that determine, for example, which attributes must be used in all attribute schemas in all projects
- Stakeholders of the requirements engineering process

You can adopt schemas from standards or adapt them as required.

If, in a (generally larger) company, attribute schemas are defined in many different projects (e.g., through reuse and subsequent adaptation), it makes sense to define general rules for creating an attribute schema. In doing so, you can define, for example, that selecting the attribute "Stability" for an attribute schema also requires the selection of the attribute "Risk", and therefore both attributes must be present in the attribute schema for the respective project. This makes sense if, when assessing the stability of a requirement, the company also wants to evaluate in parallel how highly the risk of the requirement should be evaluated in terms of, for example, scheduling.

Furthermore, when selecting attributes for an attribute schema, a company can first define generally which attributes must always be considered in every project—for example, to enable cross-project evaluations. These attributes are then present in every project and should be declared as mandatory fields.

Another source of attributes for an attribute schema are the stakeholders of the requirements engineering process (see also Section 3.6). The stakeholders are identified first and then their needs.

Peter Reber decides to use the IREB attribute schema for his first draft (see Table 2 and Table 3).

The stakeholders of the requirements engineering process are the project managers, the business analysts, the IT security experts, the usability expert, and the Customer Advisory Board, but also legal risk management requirements such as BASEL II.

When a reference system is adopted, it is easy for unnecessary attributes to be adopted as well. Therefore, Peter Reber first asks each stakeholder individually for their needs. If an attribute contained in the reference schema is not specified, Peter will check whether anyone really needs it. It may be the case that the stakeholders have simply forgotten to mention it.

In the first step, the stakeholders name the following attributes:

- Project manager: status of and effort involved in the requirements, the release that a requirement is planned for, and the status of the artifacts, but above all, an overview (weighted by effort) of the proportion of requirements (of a release) with each specific status
- Business analyst: a priority value that measures the benefit of a requirement for the bank
- IT security expert: criticality
- Usability expert and Customer Advisory Board: priority in the sense of "benefits for the user/customers of the bank"
- BASEL II: "Author" and "Version" allow you to trace who changed what and when, "Justification" documents the reason; the attributes "Stability" and "Legal liability" are used for risk management

3.4.2 Selecting Attributes

The attributes must be selected specifically and appropriately for the project so that they are actually of benefit. This applies regardless of whether you are using a reference schema or defining a new attribute schema.

If you use a reference schema as a basis, for every single attribute, check whether you need it and if so, what for. The attribute is then adopted, adapted, or (on a project-specific basis) removed. You can also add new attributes. If you do not use a reference schema, you have to create a new attribute schema. To do this, you have to identify corresponding attributes—for example, by asking the relevant stakeholders (see step 1).

The ID (identification) of a requirement is particularly important when you are assigning attributes to requirements. It is used to identify each requirement uniquely and it is a mandatory attribute in every attribute schema. Within the company, you have to define the context in which the requirement is unique. The context can be based on the organizational structure and can define, for example, that the requirement has to be unique only within departments. Another option for delimitation can lie in technical constraints—for example, the requirement has to be unique only in the database used.

In this case, you have to define the procedure for handling requirement IDs that are exchanged between these databases, as this type of exchange can lead to ambiguous IDs (because they appear more than once).

To select attributes for an attribute schema and to evaluate whether a schema is complete, you can use the seven categories presented in [Pohl2010] as a checklist: identification, context relationships, documentation aspects, content aspects, agreement aspects, validation aspects, and management aspects (see Section 3.1). All seven categories should be covered in an attribute schema.

To define an attribute schema using the categories specified, we recommend you perform the following activities:

- For each category, check which of the proposed attributes has already been selected for the project (e.g., via the selection of a reference schema). Note that sometimes, the same attribute has different names in different schemas, or the same designation is used to describe different content. For example, the "Source" attribute in the "Context relationships" category can be reflected with an attribute "Basis" in a reference schema. To ensure that this match can be discovered, the semantics of the respective attribute must be documented and traceable.
- Systematic consideration of the individual categories and the attributes proposed there. In this activity, for each attribute proposed, the benefits of the attribute for the current project must be evaluated and checked. Only those attributes where the stakeholder who will use it is clear, and for what purpose, should be used.
- Expansion of the categories or the reference schema. When a new attribute is identified during the analysis for the selection of attributes, and this new attribute does not exist yet in the reference schema or in one of the categories, analyze whether it makes sense to expand a category or the reference schema. In this activity, you have to weigh up whether a new attribute should be added to the reference schema or a category to be used as a proposal for the definition of an attribute schema in subsequent projects. You have to estimate how likely it is that a new attribute will be used in most of the subsequent projects. If it will be used frequently, it makes sense to expand the reference schema. If it is unlikely that the new attribute will be used very often, it should be documented in one of the categories.

In processes for defining an attribute schema, it is often the case that a lot of attributes are defined initially because, for example, different stakeholders want to record and evaluate information for a requirement from different perspectives. In practice, the result is often that attributes are not filled and are not used for good reason. In this case, the attribute schema has to be adapted, which is not always easy (see Section 3.5).

To avoid this type of subsequent adjustment as far as possible, the attributes should be limited to a practical quantity that can be used. Therefore, only attributes that support a clear goal or a specific task should be used. An example of such an objective can be that the project manager wants to perform an earned value analysis.

To do this, the manager has to know, for example, the degree of completion of the project, and thus requires a calculation, weighted by effort, of the proportion of the requirements that has already been completed. Each requirement therefore needs two attributes: one "Effort" attribute and one status attribute that receives the value "Completed" as soon as the work on the requirement has been completed. For a more differentiated calculation of the degree of completion, partial degrees of completion can also be considered for earlier status values (e.g., a degree of completion of 80% if the requirement has been implemented but has not yet been tested). Of course, the formula for the degree of completion can use only status values that the attribute will actually receive. This may sound obvious, but when the attribute schema is changed, the consistency between the attribute schema and views/reports can easily be lost, even while the schema is in the process of being defined.

During the discussion with the stakeholders, it is established that they think it is unnecessarily complicated to use three different status attributes. The decision is therefore taken to use only one single attribute "Status". This attribute can adopt the following values to reflect the lifecycle of a requirement or a change: in progress, in evaluation, released, changed, rejected, deleted, implemented, tested, completed.

The stakeholders have not specified the following attributes (see Table 2 and Table 3): identifier, name, description, source, owner, requirement type, cross-references, general information. These attributes are important for the requirements manager. The different dimensions of the requirements landscape from Chapter 2 must also be taken into account, and only the requirement type is already included in the schema.

Therefore, the first draft of the attribute schema corresponds to that of the IREB, with the following differences: there is only one status attribute instead of three, and there are two different priority values that each measure the benefit for the different stakeholders: from the view of the bank and the view of the bank customers. The following requirement attributes are added to the requirement type: solution independence, form of presentation, and level of detail.

Practical tip: Less is more. If you are in any doubt, leave an attribute out initially. It is better to add a new attribute later and maintain the content than for an attribute not to be filled at all, or to be filled with nonsense or reluctantly and then possibly never used. That frustrates employees and raises doubts about the sense of other attributes.

3.4.3 Defining Permitted Attribute Values and Properties of Attributes

The attribute schema also defines the permitted values for the attributes. For example, for the attribute "Risk" or "Criticality", you can define that only the following values are permitted to quantify the risk: high, medium, low, or none. This makes sense because, firstly, it is not really possible to determine risks more precisely in advance, and secondly, this value is used only to differentiate the particularly critical risks from the less critical risks. In a risk management view, the risks can then be presented by category and different measures defined for each category: from "Bears a risk", through "Take measures", down to "Cancel project". The values must be clearly defined. For example: When is the risk of a requirement deemed to be "High"? What basis is used to measure this? The probability of a problem, the possible damage, or the product of both? How high does the risk have to be to be deemed "High"?

For example, the value "High" could be assigned only if an interruption to operations is feared. These definitions must be communicated to all parties that maintain or evaluate this status and should also be defined in the help text in the requirements management tool.

Furthermore, the attribute schema must also specify for the selected attributes whether (in each case) they are mandatory or optional attributes. Both have their advantages and disadvantages. If important attributes are not filled, reports based on these attributes will not be complete. If, for example, the risk is not entered for a critical requirement (e.g., because it is not yet clear whether the risk is "High" or even "Catastrophic"), because this field is empty, risk management will miss this requirement. In the worst case, this could mean that a risk that puts the entire project into question is missed. With online banking, for example, it may be better tactically to not introduce risky functions until the risks caused by these functions are securely under control. If the attribute "Risk" is a mandatory attribute, the person creating the requirement would have entered at least "High" and would have noted in a comment field that this requirement should be checked again and potentially given a higher value. However, the requirement would have already been detected during risk management in the category of critical requirements. On the other hand, mandatory fields can force people creating requirements to make statements too early, with these statements not being checked or corrected at a later stage. It is sometimes not possible to make evaluations at the point in time when a requirement is created. As long as an attribute remains empty, a corresponding view can show that this requirement needs to be edited and the expert team for security will take care of the requirement that has not been classified yet.

Another property that has to be defined is whether several values or only one value can be selected in the field for each requirement.

Instead of forcing the selection of a single value, you can also offer the following selection for the attribute value: "All possible values are valid" or "No value is valid". In the case of "No value is valid", you must make sure that this value is differentiated semantically from the attribute not being filled (i.e., it is not equated with the attribute not being filled), because selecting the value "No value is valid" is an intentional statement, whereas not filling the attribute allows two interpretations: either the predefined selection options do not apply, or the attribute has not been processed.

The table below shows the attribute schema for our example bank. Mandatory fields are marked with an asterisk:

Attribute	Meaning	Values
Identifier*	Short, unique identification	The bank's schema is: project code + sequential number, so in this case, OBA0001, OBA0002, etc.
Name*	Unique, characteristic name	Free text
Description*	Describes the content of the requirement in compact form	Free text
Version*	Current version of the requirement	The format of the versioning is still to be defined
Author*	Author of the requirement	Only the following persons are allowed to write requirements: Peter Reber, Martin Geldmann (business analyst), Anja Streng (IT security expert), Kirsten Uba (usability expert)
Source*	Designates the source or sources of the requirement	List of all stakeholders
Justification	Describes why this requirement is important for the planned system	Free text
Stability	Designates the probable stability of the requirement	"Stable", "Volatile"
Criticality	In the sense of an estimation of the level of damage multiplied by the probability of occurrence (risk)	"Low", "Medium", "High"
Priority for bank	Measures the benefit of the requirements for the bank	"Low", "Medium", "High"
Priority for customers	Measures the benefit of the requirements for the bank customers	"Low", "Medium", "High"
Owner	Designates the person, stakehold-er group, or organization(al unit) responsible for the content of this requirement	List of all stakeholders

Attribute	Meaning	Values
Requirement type	Type of requirement	"Functional requirement", "Quality requirement", "Constraint"
Solution dependency	Solution dependency	"Goal", "Scenario", "Solution-oriented requirement"
Level of detail	Level of detail	Level 1: Business scope Level 2: User scope Level 3: IT scope
Status*	Status in the lifecycle of the requirement	In progress, in evaluation, released, changed, rejected, deleted, implemented, tested, completed
Effort	Forecast implementation effort for this requirement in days; for requirements with the status "Completed", the actual value is entered here	Only whole or half values
Release	Number of the release in which the requirement is to be implemented	Takes the form year/quarter, e.g., 2014/3
Legal liability	Indicates the degree of legal liability of the requirement	"Optional", "Recommended", "Must"

3.4.4 Defining Dependencies between Attributes and Their Values

Attributes can be interdependent with regard to their values. When defining an attribute schema, you can define, for example, that certain combinations of two attributes with predefined attribute values are not allowed. For example, you can prevent a requirement with the value "Volatile" in the "Stability" attribute simultaneously receiving the value "Released" in the "Status" attribute. This ensures, for example, that only requirements that are considered stable are approved for development.

However, it may also make sense to combine these two attributes in one attribute and offer only the permitted combinations there. This is a solution particularly if the tool used does not support the consideration of dependencies between attribute values.

In variant management, assigning requirements to specific variants can be prohibited. It is also feasible to not allow certain combinations of variants.

You can also define that any transitions from one attribute value to another are not permitted. In particular, the transitions from one status to another should observe the defined lifecycle of the requirement (see Chapter 5).

Dependencies between attributes and their values can also arise through the hierarchization of requirements. For example, if requirement A is detailed by requirements A.1 and A.2, for the attributes of the attribute schema, you must define whether or not the value of A in an attribute depends on the values of A.1 and A.2.

For an attribute "Status", for example, it makes sense to define that the requirement can only receive the value "Released" if requirements A.1 and A.2 also have the value "Released" (and not if A.2, for instance, still has the value "In progress"). It would also not be explicable for requirement A to have a lower priority than A.1 or A.2.

Together with the stakeholders, Peter Reber decides that they do not want to define any limitations, except for those specified in the requirements landscape (see Chapter 2), and that not every transition between requirement statuses is permitted. The requirements information model specifies that business goals and business processes belong to detail level 1. Business goals are described in text form, with use case templates used for business processes. This results in limitations for the permissible combinations of attributes.

To ensure that the status transitions follow the predefined lifecycle of the requirements, and that it is not possible to accidentally skip steps (e.g., approvals), only the transitions shown in Figure 11 are permitted.

3.4.5 Providing Support for Recording Data

There is always one person responsible for each attribute, and specifically, for recording the attribute and regularly checking that the attribute is complete and plausible. By default, this is the requirements manager, but the manager can delegate responsibility for individual attributes to other persons involved in the project.

For the attribute owner, recording the attributes as well as the requirement is an additional effort. Clicking a value in a dropdown list is quick and easy but collecting the requirement information from the different stakeholders and clarifying contradictory statements is time-consuming and tedious. In many cases, therefore, technical support from a tool for recording and managing attributes is very important.

It is helpful, for example, to define standard ("default") values for attributes which are then set automatically when new requirements are created. The default value can be the most frequent value or the least meaningful value, or the value that all new requirements created have—for example, stability "Volatile". Note, however, that it may not be desirable for requirements classified as "Medium" not to differ from new requirements created that have not yet been classified. Default values are particularly useful for mandatory fields that must always be filled when a requirement is created.

You can also group two attributes into a useful "combined" attribute if only a few combination options are permitted for the respective attribute values. What is also helpful is common input functions, such as selecting or deselecting an entire values list with one function to select them for a requirement, for example (provided multiple values are permitted in the attribute), or vice versa: the selection of a list of requirements and setting an attribute in all of them.

Other help that can be provided by a tool is the automatic insertion of dependent attribute values of different attributes (where applicable, with a query). For example, a tool can automatically assign the status "In progress" to a requirement that already had the status "Released" but has subsequently been changed (where applicable, with a note). Help can also be realized, for example, when for hierarchically dependent requirements, values that are set in the parent node are automatically transferred to the "child nodes".

Some values should be set automatically in any situation as incontestable evidence—for example, the author (= user name of the author) and date of a change.

Default values for the attributes are now added to our attribute schema where this makes sense:

Attribute	**Values**	**Default Value**
Identifier	The bank's schema is: project code + sequential number, so in this case, OBA0001, OBA0002, etc.	Incremented automatically
Name	Free text	
Description	Free text	
Version	The format of the versioning is still to be defined	1
Author	Only the following persons are allowed to write requirements: Peter Reber, Martin Geldmann (business analyst), Anja Streng (IT security expert), Kirsten Uba (usability expert)	User name of the author
Source	List of all stakeholders	
Justification	Free text	
Stability	"Stable", "Volatile"	"Volatile"
Criticality	"Low", "Medium", "High"	
Priority for bank	"Low", "Medium", "High"	
Priority for customers	"Low", "Medium", "High"	
Owner	List of all stakeholders	Peter Reber
Requirement type	"Functional requirement", "Quality requirement", "Constraint"	"Functional requirement"
Solution dependency	"Goal", "Scenario", "Solution-oriented requirement"	
Attribute	**Values**	**Default Value**
Level of detail	Level 1: Business scope Level 2: User scope Level 3: IT scope	
Status	In progress, in evaluation, released, changed, rejected, deleted, implemented, tested, completed	"In progress"
Effort	Only whole or half values (in person days)	
Release	Takes the form year/quarter, e.g., 2014/3	
Legal liability	"Optional", "Recommended", "Must"	
Cross-references	"Being tested by", "Refined", "In conflict with", "Replaces"	
General information	Free text	

3.4.6 Documenting the Attribute Schema

Attribute schemas are presented in a tabular form or in an information model, depending on the degree of complexity (for example, with regard to the number of attributes, dependencies between attributes or their values) (similar to the presentation in Section 2.3). A requirements management tool then maps the corresponding attribute schema by providing the corresponding fields.

3.5 Change Management for Attribute Schemas

Retrospective changes to an attribute schema during the course of the project should be avoided if possible [RuSo2009]. The important things to note for a retrospective change to an attribute schema depend on the type of change.

3.5.1 Adding, Changing, or Deleting an Attribute

When a new attribute is added, the previously documented requirements should be updated to take account of the new attribute. This can be time-consuming. If you change the name, designations in different documents, views, and reports may no longer be consistent with one another. If an attribute is to be deleted, this often causes difficulties if a view, a report, or an interface to another system queries this attribute. Instead of deleting it, you can add "(no longer used)" to its name.

3.5.2 Adding, Changing, or Deleting Possible Attribute Values (Value Range)

Adding attribute values for an existing attribute is usually no problem for the underlying tool. From a technical point of view, you must check whether the requirements for which this attribute was already set have to be analyzed again and whether, if applicable, the new attribute value is better.

For example, if a new attribute value "Very high" were to be added for the criticality, all requirements with the previous criticality "High" must be checked again to establish whether the criticality is actually "Very high".

When deleting attribute values from a value range, it is important to ensure that requirements do not become inconsistent due to empty entries in the attribute. Problems are mainly caused by mandatory fields, since the requirement must have a valid value in the attribute under consideration. In this case, the solution may be to enter the default value in the field. For attributes with dependent attribute values, you must make sure that the removal of an attribute value does not result in impermissible attribute combinations.

When attribute values are changed, it is important to ensure that the changes are made in all requirements that contain the original value. In the particular case of requirements being exchanged between different systems, inconsistencies can occur if, for example, the change of a requirement value is not executed in a database (because it is not desired). In general, when adding, changing, or deleting requirement values, you have to decide whether this change will affect requirements that have already been entered or only applies to future requirements.

3.5.3 Adding or Deleting Relationships between Attributes

If you add a relationship or dependency between attributes, this can lead to some attribute combinations for the requirements already recorded suddenly no longer being allowed. For example, if you add the limitation that selecting a value for the attribute "Stability" must now always lead to the attribute "Risk" being populated (and this attribute is therefore always populated when "Stability" is populated), you must then check which requirements have a value for "Stability" but none for "Risk" and must therefore be updated.

Deleting a relationship between attributes is generally not critical. More is now permitted than before. You may be able to check whether, in some cases, the previously prohibited attribute combination would now make sense.

3.5.4 Changing Default Values for the Attribute Type

Changing default values should initially affect only the entry of new requirements. In this context, however, requirements that have been assigned the previous default value should be analyzed to check whether they still have the correct value, or whether they need to be adjusted.

3.5.5 Changing the Binding Character of Attributes ("Mandatory Fields" and "Optional Fields")

Changing a mandatory field to an optional field usually does not result in any subsequent effort. In contrast, if a change from an optional attribute to a mandatory attribute is planned, you must make sure that the attribute is populated with an appropriate value for all requirements that have already been documented. It may be necessary to assign custom values rather than use a default value.

Generally, when making changes to attribute schemas, you have to analyze the extent to which the views, reports, and interfaces to other tools are affected. For example, if scripts have been created in the tool that check or process a particular attribute, a change to this attribute can result in the corresponding scripts no longer being executable.

3.6 Goals and Types of Views

Projects often cover hundreds or even thousands of requirements. The people involved in the project no longer have an overview of this volume of requirements. Therefore, we need concepts to reduce this complexity. The view concept is very important here. A view is a reduced presentation of the requirements—for example, reduced to just some of the requirements or just some of the information (including the attributes). The basis for this is the filtering and sorting of requirements.

Definition 3-4: A view is a goal-oriented abstraction of the requirements that covers only those requirements and associated information that are relevant for the respective purpose (e.g., stakeholders, decision requirement). From a technical perspective, however, a view is a predefined reusable combination of filter and sorting settings as well as abstractions and aggregations.

The IREB Foundation Level differentiates between two basic types of views [IREB2015]:

- *Selective views:* presentation of a subset of the requirements that have a specific attribute value—for example, all requirements with the status "In progress".
- *Condensed views:* presentation of summarized information for the selected requirements—that is, information that is not present in the original requirements list but is calculated, such as the number of all requirements with criticality "High" and status "In progress".

There are also *projective views*: in projective views, those attributes that are not relevant for the view under consideration are hidden so that only the information about a requirement that is relevant for a view is displayed.

In a requirements engineering tool, you can define and then save a view. Due to regular use of exactly the same view, over the course of the project, reports are created that are comparable with one another and that document the course of the project.

The attribute schema is the basis for the definition of a view, as every view is based on the information defined in an attribute schema. Therefore, when you create an attribute schema, you must also consider the views that are to be created later.

Stakeholders of a project often need specific information to perform their tasks. Different roles need different information and therefore different views of the requirements:

- The s*takeholders*, in particular those who define the requirements, want to know where a requirement comes from (the source, e.g., stakeholder or document) and what goal the requirement supports. They want this information so that they can ensure that all requirements are actually necessary (= backward traceability or pre-requirements specification traceability). The stakeholders are also interested in forwards traceability (= post-requirements specification traceability)—that is, the traceability of requirements to subsequent development artifacts such as the system architecture, implementation, and test [IREB2015]. For example, what test can the stakeholder use at acceptance to check whether a specific requirement has been implemented correctly?
- *Requirements engineers and requirements managers* want to make sure of the quality of the requirements, particularly the consistency of the requirements between documents and detail levels. Traceability is also important for this, as is an overview of the status of the traceability.
- *Project managers* are interested in the status of the project (or a release), so that they can forecast the residual effort and residual duration for requirements engineering or the project. They could perform requirements-based project management—for example, use the status of the requirements to evaluate the degree of completion of the project. This is common in agile development but is generally always possible if the quality of the content of the attributes is correct—that is, the probable implementation effort for each requirement is defined here, the processing status is clear from the status, and corresponding evaluations are available in the tool. When requests are made for changes to requirements, the project manager wants to be able to predict the probable effects, such as the effort, side effects, and new risks. Traceability is very important for such evaluations. The project manager also wants to know who the contact person is for a specific requirement. That is a reason for the attributes "Source", "Owner", and/or "Next processor".

- The main thing that the *architect* needs is a structured view of the requirements—for example, the grouping of the requirements according to technical criteria for the assignment of components. For this purpose, attribute types such as "Interface requirement" are useful.
- *Developers* want access to the original requirements that belong to the next component they are implementing. This is supported by traceability between requirements and components.
- *Testers and test management* want to know which and how many requirements still need to be tested, and how many they have already confirmed as being free of errors. This allows them to measure the progress of testing and to estimate the residual effort for testing. It is also important to know which test cases have to be executed again in the event of a change.

The above-mentioned views can be created relatively easily by filtering and sorting the requirements according to their attribute values and through evaluations such as the creation of totals across attributes. However, the absolutely essential prerequisite for this is that the corresponding attribute values are present and well maintained, and that the requirements engineering tool allows the required evaluations.

In addition to providing a clear focus and presentation of the information available for the different roles, views can also regulate access to requirements on a role-specific basis. It is often the case that you do not want every person involved in the project to have access to all information, and you therefore need to be able to generate role-based views (see [Pohl2010]). This type of authorization concept can be realized so specifically in a tool that specific roles can only see specific views.

3.7 Defining Views and the Risks of Views

The process for defining views includes the following steps:

- *Stakeholder identification:* definition of the stakeholders who need one or more views.
- *Reuse*: views from other projects or from a reference project can also be used as a template for the views to be defined.
- *Specification of goals*: for each stakeholder, you need to know the goal of their views. This determines which information should be filtered out or summarized, or which sorting should be set initially. In this context, you must also define the rights of roles and views, that is, which role should be able to activate which view. It is efficient if one view can be used by multiple roles.
- *Specification of required attributes and comparison with the attribute schema*: to be able to fulfill the goals of a view, you must ensure that it is possible to collect the necessary information and that the corresponding attributes are also available. An evaluation of the number of requirements that are still in a volatile status can only be generated if this status is also documented in a corresponding attribute. The comparison with the attribute schema often leads to the discovery of new views because when they look at the attribute schema, the stakeholders realize which evaluations would still be possible. The definition of views and attribute schemas therefore influence each other.
- *Implementation of the view*: finally, the predefined views must be implemented and tested in the underlying tool.

Sometimes, users of a requirements management system are not aware that the complex information about a requirement can be restricted by views as required. They then often work with a global and all-encompassing view and perceive the tool unjustifiably as too extensive and possibly obstructive in their work. At the same time, views also bear risks. For example, in a view, too much context could be lost: if you create a view in which atomic requirements are given in a list without any context (e.g., use cases), for example, this overview will only be meaningful to a limited extent. To avoid such ineffective views as far as possible, when you define a view you must always take the underlying goals of the stakeholders into account.

3.8 Implementing a View

Most views are implemented in a requirements management tool by selecting/filtering and sorting by attribute values (see also [Pohl2010]).

When selecting requirements, you can use a predefined filter to hide requirements—or attributes of the requirements or both—that you are not interested in for the current situation. This means that the view presents only an extract of the available data. For example, by using a filter on a specific release, a tester can select the requirements that are relevant for the current release being tested. The tester can identify these requirements from the fact that they have the value that the tester is searching for in the "Release" attribute. At the same time, the information about the predicted effort for the realization of a requirement is probably not relevant for a tester, so the corresponding attribute can be hidden for the specific view for the tester. The tester can also sort by criticality so that they can begin testing with the critical requirements. By sorting, a stakeholder can change the focus on requirements (for example, all requirements with a high criticality at the beginning of a list) without hiding requirements.

Condensed views also show information that is not contained in the requirements in this form. This information arises, for example, from the creation of totals and subtotals or through the calculation of percentage ratios. This allows you to determine, for example, the proportion of requirements with the status "Tested"—where applicable, also weighted by effort. You can then determine how the degree of completion of the requirements continues to develop over a specific period of time.

Condensed views can be combined with selective views—for example, to determine the degree of completion of the requirements in relation to a specific release. The summarization is calculated on a filtered set of requirements.

The following are examples of views you can create through filtering:

- All released requirements
- All requirements that belong to a specific release
- All requirements that have already been tested
- All requirements for which a specific person is responsible
- All requirements that a developer has to take into account when implementing a specific component

The following are examples of views you can create through sorting:

- A presentation of the requirements in the order of their criticality
- Sorting the requirements according to the person responsible shows the distribution of work across the team members

3.9 Optimizing the Assignment of Attributes and Creation of Views

Practice shows that in some projects, the attributes are not populated to the expected extent, sometimes even for a good reason. Therefore, it makes sense to check regularly and at specific points in a project whether, and how, an attribute should still be used [RuSo2009].

In principle, only those attributes that provide a benefit in a view or a report for at least one stakeholder should be retained. However, the required attributes should be maintained as well as possible.

One attribute deficit that can be checked easily is where attributes have not been populated. To find unpopulated attributes, you can either define a view for this purpose or sort by the corresponding attribute. The requirements with the empty field are then either at the very top or the very bottom of the list.

If you want a specific attribute to always be populated, the easiest way to ensure this is to define the attribute as a mandatory field. This forces input when a requirement is created. However, this is rarely the optimal solution. Note that defining too many mandatory fields can greatly impede processing and that, when a requirement is initially created, some information may not be available yet, such as the cost estimation. For this reason, mandatory fields should be declared only sparingly and with a sense of proportion.

For optional attributes whose entry is not mandatory, the following conclusions can be drawn from evaluations of their previous use:

- The attribute was not used in either a view or a report: this indicates that the attribute in question does not support a specific goal and is probably not of interest to any of the stakeholders. This raises the question of the point of this attribute, as every attribute present causes a maintenance effort.
- The attribute is always populated with the same value, for example the default value: in this case, there does not seem to be any real distinction between the different requirements in relation to this attribute, which means that the proposed list of values for selection is not suitable. You can either discontinue the attribute (because nobody uses it) or adjust the selection list. In the latter case, the notes from Section 3.5 about changing attribute schemas should be taken into account.
- The attribute is never populated: if the attribute has deliberately not been populated, the information may not be important. If this assumption is confirmed, the attribute should be removed. However, the reason for attributes not being populated often lies in the fact that users are not aware of the definition or the benefit of the attribute, or do not directly see any benefit since the information is used by another stakeholder. Then the users should be (re)trained to explain to them the added value of the particular attribute. In this case, the attribute can continue to be used.

- The attribute is only filled for a few requirements: the question here is whether the goal associated with the attribute can be achieved, or whether it is still relevant at all. If this is not the case, the attribute can be removed. However, if it turns out that the attribute is important, it can be declared as a mandatory field, which forces entry in the future.

  In this case, requirements which have already been documented but have no value in the attribute under consideration must be updated retrospectively (for example, automatic population with a default value).
- The attribute is not populated in individual cases: it must first be determined whether this attribute is still relevant for the project. If yes, the requirements engineer should complete the relevant requirements. If the attribute is no longer considered essential, it can either be removed or the gaps can be tolerated.
- The attribute is always populated: in this case, no further activity is necessary.

In addition to checking that attributes have been populated, you should not forget to ask stakeholders if they are satisfied. They may be missing some information in their views which is also missing in the attribute schema. The missing attribute or missing attribute value should then be added and if necessary, populated retrospectively for the requirements that have already been recorded. It may also be the case that an attribute or value is obsolete. If a stakeholder no longer needs it in their view, the question should be asked as to whether other views use the attribute or value, or whether it can be deleted. Care must be taken when making changes to an attribute schema (see Chapter 3.5).

3.10 Content for the Requirements Management Plan

The requirements management plan documents the attribute schema. The schema describes the requirements attributes to be used. For each attribute, the name, a description, the person responsible, permitted values, and dependencies to other attributes are documented. You create the attribute schema in tabular form, for example, (see Table 2), or in an information model.

In the requirements management plan, you also define the views to be supported. For each view, the goal and the stakeholders that use the view are documented, as well as the attributes to be displayed, the filters to be applied, and predefined sorting. You must ensure that the attribute schema contains all of the required attributes.

3.11 Literature for Further Reading

[Pohl2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, 2010.

[RuSo2009] C. Rupp & die SOPHISTen: Requirements-Engineering und –Management. Hanser, 5th edition, updated and extended, 2009 (available in German only).

4 Evaluating and Prioritizing Requirements

4.1 Motivation and Difficulties When Prioritizing Requirements

Not all requirements are equally important. This becomes obvious at the latest when only some and not all of the requirements can be implemented within a fixed budget or time period. You then have to decide whether to increase the budget, deliver later, or reduce the scope of delivery. And suddenly, some requirements—functions or quality requirements—are no longer indispensable. In some cases, two requirements cannot both be implemented, or at least cannot both be implemented to the same quality for technical reasons (= requirements conflict). To resolve this conflict, you have to decide which of the two requirements is more important. In a lot of cases, the following applies: "Schedule should drive requirements" [Davi2005]. This means that when time is short, some requirements often have to give way to other requirements.

Definition 4-1: The priority (or importance) of a requirement documents the importance of a requirement compared to other requirements with reference to a defined criterion (IREB Glossary [Glin2014]).

"Compared to" means that this priority does not necessarily require absolute values—for example, the importance measured in euro, or implementation effort measured in person days. As the value is being used to compare the requirements, it can be a relative value, on a scale of 1 to 10, for example. You just want to know which requirement is more important than another requirement.

The priority of a requirement is the basis for some decisions in the software or system development process. In addition to resolving conflicts between requirements and release planning, such decisions include technical decisions as well as prioritization for testing. The more important or more critical the requirement that is being tested by a test case is, the more important the test case also is and therefore also the errors found, and thus the more thoroughly this requirement should be tested.

Definition 4-2: Prioritization refers to the activity of determining the priorities of requirements.

The prioritization prepares the negotiation and selection of requirements as well as the release planning.

Prioritization is made more difficult by the fact that the importance of a requirement ultimately depends on many factors, in particular:

- The criteria you create
- The perspective you take (i.e., the importance of a requirement differs for different stakeholders)
- The decision to be supported by a requirement (i.e., does a low prioritization mean that this requirement will never be delivered, or will it simply be delivered four weeks later than the other requirements?),

- The point in time at which the importance is evaluated

For example, is it just financial criteria such as costs and benefits that are important, or does customer satisfaction also play a part, or is the aim to minimize risks? In a system in which security is critical, for example, reducing risks will be seen as more important than user-friendliness; in other systems, the reverse is true.

4.2 Principles of Evaluation

The evaluation of the requirements is the basis for prioritizing the requirements. The priority (that is, the importance) of a requirement is often determined from multiple evaluation criteria, for example, by comparing the costs and benefits of this requirement.

The following are examples of evaluation criteria:

- Implementation effort or other costs
- Importance or benefit for the user or other stakeholders
- Probable frequency of use of a function
- The legally-binding nature of a requirement,
- Dependencies between requirements (underlying requirements must be implemented before requirements that are dependent on them)
- Criticality (also referred to as risk)
- Stability or the degree of innovation (see the Kano prioritization in Section 4.5.8)

The priorities of the requirements naturally correlate with the priorities of the associated system functions, test cases, and errors discovered during the test. In other words: if an important requirement or functionality has errors, these errors are more severe than similar errors in functionality with low importance that is used only rarely.

Stakeholders with the appropriate qualifications are responsible for evaluating requirements. For example, the end users or product management are the best persons to evaluate the benefits of a requirement. The best contact persons for costs are the technical personnel. However, even the question of which evaluation and prioritization criteria are to be used must be agreed with stakeholders. It is primarily the users of the prioritization results that do this (e.g., the project manager), that is, the persons who have to take decisions based on the priorities. The prioritization criteria must be defined precisely, including the scale to be used and the evaluation method. For example, should the costs be determined by means of an evaluation by experts, or by means of a counting method such as the function point method?

Are function points sufficient as a relative dimension or does the effort have to be converted into person days? And who is permitted to or should perform this evaluation? When (at the earliest or latest) should the evaluation be conducted?

Sources for evaluation criteria include:

- Project management
- Guidelines and standards
- The requirements attribute schema
- The disciplines that follow development, such as quality assurance

Requirements management is responsible for ensuring that the priorities are determined and suitably documented. The attributes are ideal for this documentation. The prioritization criteria must therefore be part of the attribute schema (see Chapter 3).

The attribute schema in our case study already contains some attributes that are suitable for prioritizing requirements:

- Stability: volatile requirements should not be implemented yet; instead, they should be shifted to a later point in the release plan until they have become stable.
- Criticality: particularly critical (i.e., risky) requirements are treated differently to non-critical requirements. For example, the IT security experts should perform a systematic risk analysis for the risky requirements. The critical requirements should also be tested particularly thoroughly.
- Priority for the bank: if the goal is to quickly create as much return on investment as possible for the bank, the requirements evaluated as "High" here should be the first to go live.
- Priority for customers: if the goal is to quickly provide as many user benefits as possible, the requirements evaluated as "High" here should be the first to go live.
- Effort: the effort can influence the prioritization in two different ways. The fixed, predefined budget for a release specifies a cost limit. Therefore, in the release planning, the budget determines how many of the most important requirements may be selected (i.e., the budget must not be exceeded). Furthermore, the benefits and effort for a requirement can be used to calculate the cost/benefit ratio of this requirement so that this can be used as a prioritization criterion. When evaluating the importance, it is then not the absolute benefit that is the deciding factor, but whether the efforts incurred are worth it: do the benefits exceed the costs, or the costs exceed the benefits?
- Release: the release number of a requirement is already the result of a prioritization, probably due to other prioritization criteria.
- Legal liability: all "must" requirements must be included in release 1.

4.3 Prioritizing Requirements

As we saw in the previous section, there is a lot to consider when prioritizing requirements. The ideal solution is to proceed systematically, in the following order:

- *Define the goals of the prioritization:* Who will need priorities to make decisions and when will they need them? Which decisions have to be made? Why and for what purpose is this decision important? Which (higher level) goals should this decision support?
- *Define the prioritization criteria:* Which criteria should be used for the prioritization to achieve these goals? If, for example, the goal is to maximize the cost/benefit ratio of the entire project, it makes sense to also determine the cost/benefit ratio for each requirement and use this criterion in the release planning. The scale and the value range must also be defined for each criterion. Should absolute or relative values be determined? This depends on the type of decision to be made and the level of detail required for the evaluation. If, for example, you want to determine the most important third of requirements from a list, an evaluation on a scale with 1, 2, or 3 points is sufficient. However, if you want to create a list sorted by importance, an ordinal scale is the best solution. An ordinal scale assigns values to the requirements, from "Most important requirement" through "Second-most important requirement", right down to the least important requirement. This is represented via a sequential whole number which can start at 1 for the most important requirement but can also assign 1 to the least important requirement.
- *Define the prioritizing stakeholders:* The requirements are initially evaluated based on the evaluation criteria. In each case, different stakeholders may have the necessary expertise to reliably evaluate the different respective evaluation criteria. Based on these evaluations, a person or group of persons then prioritizes the requirements. All of these stakeholders must be selected according to their competence.
- *Define the requirements artifacts to be prioritized:* If the requirements are described at different levels of detail, you will probably want to prioritize at just one of these levels. You therefore select the level at which the decision has to be made. If, for example, you want to select the business processes to be implemented in the first release, you prioritize only the business processes, and not the refining usage scenarios. It also does not make sense to compare apples with pears—for example, to compare features with mock-ups. When selecting requirements to be prioritized, note that the requirements should be at a similar level of detail to avoid distorting the result of the prioritization [WiBe2013]. Less refined, more abstract requirements tend to have higher priorities than more detailed requirements, as a less refined requirement covers multiple detailed requirements. The primary aim here is to limit the number of requirements to be prioritized because otherwise the effort involved in prioritization can be very high. (We will come back to this point later on.)
- *Select the prioritization technique:* This point refers to both measurement procedures and evaluation methods for determining criteria, as well as a sorting method for the requirements. Various prioritization techniques are described in Sections 4.4 and 4.6. These are mainly sorting and measurement procedures. You also have to define who performs this prioritization and when they do so. It will probably be the case that different experts will evaluate different criteria based on the required competence.

- *Where necessary, adapt the attribute schema:* The priorities of the requirements are generally documented in attributes. The prerequisite for this, of course, is that the attribute schema contains the corresponding attributes with the correct value lists. If this is not the case, you have to adapt the attribute schema. We discuss the points to note for this critical activity, particularly if the attribute schema is already in use, in Chapter 3.
- *Prioritization:* The prioritization is now performed as planned. All evaluations and priorities are documented, including the justifications and any assumptions made.
- *Check the requirements regularly and reprioritize where necessary*: Things change over time—some things become more important, some less important. Sometimes our knowledge of the facts also improves. Priorities thus also change over time and requirements must therefore be checked regularly. The best time to do this is always just before an important decision has to be made based on the priorities.

Stakeholders of the prioritization, goals, and criteria:

In our case study, the changes to the online banking system are to be delivered in individual releases. The project manager performs the release planning, supported by our requirements manager, Peter Reber. The following rules are defined for the release planning:

- Stability: volatile requirements are never included in the next release.
- Legal liability: all "must" requirements must be included in release 1.
- Of the remaining requirements, the most important requirement for the bank (attribute "Priority for the bank") and then the most important requirement for the customers (attribute "Priority for the customers") is selected alternately until the release budget has been consumed.

Only business processes are to be prioritized, as every release should contain only complete business processes. Half-implemented business processes are generally of no use to anybody.

To allow quick decisions in cases of crisis or conflict, the project manager wants to always be able to see a sorted list of requirements, sorted either by their importance for the bank or their importance for customers. In addition to the business processes, the project manager would also like to see the user and system use cases prioritized, as well as all solution-based requirements. However, it soon became clear that this detailed prioritization would involve an enormous amount of effort.

Prioritization workshops lasting hours or even days would be necessary. Therefore, there is initially no prioritization of the solution-based requirements (and therefore no workshop).

The IT security experts use the attribute "Criticality" to note requirements for the next steps: requirements evaluated as "High" are to be subjected to a thorough risk analysis by the group, including an error tree analysis; the "Medium" criticality level requirements are to be subjected to a normal risk analysis by one person; and the requirements evaluated as "Low" criticality will not be considered further.

The goal of this procedure is to draw attention to the particularly relevant areas.

The usability expert will deal intensively with the requirements that are particularly important for the customers.

The attribute schema therefore contains all of the attributes required for decision making (see Chapter 3). The potential introduction of a further attribute that highlights all requirements that are particularly important for accessibility was briefly considered. However, an initial analysis established that this would apply for almost all requirements. The attribute would therefore not be useful for the intended purpose as a differentiation and prioritization criterion. Therefore, this plan was abandoned.

We will select the appropriate prioritization techniques later in the chapter, once you have learned about the different techniques.

4.4 Two Types of Prioritization Techniques

There are a lot of prioritization techniques. They differ in the level of effort required, the level of subjectivity and the rate of errors, and in their suitability for different purposes. We cannot provide a complete overview of all the techniques that exist here. Therefore, we present the techniques that are the most important, most widespread, and most suitable for practice.

We assume a situation in which the goals and criteria for the prioritization have already been defined, as well as the persons who can evaluate (or often, even predict) which requirement will fulfill which criteria and how well. The artifacts to be prioritized have also been defined. If you have a complex requirements landscape and, for example, specify the requirements at different levels of detail, you should compare only those requirements that are at the same level of abstraction. Anything else would be a case of comparing apples with pears. To get reliable results, depending on the decision to be made, you should prioritize at only one level of detail: the level at which this decision is to be made. For example, if the goal is to select the most important business processes for the first release to be delivered, then prioritize the business processes. Alternatively, if you want to define the order of implementation of the use cases, then prioritize the use cases. The starting point for the prioritization is therefore an unsorted list of requirements at the same level of abstraction.

The prioritization technique converts this list into a sorted list in which a priority value is assigned to every requirement. Further activities are then possible based on this list, such as the release planning.

We differentiate between two types of prioritization techniques:

- With an ad-hoc prioritization technique, an expert assigns a value to every requirement (based on experience). This can be done quickly but is more prone to error.
- The analytical prioritization technique is a more systematic process, comparing, for example, pairs of requirements with one another; or different experts evaluate different criteria, which then results in the priority. This technique involves more effort, but the result is more reliable and has been created more carefully.

4.5 Ad-Hoc Prioritization Techniques

The following ad-hoc prioritization techniques have proven to work well in practice:

- Requirements triage
- Ranking
- Top-Ten Technique
- Single-criteria classification
- Planning Poker
- Two-criteria classification
- 100-dollar technique
- Kano classification

4.5.1 Requirements triage

Requirements triage [Davi2003] is a single-criteria classification based on medical science. It can also be used to presort requirements and to simplify the prioritization. Each requirement is assigned to one of three categories:

- Requirements which "must" be implemented (e.g., in the next release)
- Requirements that are not necessary (yet)
- Optional requirements, for which the priority is not yet clear; these requirements need to be prioritized more precisely, or their implementation depends on the available resources

The optional requirements can be evaluated with a different prioritization technique. For the other two requirements groups, the decision has already been taken. If there are too many "must" requirements, or you want to put them in order, a more detailed prioritization makes sense here.

Triage can also be used for other decisions as well as deciding which requirements to implement.

4.5.2 Ranking

In the ranking technique, the stakeholders sort the requirements into an order based on the prioritization criterion (e.g., benefits, costs, urgency). Any prioritization criterion is possible here. As a result, the requirements are assigned to the ordinal scale we have already mentioned (see Section 0), in which, for example, the most important requirement receives number 1, the second most important number 2, and so on. If the number of requirements is low, they can be sorted ad-hoc at a glance. This can be done, for example, by writing the requirements on index cards and then sorting them on a table, in a group if desired.

If there are more requirements than you can compare at a glance (e.g., more than 10), either presorting (triage) or a systematic sorting procedure helps. After presorting by means of requirements triage, you then use the prioritization technique within only one category of requirements—for example, for the optional requirements.

The following is recommended as a systematic sorting procedure: you place a requirement on the table (or display it electronically). You then take the next requirement and decide whether it is more or less important than the first requirement. You then place this requirement either further up or further down in a list accordingly. As far as possible, no two requirements should have the same rank, although an equal evaluation would be possible in an exceptional case. You then proceed in the same way with every additional requirement. It is usually quickly clear whether the new requirement is relatively important or unimportant and should therefore be compared with the requirements at the top or bottom of the list. In the end, you have a complete list of the requirements sorted by priority.

4.5.3 Top-Ten Technique

You often do not need a completely sorted or prioritized list of requirements. It is often the case that you are simply looking for that group of requirements that currently has the highest priority and should be processed in the next step—for example, implementation or testing. The approximate number of requirements you are looking for is also often clear. If, for example, a requirement causes an average implementation effort of four days, and in the next iteration, resources are available for 40 person days, the objective of the prioritization is to determine the ten most important requirements: the top ten. The remaining requirements do not need to be prioritized other than determining that they do not (yet) belong to the top ten. Of course, the method also works for the most important three or twenty-four requirements, for example.

The procedure is as follows: in the first round, you collect all candidates for the top ten. You will probably not get exactly ten candidates. If there are too many candidates, select those that do not initially belong to the top ten. If there are not enough candidates, look at the rejected requirements again. Removing requirements from the list of candidates is usually easier than looking at all requirements again.

Therefore, in the first round, if there is any doubt about a requirement, you should include it as a candidate rather than reject it. To remove requirements from the top ten list, you can use one of the other prioritization techniques (e.g., ranking or the techniques described below) and then cut the sorted list of requirements off after number 10.

If multiple stakeholders or stakeholder groups (e.g., five) are involved in the prioritization, you could, for example, arrange the process such that the five groups are each allowed to select their top two requirements, resulting in the top ten list.

4.5.4 Single-criteria classification

With single-criteria classification, you evaluate each requirement in order according to the prioritization criterion.

You can use any useful scale—for example, absolute values such as the implementation effort in person days, or relative values on a points scale that you define yourself (e.g., 0 to 10 points), or categories such as low/medium/high or mandatory/optional/nice to have. You can do this relatively quickly. However, you must define clearly beforehand what each number of points means. For example, you can define that criticality 10 can only be assigned if there is a risk to human life or the existence of the entire company, or that each requirement for which the work is possible by means of a workaround if the requirement is not implemented is an optional requirement. The greater the number of people involved in the prioritization—regardless of whether that is in a group or if an average value is to be calculated from the individual evaluations of the evaluators—the more important such definitions are. If this is not done as described, where prioritization is performed in groups, time will be wasted in a lot of cases—for example, due to the discussion of the meaning of the points value 10, rather than a discussion about the priorities themselves.

If you compare and aggregate independent evaluations by different experts, in principle you receive more reliable values than in a group discussion, as groups cannot always decide optimally because of the effects of group dynamics. However, you also see then that different evaluators have different levels of optimism. One evaluator assigns the value 1 more often, for example, and generally lower values, whereas another assigns the value 10 more often and this evaluator's values are higher on average overall. However, this difference is often due less to differences in character, and more to a different understanding of when the extreme values 1 and 10 are to be assigned.

4.5.5 Planning Poker

In planning poker, the requirements are also evaluated with reference to a criterion, usually with reference to costs or the benefits of the requirements with regard to the scheduling of the requirements in specific releases. This technique is widespread in agile development but also works in other areas as well.

It takes into account the pitfalls of decision processes in groups and therefore potentially leads to better evaluations than a group discussion. We have already mentioned the effects of group dynamics. When experts evaluate separately, they can, however, develop a different understanding or overlook a relevant factor.

Therefore, planning poker uses a pragmatic compromise between individual evaluation and group discussion as its decision process. Furthermore, the evaluation is based not on an even scale of points from 1 to 10, but on Bernoulli numbers which have been proven to work for this purpose. Each of the evaluators sitting at the table together receives a set of playing cards with the following points values: 0, 1, 2, 3, 5, 8, 13, 21, 34. There are also cards that the evaluators can use to register a need for discussion or a break.

To prioritize the requirements together in the group, the process is then as follows:

1. Presentation of the requirement to be prioritized (2 minutes)
2. Each evaluator makes their own evaluation (½ minute): each evaluator selects one of their cards and places it face down on the table to signify that they have made their decision. The other evaluators thus see only the back of the card.
3. The cards are revealed simultaneously: as soon as everyone has selected a card, all evaluators turn over their cards.

4. Explanation of the highest and lowest evaluations (1 minute): the two people who have given the lowest and the highest evaluation each explain their value. Their evaluations are usually based on different assumptions than those made by others, either justifiably or unjustifiably. These assumptions are now discussed.
5. Everyone makes their own evaluation (½ minute): based on the new knowledge provided by colleagues, the evaluations are repeated: everyone selects a card and places it face down in front of them.
6. The cards are revealed simultaneously
7. Agreement on an evaluation (1 minute): in the second round of evaluations, the values are closer but not necessarily identical. The group can now decide to take either the most frequently occurring value or the average of the evaluations.

This technique can be used for all prioritization criteria imaginable—costs, benefits, or other criteria.

4.5.6 Two-criteria classification

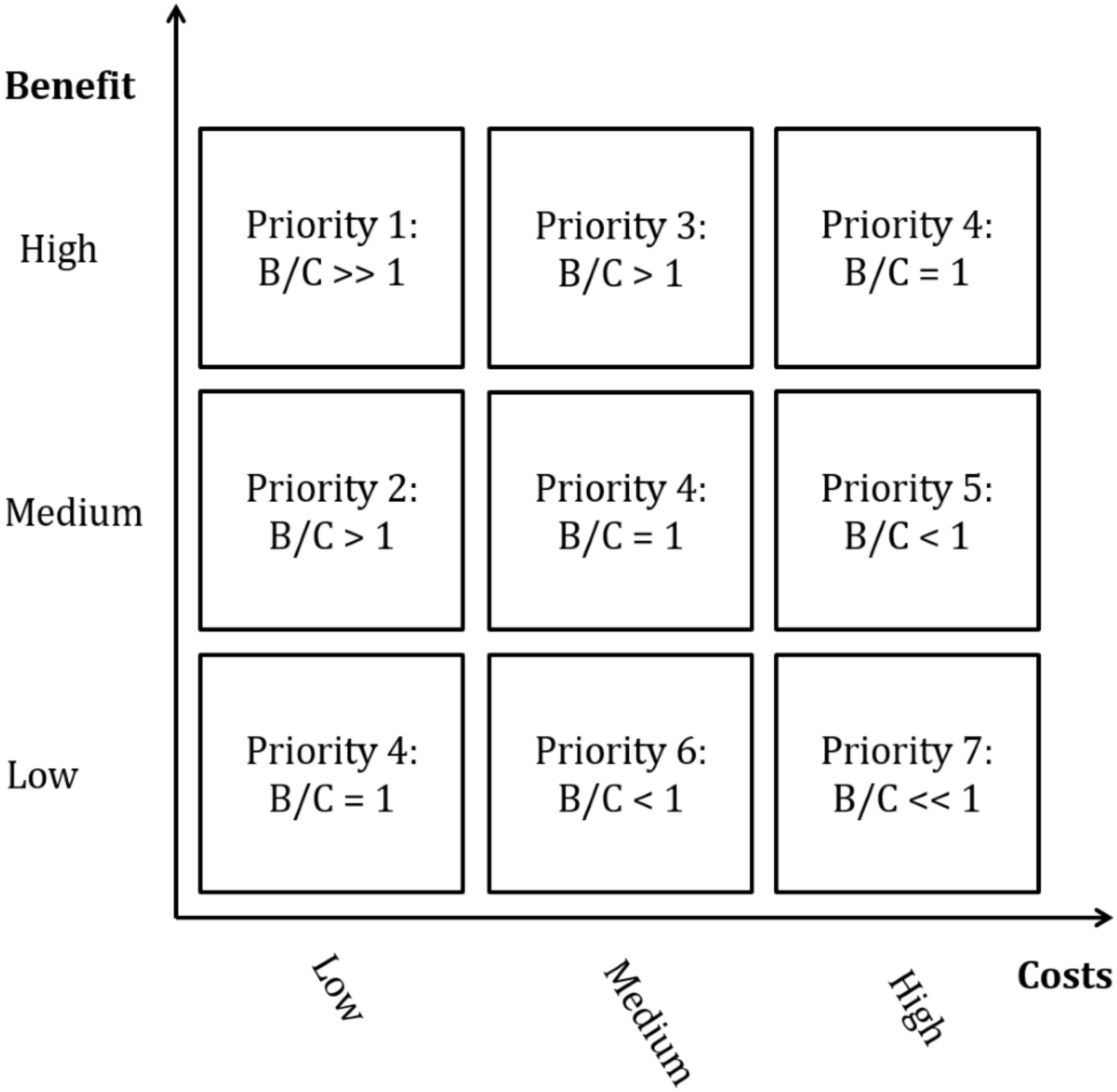
In some cases, multiple prioritization criteria are to be considered simultaneously. There are various options for combining two criteria:

- *With a formula:* If, for example, you are interested in the cost/benefit ratio, for each requirement, you determine firstly the costs and then secondly the benefits. This process probably involves questioning various stakeholders—for example, technical experts for the costs and the users for the evaluation of the benefits. The cost/benefit ratio is then calculated for each requirement from the quotient between the two values. The requirement with the highest benefits per cost unit invested then has the highest priority. In this process, the costs and benefits do not necessarily have to be determined in the same unit (e.g., euro). A ratio in the unit "points/person day" is also useful.

  For the purposes of documentation and use in views, it makes sense to define not only the cost and benefit evaluations in a separate attribute in the requirements engineering tool, but also their quotient.
- *With a matrix:* You can determine the criticality of a requirement as its calculated risk, for example, which is defined as the probability of occurrence of a risk event multiplied by the damage incurred if the event occurs. With online banking, these risks can be very high. However, this number does not ultimately contain all the relevant information. Extremely rare catastrophes with an almost inestimably high level of damage amount perhaps to €10 per month, in the same way that small accidents that occur regularly and cause damage of €0.01 each time also amount to €10 per month. In many cases, these categories will be handled separately and differently. Instead of multiplying the probability by the damage, the preference is to create a risk matrix to present the risks in a two-dimensional diagram for the purpose of classification.

Figure 4 shows an example of a matrix in which requirements are prioritized according to their cost/benefit ratio. At the very top left (Priority 1) we can see the quick wins—that is, requirements that bring a high benefit at low costs. These thus receive priority 1, the highest priority. In the fields with priorities 2 and 3, the benefits are also still higher than the costs. The requirements on the diagonal, where the costs and benefits more or less balance out, are classified in the same category (Priority 4), and so on.

How you assign the requirements in priority categories based on the two criteria is your decision and naturally influences the next steps, for example, the release planning.



Benefit \ Costs	Low	Medium	High
High	Priority 1: B/C >> 1	Priority 3: B/C > 1	Priority 4: B/C = 1
Medium	Priority 2: B/C > 1	Priority 4: B/C = 1	Priority 5: B/C < 1
Low	Priority 4: B/C = 1	Priority 6: B/C < 1	Priority 7: B/C << 1

Figure 4: Prioritization matrix for requirements according to costs and benefits. B/C designates the ratio (i.e., the quotients) of benefits and costs.

Figure 5 shows an example of a prioritization matrix according to risk (= risk matrix). Here, the risks linked to a requirement are prioritized and thus also the criticality of the requirement which is threatened by the risk. For example, the requirement (and the functionality) for transfers in online banking bears the risk that hackers will be able to get hold of the account holder's access data and execute an unauthorized transfer. This risk is possible and serious and is therefore in a red box in the matrix. This means that countermeasures must be taken at all costs.

For the risks in the yellow area, you weigh up which measures make sense economically. The risks in the green area may be accepted, unless they can be prevented with simple countermeasures. The three areas of the matrix therefore determine how the respective risks are to be handled.

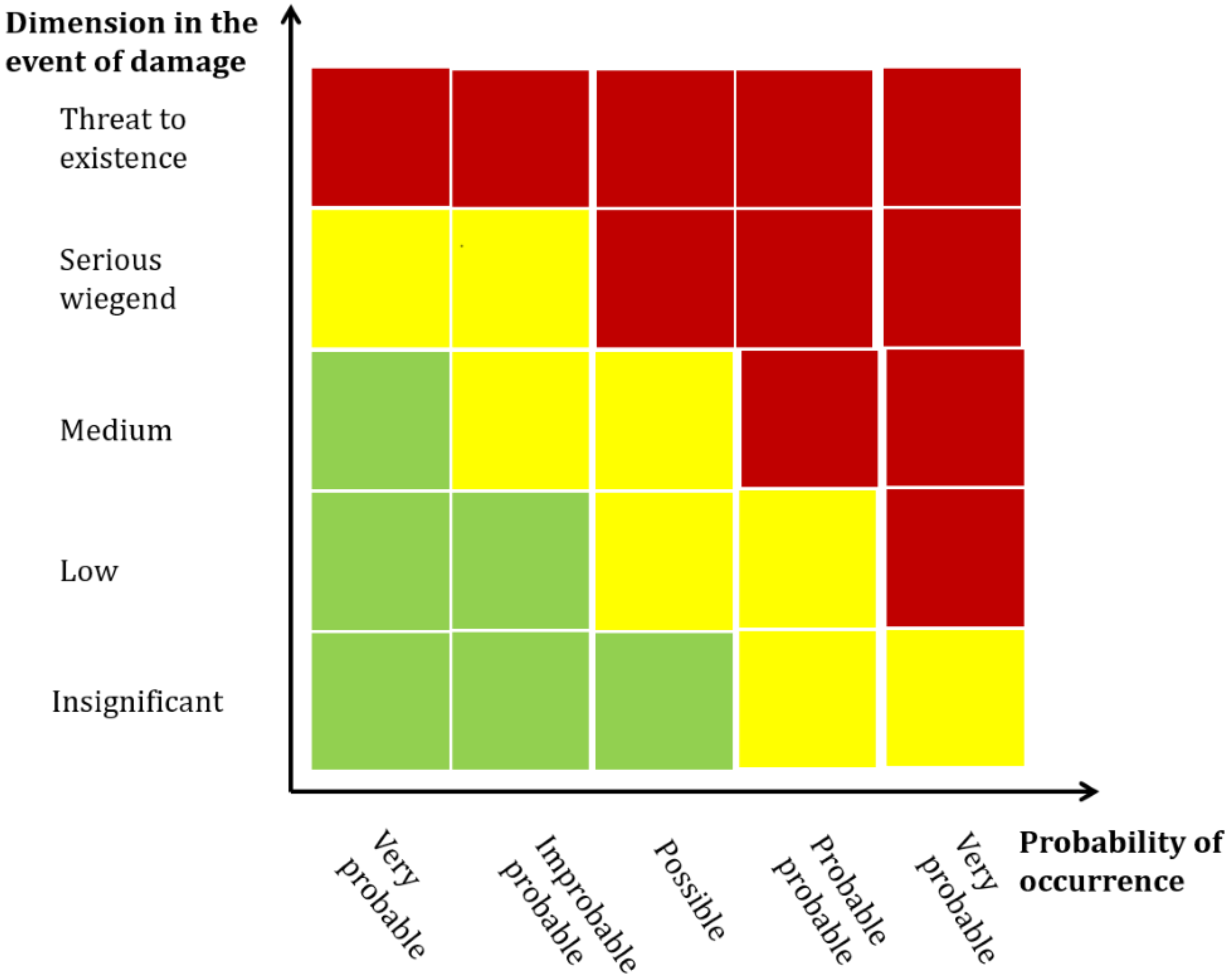


Figure 5: Risk matrix for the prioritization of requirements with reference to risk and criticality

4.5.7 The 100-Dollar Technique

The 100-dollar technique [LeWi2000] is particularly suitable for prioritization with multiple persons who do not necessarily have to meet up for a group discussion.

With this technique, stakeholders are granted 100 imaginary units (money, time, etc.) which they can assign to the requirements. Any requirement that is worth, for example, double the amount of money to a stakeholder than another requirement should also be assigned double the number of units/points. Each stakeholder can assign a maximum of 100 units. At the end, the points that the different stakeholders have assigned to the same requirements are added up. The requirement with the most points has the highest priority.

This technique is difficult to implement for larger numbers of requirements. In that situation, the stakeholders find it difficult to weigh up all of the requirements against one another, and it is also more difficult to ensure that a maximum of 100 points is assigned. Therefore, we recommend this technique for prioritizing rough requirements (e.g., features) with a high level of abstraction, or for mutual prioritization of entire requirement groups. Within the requirement group, the 100-dollar technique can then be used to compare the requirements with one another.

Two situations must be avoided with this technique: if the stakeholders want to make their lives easier, they assign the same number of points to every requirement. For example, in the case of 100 requirements, every requirement receives exactly one point. At the end of the process, all requirements will be equally important and there will be no benefit from the prioritization. Therefore, advise the stakeholders that they should ideally assign their points unevenly to allow clear statements to be obtained. If a stakeholder does in fact assign identical numbers of points to every requirement, you can also reject these evaluations and ask for a new evaluation.

The stakeholders must also submit their evaluations independently of one another to avoid influencing one another. Independent voting can be achieved via a questionnaire or individual interviews. In a vote within a group session, one stakeholder could "repair" an evaluation made by a colleague that they deem to be incorrect by counteracting this evaluation. However, even knowing the preferences of the other evaluators (without knowing how many points they have assigned) influences the voting procedure. If a stakeholder knows that other stakeholders will give his favorite requirement a low value, but give other important requirements a lot of points, this first stakeholder will award his favorite requirement more points, trusting that the other requirements will receive their points from someone else. Again, at the end, all requirements will appear to be equally important.

The 100-dollar technique can also be applied in a variant with 1,000 or 10,000 units. A greater number of points naturally allows more differentiated evaluations. However, the prioritization also causes more effort, and it is more difficult to check how many points a stakeholder has awarded in total. From a practical perspective, this requires a tool.

4.5.8 Kano classification

In the Kano model [Kano1984], requirements are classified and prioritized in three categories with respect to user expectations. The Kano model is already described in detail in the handbook for the Foundation Level [PoRu2011] and is therefore repeated only briefly here. In this technique, the requirements are assigned to one of three categories:

- *Basic factors* are requirements where the users take fulfillment of the requirement for granted. Therefore, fulfillment of the requirement does not make them explicitly satisfied, but if the requirement is not fulfilled, they are very unsatisfied.
- *Performance factors* are requirements explicitly required by the users. Fulfillment of these requirements makes the users satisfied; non-fulfillment makes them dissatisfied.
- *Excitement factors* are requirements that the user does not expect. If the requirements are not fulfilled, the user does not notice. However, if the requirements are fulfilled, the user is excited about this innovation.

To determine which Kano category a requirement belongs to, ask the users two questions: How satisfied would you be if the requirement were fulfilled (satisfaction)? How dissatisfied would you be if the requirement were not fulfilled (dissatisfaction)?

Figure 6 shows a matrix for a two-criteria prioritization according to Kano. You can see that in addition to the three requirements categories referred to above, there is a further category, which appears only rarely: the insignificant requirement.

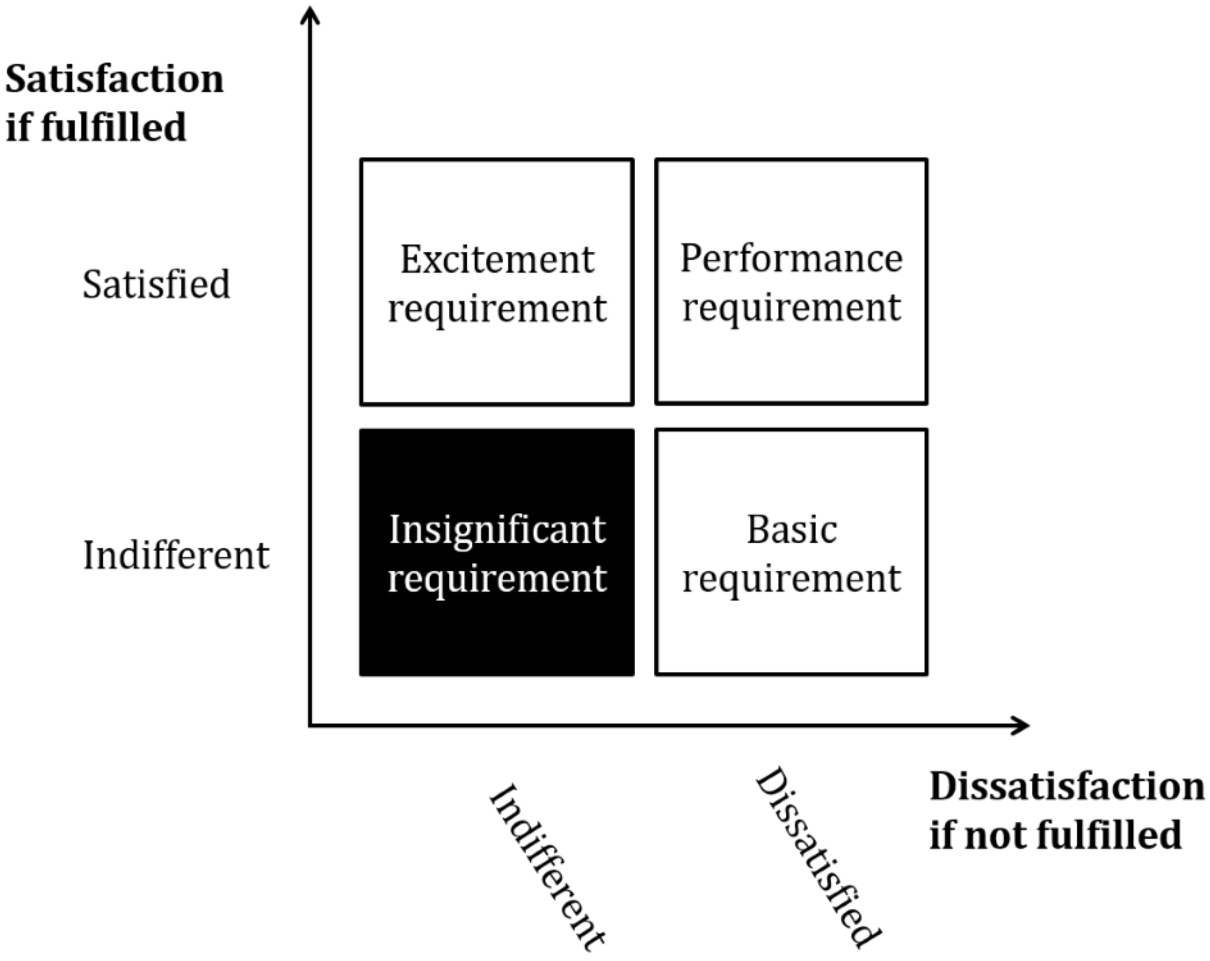


Figure 6: Matrix for a two-criteria prioritization according to Kano

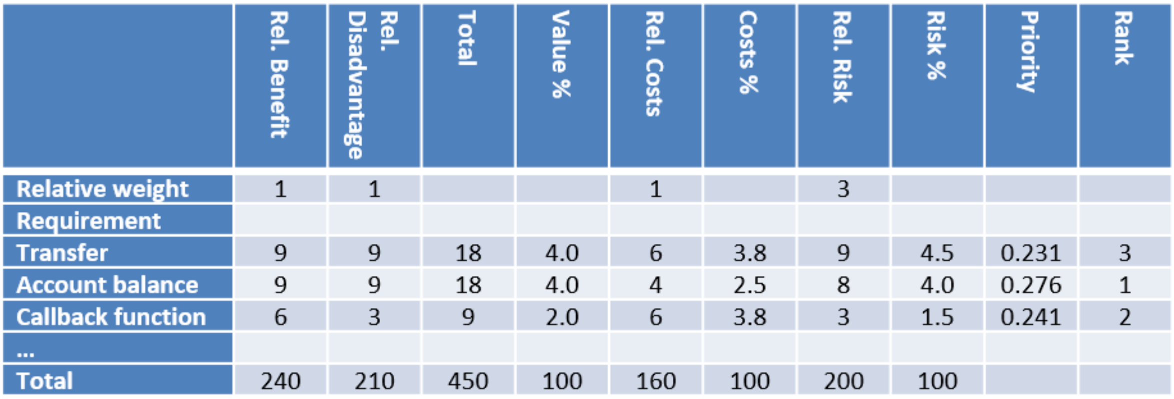
4.6 Analytical Prioritization Techniques

The ad-hoc prioritization techniques have the advantage that they are easy to use and very efficient. However, their results are subjective and often impossible to trace at a later point in time. They are not optimal for critical decisions or in a security-critical environment. In these situations, analytical prioritization techniques allow a more neutral and more traceable prioritization. We present two techniques here:

- Prioritization matrix according to Wiegers
- The Analytical Hierarchy Process (AHP)

4.6.1 Prioritization Matrix according to Wiegers

The prioritization matrix according to Wiegers is a prioritization technique that uses more than two criteria to prioritize requirements. It compares the relative advantage (of the fulfillment) and relative disadvantage (of the non-fulfillment) of every requirement with the relative costs and the relative risk of this requirement [WiBe2013]. Figure 7 shows an example or rather an extract from such a matrix for our online banking system.



	Rel. Benefit	Rel. Disadvantage	Total	Value %	Rel. Costs	Costs %	Rel. Risk	Risk %	Priority	Rank
Relative weight	1	1			1		3			
Requirement										
Transfer	9	9	18	4.0	6	3.8	9	4.5	0.231	3
Account balance	9	9	18	4.0	4	2.5	8	4.0	0.276	1
Callback function	6	3	9	2.0	6	3.8	3	1.5	0.241	2
...										
Total	240	210	450	100	160	100	200	100		

Figure 7: Prioritization matrix according to Wiegers for an example

The procedure for calculating the priority is as follows:

1.) Create or obtain a template

2.) Define and enter the weighting of the prioritization criteria: in our example of the online banking system, the risks outweigh the costs, for example.

3.) Add the list of requirements to be prioritized. As noted previously, these requirements should be at the same level of detail. We use example usage scenarios here.

4.) Evaluate each requirement with reference to the prioritization criterion "Benefit" that fulfillment of the requirement brings, using the matrix according to Wiegers on a scale from 1 to 9

5.) Evaluate each requirement with reference to the prioritization criterion "Disadvantage"—that is, the disadvantage of non-fulfillment of the requirement, again on a scale from 1 to 9

6.) Calculate the total value of both evaluations as a weighted total, weighted according to the weighting factors. This total can be calculated automatically in a template.

7.) Calculate the percentage value of every requirement with reference to the total list of requirements: *total value/total of all total values*. This can also be calculated automatically.

8.) Evaluate each requirement with reference to the prioritization criterion "Costs", again on a scale from 1 to 9

9.) Calculate the percentage portion of the costs based on the total costs: *costs / total of all costs*

10.) Evaluate each requirement with reference to the prioritization criterion "Risk", again on a scale from 1 to 9

11.) Calculate the percentage portion of the risk based on the total risk: *risk / total of all risks*

12.) Calculate the priority of each requirement according to the following formula: *Priority = value %/(costs% x weighted costs + risk% x weighted risk).* This is a prioritization according to the cost/benefit ratio, whereby the risk is added to the costs. (Other prioritization techniques deduct the risk from the benefit.)

13.) Determine the rank of each requirement: the higher the priority of a requirement, the higher its rank.

Prioritization of the requirements with the prioritization matrix according to Wiegers

In our case study, the requirements manager Peter Reber has decided to use the Wiegers method because in the bank environment, decisions must be taken carefully and based on solid reasoning. In addition to the matrix according to Wiegers, the justifications for the respective evaluations are also to be documented—that is, why each number of points was assigned in the respective field.

The values are evaluated by the following stakeholders: with reference to the benefits and disadvantage, the usability expert performs requirements triage and sorts out the requirements that are very important and those that are very unimportant and assigns points to them. The Customer Advisory Board determines the benefits and disadvantage of the remaining requirements. With regard to the costs, the developers are questioned. They determine the costs together by means of planning poker. The IT security experts investigate the risk. To do so, they first use a risk analysis to determine which risk events can even occur with reference to a requirement. Then, based on their experience, they use AHP to determine the probability of occurrence, and an analysis of company-specific statistics and key figures to determine the damage.

Figure 7 shows the results of the evaluations or rather an extract from the results. We can see that the relatively unimportant but less expensive and less risky callback function has made it into second place, ahead of the transfer.

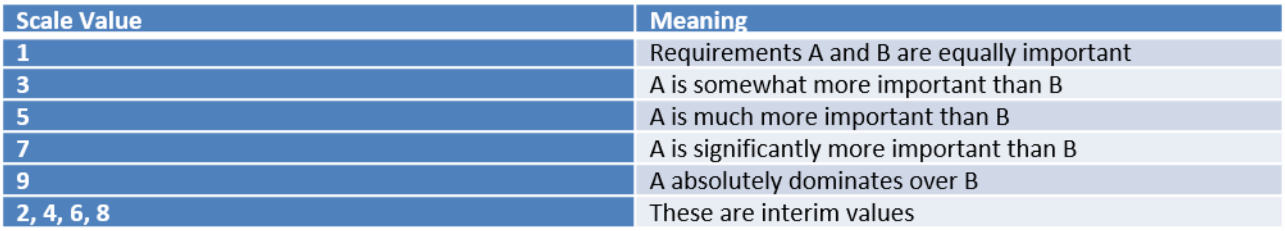
Practical tip: Of course, the result of the prioritization depends on the prioritization criteria selected and the technique used! If you had classified the above-mentioned requirements with the Kano method, as a basic function of online banking, the transfer would naturally have been more important than the callback function. Therefore, make sure you select the prioritization criteria wisely!

4.6.2 The Analytical Hierarchy Process (AHP)

The analytical hierarchy process (AHP) is a mathematically sophisticated, theoretically very interesting model. To benefit from the advantages of AHP, it is advisable to use a tool that supports the method and performs the required calculations.

The basic idea of the method, however, is simple. The prioritization is simplified for the evaluator, who has to compare only two requirements as a pair: which of the two is more important (or more expensive or riskier) than the other and by how much? Any prioritization criterion is possible here. This decision between two requirements is easier to make than deciding which requirement from an entire list of requirements is the most important. However, the disadvantage of the method is that each requirement has to be compared in a pair with every other requirement. For *n* requirements, this results in *n (n-1) /2* comparisons. This creates a lot of effort. An example calculation for the effort is given in the practical tip below. The great strength that no other method can demonstrate is that this method can balance out errors made by the evaluators. If, in the case of three requirements A, B, and C, A has been deemed to be more important than B, and B more important than C, then A must also be more important than C. However, if C is deemed to be more important than A, there is obviously an error. The method can measure how good and reliable the evaluations are overall using a "*consistency ratio*". The mathematics of the method were described by the inventor Saaty [Saat1990]. A summary of the method is given by Karlsson and Ryan [KaRy1997].

Here, we are particularly interested in the procedure from the evaluator's view. The evaluator receives two requirements for selection and has to decide which of the two is more important. The scale from Figure 8 is used for this. Apart from the fact that the evaluator has to do this for a large number of pairs of requirements, there is no further difficulty for the evaluator in this method. A tool or an expert performs the evaluation and then determines the priorities for the requirements.



Scale Value	Meaning
1	Requirements A and B are equally important
3	A is somewhat more important than B
5	A is much more important than B
7	A is significantly more important than B
9	A absolutely dominates over B
2, 4, 6, 8	These are interim values

Figure 8: AHP scale for the comparison of two requirements

Open source tools are available to support AHP, for example PriEsT [SMK2013], [PriEsT].

Practical tip: For each requirement, the matrix according to Wiegers requires an evaluation of four different values. You have to assume that each evaluation takes 1–2 minutes. Therefore, if you have 100 requirements in your list, the effort involved is 400–800 minutes, 6.7 to 13.3 hours, without any breaks.

In planning poker, with a disciplined process, prioritizing one requirement takes five minutes. 100 requirements therefore take 500 minutes, 8.3 hours.

AHP has the worst average: for 100 requirements, you have to perform 100 x 99 / 2 = 4950 comparisons. If every comparison takes between half and one whole minute, this results in an effort of 41 to 82 hours, 5–10 working days per evaluator.

You should therefore consider carefully in advance which requirements you want to prioritize and how many people are really needed for this. The ad-hoc techniques usually evaluate only one or two criteria for each requirement, and therefore they require correspondingly less effort provided time is not lost through long discussions. If we assume 1–2 minutes per requirement, the result for 100 requirements is 1.6 to 3.3 hours.

The use of the matrix according to Wiegers and the AHP method is recommended only for requirements lists with a maximum of 30 requirements [WiBe2013], [Mois2002].

4.7 Combining Prioritization Techniques

Different prioritization techniques have different advantages and disadvantages. The ad-hoc techniques are easy to use but lead to less traceable and not completely objective results. The analytical techniques are better, but do not scale well. In the prioritization matrix according to Wiegers, four evaluations have to be performed for each requirement, and with AHP, the prioritization of double the number of requirements does not produce double the amount of effort, as is the case for most techniques, but rather four times the effort. Where requirements lists are long, therefore, an effort of many hours or even days arises.

As many projects work with hundreds or even thousands of requirements, pragmatic solutions are required. In most cases, it is not necessary to prioritize the entire requirements list in detail. You can save a lot of time but still achieve an almost identical prioritization quality by combining an ad-hoc technique with an analytical technique.

For example, in the first round of the prioritization, you can use an ad-hoc prioritization technique to reduce the number of requirements to be considered. If, for instance, you want to determine the most important requirements for the next release, you do not need a particularly sophisticated prioritization for the requirements that are currently less important as these will initially be deferred in any case. Even for the apparently urgent requirements, no further differentiation is necessary. However, a closer examination is worthwhile for requirements that initially appear to be approximately equally important, but one is to be included in the release and others have to be postponed. In this requirements group, you can now use an analytical technique to draw the dividing line between the requirements that will be included in the release and those that will not.

In our case study in Section 4.6.1, we have already presented possible combinations of different prioritization techniques. Which technique you use and how you use it depends on which criteria are to be used for the prioritization, what the goal of the prioritization is (determine the top ten? Weigh up two groups of requirements?), how much time is available, whether one or more persons are to be questioned (some methods are less suitable for decision making within a group), and the level of knowledge of the stakeholders.

4.8 Content for the Requirements Management Plan

When creating the requirements management plan, you have to define the criteria to be used to prioritize the requirements (based on which decision), when they are to be prioritized, by whom, and using which technique. Make sure that the attribute schema contains the attributes that correspond to these prioritization criteria. This is the only way to document the priorities in the attributes in the requirements management tool and to evaluate the priorities—for example, to filter out the most important requirements for the release or iteration planning.

4.9 Literature for Further Reading

[Cohe2005] Mike Cohen: Agile Estimating and Planning, Prentice Hall International, 2005.

[Davi2005] Alan M. Davis: Just Enough Requirements Management - Where Software Development Meets Marketing. Dorset House Publishing, 2005.

[Ma2009] Qiao Ma: The effectiveness of requirements prioritization techniques for a medium to large number of requirements: a systematic literature review. Master Thesis, AUT University, 2009, http://aut.researchgateway.ac.nz/bitstream/10292/833/3/MaQ.pdf.

5 Version and Change Management

Life as a whole is marked by changes and new requirements that are motivated partly by external factors and partly by internal factors. In the same way, we encounter change requests in all projects, from town planning to software development.

Changes are not necessarily bad in themselves and they take place regardless of how well the originally accepted contractual basis or the accepted requirements document was. In the development of technical systems in particular, beyond the phase of requirements elicitation, we experience a strong rise (albeit not constant) in the awareness of problems and solutions through lessons learned over time.

> "*Requirements are rarely static. Although from the development management perspective, it is desirable to freeze a set of requirements permanently, it is rarely possible. Requirements that are likely to evolve should be identified and communicated to both acquirers and the technical community. A core subset of requirements may be frozen early.* The impact of proposed new requirements are evaluated to help ensure that the initial intent of the requirements baseline is maintained or that changes to the intent are understood and accepted by the acquirer." [ISO29148]

To ensure that you keep changes under control and that you are not controlled or overwhelmed by changes, as the requirements manager, it is particularly important that you are prepared for handling changes. Therefore, in the requirements management plan, plan how you want to handle changes as part of the elicitation of requirements and in the subsequent phases of the project. The following sections explain the basic concepts for finding your way in the jungle of continually changing requirements and requirements documents (version control). The sections also explain the reasons for change and how these changes can be implemented by a change management process.

5.1 Versioning Requirements

Versioning requirements enables you to track the development of a requirement over its entire lifecycle.

This means that by versioning requirements, at any point in time we can:

- Make statements about the frequency of changes
- Check the evolution of individual requirements
- Access previous versions of requirements

In direct conjunction with versioning requirements, we have to look at configuration management. Here, specific sets of requirements versions are grouped in a requirements configuration (see Section 5.1.2).

Software configuration management is the discipline for tracking and controlling the evolution of software. It is essential for the development and maintenance of large, long-lasting software systems [BSB2008].

The following sections describe:

- How a version control can be implemented for requirements
- What requirements configurations are
- What requirements baselines are
- Important points in the parallel further development of requirements

5.1.1 Version Control for Requirements and Requirements Documents

Version control for requirements refers to the process that enables specific development statuses of requirements and requirements documents to be kept available throughout the lifecycle of a system or product.

Definition 5-1: Version control: Version control (or a version control system) is used to document, manage, and restore documents, files, and individual artifacts (e.g., requirements). Version control allows you to trace changes to documents and artifacts and to revise changes made so that you can return to old versions. Version control therefore enables a sequential consideration of the evolution of a document or artifact over its entire life.

However, before we look at versioning and version control for requirements, we will digress briefly to look at the statuses of requirements, as although statuses and versions are closely related, and are therefore often mixed or confused with one another, they are actually two different concepts.

Statuses of requirements

Definition 5-2: Status according to [RuSo2009]: "Statuses specify the progress of the processing of the requirement. *If we compare the life of a requirement with a project plan, then the statuses of the requirement often correspond to the milestones in the project plan."*

If we look at the evolution of an individual requirement or a requirements specification, over the course of its lifecycle, this requirement or requirements specification will have different statuses: for example, starting with "Created" when the requirement is recorded, through "In evaluation", "Released", and so on, see Figure 9. For an individual requirements artifact, these statuses can be documented via an attribute, for example (e.g., status), for the respective requirements artifact (see Chapter 3).

In contrast, documents often contain an introductory part which, in addition to the title, the author, the date of the last change, and the version number, also contains the status of the document. In this case, the document status is generally dependent on the status of the individual requirements in the document.

You can define which statuses and status transitions are permitted for requirements artifacts or documents individually for your project. The required statuses and status transitions are dependent on the project being executed and the requirements engineering process, including the planned review cycles.

Figure 9 illustrates a simple status machine that presents the possible statuses and status transitions for a requirements artifact. For example, a requirement can be created as version 0.1, move to the status "In evaluation", be released, and even implemented, without there ever being a change to the content of the requirement that would have necessitated a new version.

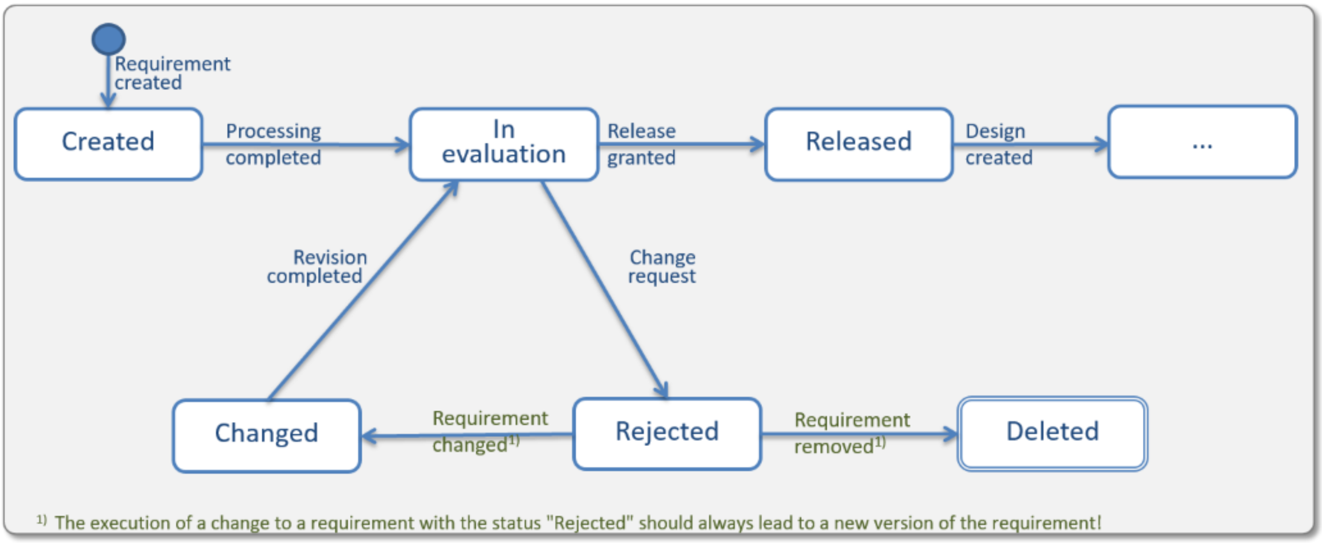


Figure 9: Statuses and status transitions of a requirement

Versions, versioning, and version control

Compared to the status of a requirement—which represents, for example, a project-specific lifecycle of the requirement—a requirement version describes a specific content status of a requirement. It is therefore possible that a requirement with the status "Created" will go through several version statuses before being set to the status "In evaluation". The same applies for a requirement that has been rejected which may be changed in multiple iterations.

Definition 5-3: Version: A version is a specific content status of a requirements artifact or document at a specific point in time. Versioning allows you to trace the history of a requirements artifact or document back without any gaps and reset it to an earlier version. Changes to content always lead to new versions.

From this point on, we refer to the process of creating new versions as versioning. Versioning can take place at different levels (e.g., at document level or at the level of atomic requirements).

- In the case of versioning at the document level, every change to the content of a document (e.g., a change to one or more requirements within the document) must lead to a new version.
- In the case of versioning at requirement level, every change to the content of a requirements artifact must lead to a new version of the requirement.

With regard to versioning, note that the "new" version always completely replaces the "old" version. If, therefore, you describe one or more textual requirements with a model-based description (e.g., activity diagram), and the model-based description is merely a supplementary (formalized) view, this is not a new version of the original requirement. This is actually a supplementary description which may and should exist in parallel. To make this dependency clear, you can use traceability relationships—we discuss these in Chapter 6.

Versioning enables a version control which allows the requirements manager, for example, to compare different documentation statuses (versions) with one another or to go back to previous documentation statuses (versions) (see **Definition 5-1: Version control**). According to [WiBe2013], version control includes the following activities:

- **Definition of a schema to identify versions**: Define the schema to be used to version requirements and requirements configurations and documents. For example, a new version of the requirement is created by incrementing the version number, but the requirement ID remains unchanged.
- **Identification of versions of individual requirements**: Define how changes to individual requirements should be identifiable. Define, therefore, what information must be recorded to document the change to the last requirement version sufficiently.
- **Identification of versions for requirements configurations** (or documents): Define how changes to requirements configurations should be identifiable. Define, therefore, what information must be recorded to document changes to the last document version sufficiently.

There is no fixed, prescribed specification for versioning requirements or documents. In principle, you can identify different versions with whole version numbers (i.e., 1, 2, 3, etc.). However, the recommendation is to use a versioning based on increments, so that the version number gives a first indicator of whether the change is a fundamental change or a marginal adjustment (e.g., correction of a spelling or grammar mistake). An increase in the increment generally represents a marginal adjustment to the content, whereas the increase to a full version represents an extensive adjustment to the content. The classification as a marginal or extensive change is of course primarily a subjective decision. However, these decisions can be objectified with some conventions.

Versioning requirements documents

When versioning documents, the recommendation is to always use a tool for version management (version control system). If you are versioning documents manually, it makes sense to use a versioning indicator (e.g., based on increments) in the file name. This enables a dedicated version to be created for every revision to the document.

To document a change, it is also important that a document has a document history (on the first pages, see Table 4) so that the changes performed can be recognized at a glance. The document history should always contain at least the following information:

- The new version number of the document
- The date on which the change was performed
- The person who made the change
- The changes that were made
- The reason for the change

Version	Date	Name	Change/Reason for Change
0.1	2014-09-19	Reber	Initial version
0.2	2014-09-20	Reber	Changes to requirements Req-0010, 0011, 0030, 0090
0.3	2014-09-30	Reber	Changes to the priority of requirements Req-0010, 0011
1.0	2014-10-02	Reber	Version created for first review
1.1	2015-10-15	Reber	Changes maintained based on the review results, Req-0030, 0034, 0035, 0089, 0090

Table 4: Example of a document history

Versioning requirements artifacts

A requirements management tool is recommended for versioning requirements. Nevertheless, versioning can also be performed without a tool.

When you change requirements (i.e., when you create a new requirement version), as a minimum, the following information must be documented to describe the change compared to the previous version:

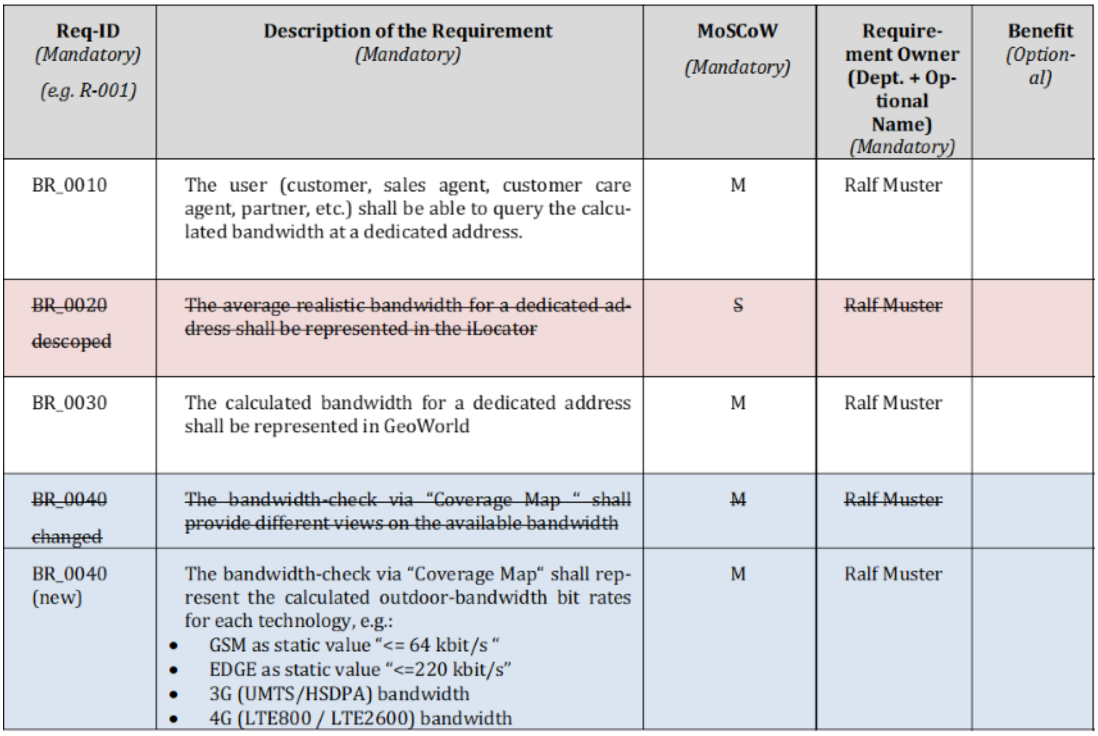
- The new version number of the requirement (whole number or increment)
- The change action performed compared to the last baseline (e.g., deletion)
- The change made to the content of the requirement
- The reason for the change (i.e., what or who was decisive for the change)
- The name and role of the person who performed the change
- The time of the change (date + time)

Req. ID	Date	Version	Name	Reason for Change	Action	Requirement
Req-30	2014-09-19	1	Reber		Created	The system should (a)
Req-30	2014-09-20	2	Reber	Changes due to new information from the department	Changed	The system should (b)
Req-30	2014-10-15	3	Reber	Change due to a review by Max Muller	Changed	The system should (c)
...						

Table 5: Example of requirements versioning

Note: If you are using a requirements management tool, some of this information (e.g., new version number of the requirement, the person making the change, the time of the change) will be documented automatically without you having to invest additional time here.

Before Peter Reber's time, requirements were described in Word and Excel documents. Nevertheless, a minimum level of attribute assignment and versioning was observed. The example below shows an extract from an old requirements document. The left column shows the requirement ID together with the version number—certainly not the best method of documentation, but better than nothing. Revised or deleted requirements were given a new status and ID accordingly. We can see two versions for requirement BR_0040: the rejected v1 ("Revised") and the current v2 ("Modified"). The information about why the requirement was changed, who triggered the change, and when the change was performed is not visible here. Nevertheless, there is at least a minimum versioning which, in the future, should be performed automatically with a tool as soon as changes are made to a requirements artifact.



Req-ID *(Mandatory)* *(e.g. R-001)*	**Description of the Requirement** *(Mandatory)*	**MoSCoW** *(Mandatory)*	**Requirement Owner (Dept. + Optional Name)** *(Mandatory)*	**Benefit** *(Optional)*
BR_0010	The user (customer, sales agent, customer care agent, partner, etc.) shall be able to query the calculated bandwidth at a dedicated address.	M	Ralf Muster	
~~BR_0020~~ ~~descoped~~	~~The average realistic bandwidth for a dedicated address shall be represented in the iLocator~~	S	~~Ralf Muster~~	
BR_0030	The calculated bandwidth for a dedicated address shall be represented in GeoWorld	M	Ralf Muster	
~~BR_0040~~ ~~changed~~	~~The bandwidth-check via "Coverage Map " shall provide different views on the available bandwidth~~	~~M~~	~~Ralf Muster~~	
BR_0040 (new)	The bandwidth-check via "Coverage Map" shall represent the calculated outdoor-bandwidth bit rates for each technology, e.g.: • GSM as static value "<= 64 kbit/s " • EDGE as static value "<=220 kbit/s" • 3G (UMTS/HSDPA) bandwidth • 4G (LTE800 / LTE2600) bandwidth	M	Ralf Muster	

Figure 10: Practical example of requirements versioning with Word

Practical tip: Requirements management tools are not yet used in all businesses and therefore, in everyday life in projects, we often encounter document-based requirements specifications (e.g., in Microsoft Word). A versioning at requirement level in the manner described above is therefore not possible without a lot of effort. If you find yourself in such a situation, use this procedure at least at document level and use the revision mode options or identify your requirement changes clearly with deletions and comments (see Figure 10). Of course, this is not the textbook method, but it at least indicates which requirements have been deleted and which have been changed. Furthermore, in documents, it is helpful to place a document history at the beginning of the document to give readers a quick overview of the history of the document and the changes it contains (see Table 4).

Access to current versions

Make sure that all relevant project participants can access the currently valid (released) requirements versions; this is not necessarily the same as the latest version of the documentation, which may be in review, for example, and not yet released. However, the processors of the requirements and the requirements manager must be able to access the latest version of a requirement at any point in time.

It is also important that changes that have led to new versions of requirements are communicated actively to all stakeholders at defined points in time [WiBe2013]. This communication usually takes place at the time of the review, when change configurations (see Section 5.1.2) are put together.

Note: The active communication to the project participants can also be performed by a requirements management tool if, for example, you have defined beforehand who is to be informed in the event of a change.

Implementing measures for version control

As the requirements manager, when you create the requirements management plan—that is, before you document the first requirement—you must define how you want to implement a version control for requirements and requirements documents in your project.

It is the requirements manager who decides the level (document level or requirement level) at which version control is to take place, and this decision is dependent on the project scope. The trend is that for complex projects with hundreds of requirements, version control should take place at requirements level, even if this means that the effort involved is significantly higher. However, this pays off over the duration of the project. Via the versioning at requirement level, you can make sure that you are always talking about the same version of a requirement over the course of the entire project. This means that you know which requirement version was in which requirements configuration (e.g., for acceptance or for development), and you can therefore discuss or distribute a specific version of a requirement explicitly.

In addition to specifying the level at which versioning is to be performed and the information that must be documented for new versions, in the requirements management plan, you must also define who is permitted to perform changes and at what level.

In principle, only a limited group of persons should be authorized to make changes (see [WiBe2013]). These roles and rights must be documented in a roles and rights matrix (e.g., RACI) (see also [Oran2013], RACI Model).

Change management limitation

Up to this point, we have discussed the implementation of a version control at different levels to allow documentation and tracking of any changes at requirement level, for example. There are many reasons for such changes.

We want to differentiate between two main points in time within a project when changes occur:

- Changes that occur as part of requirements engineering up to their final acceptance or release of the requirements specification

- Changes that occur after final acceptance or release of the requirements specification and thus require a retrospective scope change

Changes of the first type can usually be considered directly and flow into the specification as a new version of a requirement if they do not require a fundamental change to the project scope. Changes of the second type must always be processed via a regulated change management process.

5.1.2 Requirements Configurations

As part of the elicitation of requirements, at certain points in time you create requirements configurations—for example, to allow the performance of a review at a defined and consistent status of your requirements, or to obtain an estimation of the effort for the subsequent development phases for the requirements configuration.

Definition 5-4: Requirements configuration according to [IREB2015]: "A requirements configuration comprises a defined set of logically related requirements, whereby at most one version of each requirement is contained in the requirements configuration."

A requirements configuration is therefore a specific set of requirements (requirements artifacts) which is provided, for example, for review at a specific point in time and contains a specific version of the requirements. The following definition also highlights the communications aspect of configuration management so that all reviewers or users of the version involved receive a standardized and consistent and contiguous requirement status.

Definition 5-5: Configuration management from [IEEE 29148]: "*The purpose of the Configuration Management Process is to establish and maintain the integrity of all identified outputs of a project or process and make them available to concerned parties.*"

According to [PoRu2011], requirements configurations have the following properties:

- **Logical connection**: The selected requirements versions of a configuration are connected logically and are selected for a specific purpose.
- **Consistency:** The combined requirements and requirements documents are consistent and belong together logically.[1]
- **Uniqueness**: The configuration for the selected requirements versions has an identifier that identifies it uniquely.
- **Unchangeability**: The configuration is based on a specific version status of the requirements. Changes to these requirements versions lead to new versions that can be used in new configurations.
- **Basis for reset**: Configurations offer defined statuses to allow requirements to be reset to an older, consistent requirement status (version status).

[1] In practice, configurations are often created that are not consistent in terms of content. Such configurations are built out of the need to freeze the current work status in order to be able to access it later if necessary. For example, a configuration can be created that documents the starting point of review activities.

Note: The creation of a requirements configuration can be considered, for example, as a consistent requirements document, which is to be checked and accepted, with selected versions of requirements artifacts for a planned project phase. Compared to the requirements baseline (see the following section), the requirements configuration does not necessarily have to contain only stable requirements artifacts. The focus here is more on the logical connection in a "requirements composition" (i.e., a requirements configuration).

5.1.3 The Requirements Baseline

You should already be familiar with the term requirements baseline from the CPRE Foundation Level. Baselines are generally a "frozen" documentation status which is created, for example, when certain milestones (handover of a specification for cost evaluation) are reached.

Definition 5-6: Requirements baseline: Requirements baselines are selected, formally checked, and released requirements configurations that cover stable requirements artifacts and often reflect a fixed development and delivery status for a product (e.g., for a specific product release).

Requirements baselines are therefore generally visible to the outside world, whereas simple requirements configurations are used primarily for internal purposes (see [WiBe2013] and [Pohl2010]).

Definition 5-7: Release management according to [BSB2008]: "Release management is concerned with bundling requirements for a product, with the scheduling for the manufacture and, ultimately, the delivery of a finished system. [...] For a release, all current configuration elements are usually managed under one common label and the software is then created from the configuration thus created."

The requirements configuration defined as the requirements baseline should contain only requirements planned for a particular version of the product (e.g., release) and which are stable, and not those that are only proposed or are still in progress or being discussed at this point in time [WiBe2013]. Therefore, when selecting requirement versions for a requirements baseline, pay attention to their status (see also Chapter 3).

Requirements baselines support three essential activities in the development process (see [Pohl2010]):

- They form the **basis for planning delivery increments (releases)** because for the customer, they represent a visible configuration of stable requirements versions.
- They are used to **estimate the implementation costs** of a particular release.
- They enable a **comparison with competing products** on the market with the defined release.

A suitable point in time for creating a requirements baseline can be when a milestone is reached: for example, the commissioning of the design of the architecture or the implementation (see [WiBe2013]). Milestones for requirements baselines and for configurations are generally specified by the project or the development process.

Figure 11 shows two example milestones: *"For review"* and *"Creation of the architecture design"*. For the first milestone, versions of requirements artifacts that have the status "In evaluation" can also be used. In contrast, for the second milestone, only versions of requirements artifacts that have the status "Released" should be used.

Note: In your requirements management plan, define the purpose for which requirements baselines are to be created, who is permitted to create requirements baselines, and last but not least, the criteria for selecting requirements artifacts for a requirements baseline.

Consider the requirements artifacts contained in the requirements baseline as an accepted and commissioned specification. The requirements contained herein can only be adjusted via a controlled change management process.

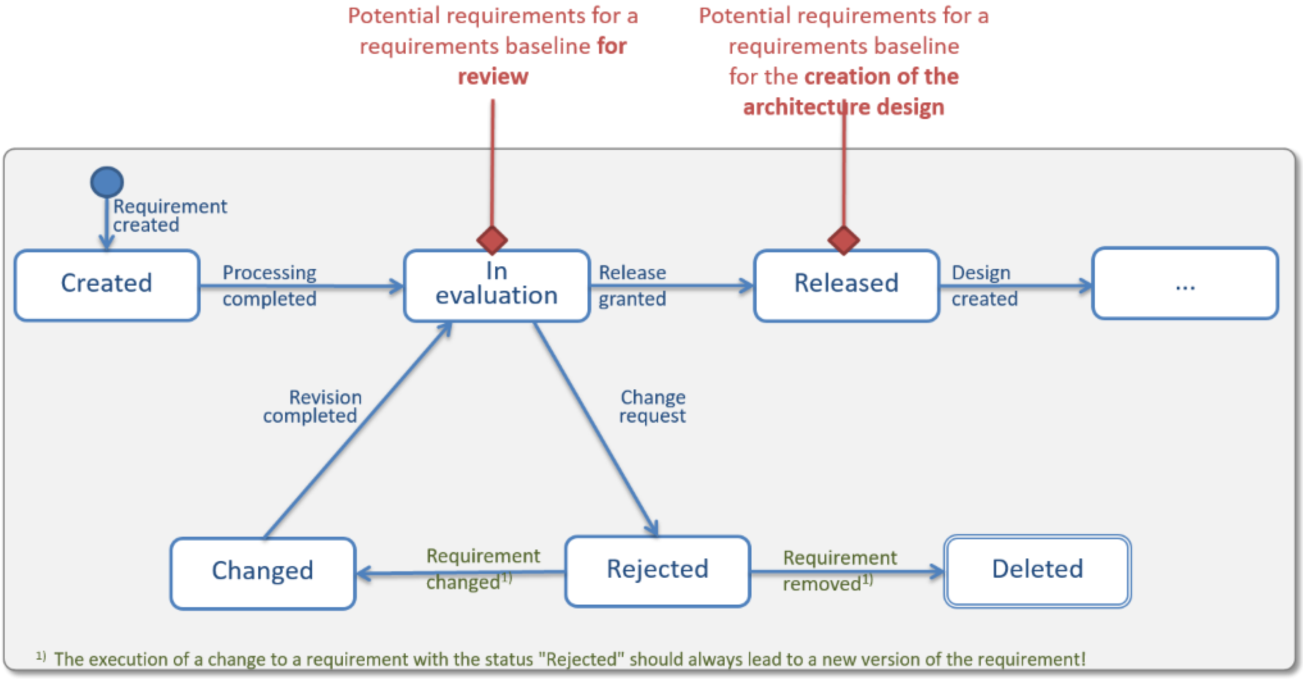

Potential requirements for a requirements baseline for review
Potential requirements for a requirements baseline for the creation of the architecture design
Requirement created
Created
Processing completed
In evaluation
Release granted
Released
Design created
...
Revision completed
Change request
Changed
Requirement changed¹⁾
Rejected
Requirement removed¹⁾
Deleted
¹⁾ The execution of a change to a requirement with the status "Rejected" should always lead to a new version of the requirement!

Figure 11: Possible milestones for requirements configurations and requirements baselines

5.1.4 Branching Requirements

The term *branching* originates from configuration management and allows the parallel development of systems in different development branches. Branches are used, for example, as part of fixed, scheduled releases to start the further development for the subsequent release on one branch, while on the parallel "production branch" of the system that has already been delivered, only bug fixing and minimal changes may be performed. The two development branches are then generally merged again before the next main release so that, for example, the errors that have already been corrected in the production branch do not find their way back into production with the new release.

Even though the term *branching* originates from configuration management and is therefore more closely associated with implementation, we also find this concept in requirements management, as the branches in the requirements strands reflect the branches in the development strands.

In contrast to versioning requirements, branching requirements allows multiple versions of requirements to be valid in parallel simultaneously. The branching mechanism is used, for example, to perform small and urgent changes in parallel with ongoing development.

To create a requirements branch, a valid requirements configuration (e.g., the last requirements baseline) is selected that the new requirements branch should build on. This configuration is copied to a new requirements branch, changed, versioned, and summarized in a new requirements configuration. Think of a requirements branch as, for example, a copy of a selected document version that can be worked on in parallel. The main point here is that the same requirement may exist in two branches in parallel, and there is thus one valid version of a requirements artifact for each branch.

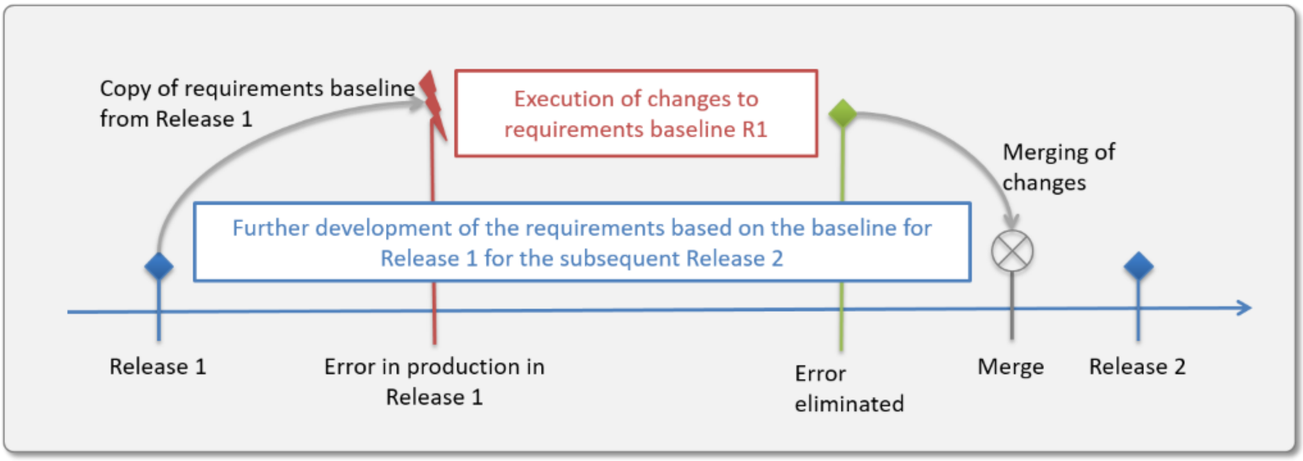

Copy of requirements baseline from Release 1
Execution of changes to requirements baseline R1
Merging of changes
Further development of the requirements based on the baseline for Release 1 for the subsequent Release 2
Release 1
Error in production in Release 1
Error eliminated
Merge
Release 2

Figure 12: Branching and merging of requirements configurations

Once the branched requirements configuration has been successfully implemented, at a later point in time—generally before a new release—the two branches are merged again. From this point on, there is again only one version of the respective requirements artifacts so that the subsequent changes only have to be performed in one version, see the example in Figure 12.

In addition to refining and versioning requirements, requirements branching is an additional dimension of the complexity of handling requirements in requirements management which should be used sparingly and deliberately. Otherwise, the uncontrolled use of requirements branches can lead to more chaos than benefit.

Problems that occur in connection with requirements branches include:

- Requirements branches make it more difficult to identify requirements uniquely
- In addition to versions and refinements, requirements branches increase the complexity of requirements engineering and requirements management
- Requirements branches generate redundant requirements information which must be maintained in parallel and then merged again in the long term

In individual cases, requirement versions that have arisen in requirements branches are intentionally not merged again and both requirement versions are intended to exist consecutively in parallel. In this case, however, we no longer refer to the same version in different requirements branches, but rather to variants (see Chapter 7). These variants are then managed with different requirement IDs so that the requirement variants can still be identified uniquely.

Practical tip: Requirements branches increase the complexity of managing requirements not only in requirements engineering and requirements management; in the subsequent phases of software development, parallel developments lead to challenges as separate development and test environments and teams must be available for every development branch. If software errors cause delays in the commissioning of branches, this can affect the acceptance and commissioning of subsequent releases. The number of requirements branches should therefore be kept low. Companies often have a parallel development branch which is used for bug fixing and small changes.

5.2 Change Management for Requirements

IEEE 29148 describes the nature of changes to requirements with the following words: "Whatever the cause of requirements changes, it is important to recognize the inevitability of change and adopt measures to mitigate the effects of change. *Change has to be managed by ensuring that proposed changes go through a defined impact evaluation, review, and approval process, and by applying careful requirements tracing and version management. Hence, the requirements engineering process is not merely a front-end task, but spans the life cycle.* In a typical project the activities of the requirements management evolve over time from elicitation to change management.“ [ISO29148]

Note: Be ready for changes and schedule them. Establish a simple and effective change process. The longer a project runs, the greater the probability of changes to your requirements. An approximate reference value is 1-5% changes per month (see also [Eber2012], [WiBe2013]).

The planned handling of changes is therefore a significant task in requirements management. What is important here is to accept that changes are the rule and not an exception.

With regard to changing requirements, we want to differentiate between two main times of change as these are usually handled differently:

- Firstly, the evolution of the requirement up to its final acceptance or release for the architecture design or the implementation. This is usually a time interval before the first requirements baseline. As part of this phase for eliciting, analyzing, and negotiating requirements, it is normal that changes are made to requirements without a separate change management process. Note the rules for versioning.

- Secondly, the evolution of the requirement after the final acceptance or as part of the creation of the design, implementation, or even during operation. These changes are also normal and can be driven by external factors (e.g., changes in legislation) or internal factors (e.g., new strategies). You should process these changes via a corresponding change management process! This is because these changes are generally those that were not estimated in advance either in terms of time or money and that have to be reevaluated via an impact analysis.

Definition 5-8: Change management according to [BSB2008]: "Change management regulates the further development of the product by monitoring in particular the change requests for the product and the processing of these change requests. Change management monitors the lifecycle of all change requests across the following steps: creation, evaluation, realization, testing, and acceptance."

5.2.1 Causes, Sources, and Timing of Requirement Changes

There are many reasons for changes to requirements. Requirements for a (software) system are subject to changes during the lifecycle of the (software) system. These changes can be triggered by different persons or roles, from different development phases, and in different project and lifecycle phases.

As a first step, it is helpful to know where changes to requirements originate and what the causes and sources of changes to requirements are. [RuSo2009] differentiates between the following **sources** for changes:

- **Incident management** (technical hotline for the systems): This is where malfunctions triggered from technical system operation and by system users, and which have to be rectified, appear. Changes can result from the analysis of these malfunctions.
- **Department and product management**: These groups of people generally create new requirements for the system which improve the use of the system or reflect new facets of the system.
- **Developers**: This group of people generally defines change requests relating to the technical implementation of the system. These changes do not directly influence the user functionality of a system.
- **Testers**: These people generally define changes aimed at rectifying errors that exist in the system (due to faulty or incomplete requirements).

According to [Pohl2010], causes of requirement changes include:

- **Errors in ongoing system operation**: changes due to incorrect system behavior that are reported as an incident by the user or application operation. These changes to requirements result from incorrect or missing requirements, and not from an incorrect implementation of the requirement.
- **Context changes**: changes that result from changes to constraints in the system context. These changes can originate from all aspects of the context (usage aspect, object aspect, IT system aspect, or the development aspect). These requirement changes result from a changing world and are submitted via the department, product marketing, or development.

The list above is not a complete list of causes of changes to requirements. It is intended primarily to give you an insight into why changes occur, and for what reasons and from what sources changes can originate. Therefore, think about which sources and reasons for changes you can expect as early as possible.

Note: When analyzing any change to be executed, in addition to the change to the actual requirement, you must also consider effects on directly and indirectly dependent requirements and other development artifacts (see Chapter 1). Of course, as the requirements manager, you are not directly responsible for the architecture design, the test, and the development, but when the change takes place in "your" requirements, the other roles must be informed about these changes so that the change can be evaluated in its entirety.

Changes can occur at different **times** during the entire project and lifecycle of a (software) system, for example:

- **During the elicitation of requirements**: Adding a new requirement leads to an adjustment to an existing requirement because the system context has changed.
- **During the architecture design**: An architecture decision for the system architecture requires that a function previously covered by hardware is to be realized by means of additional functionality in the software for cost reasons.
- **During implementation:** The implementation of a requirement demonstrates performance problems which, in turn, can only be resolved by adjusting the actual requirement.
- **During the software test**: A test result shows that a requirement has not been implemented in accordance with the specification but the implementation offers a better solution which is to be retained. In this case, the requirement must be updated accordingly.
- **During the acceptance test**: During acceptance, you establish that the customer does not accept the delivery because he envisioned a different implementation of the requirements but did not document this sufficiently. The requirements must be made more specific and the corresponding development artifacts must be revised.
- **During system operation**: When software is used, it can become clear that functionality that has been implemented has gaps for the processing of the business process and therefore, new requirements must be elicited and existing requirements changed.

The dimensions of the requirement changes (different sources, different causes, and different times) make change management a complex task that cannot be performed ad-hoc and on demand. Instead, a dedicated change management process is required (see Section 0).

5.2.2 Types of Changes to Requirements

As you can imagine, and have almost certainly experienced in your own life, no two changes are identical. If, for example, you ordered your car with an automatic transmission, but on delivery you discover that the car has a manual transmission, this results in a complaint (or in other words: a change request).

However, for you—and for the vehicle supplier—this type of change has a different significance to a change that you request 6 weeks after your order, namely that instead of "silver gray metallic" for your exterior finish, you would prefer to have "space gray metallic".

For change management, when handling changes, you should know what types of change exist so that you can develop a strategy for handling different types of changes. In the CPRE Foundation Level [IREB2015], the following classification of changes is proposed:

- C**orrective changes**: A change is corrective if it can be attributed to incorrect behavior during operation or of the product delivered, and the cause of the errors lies in the requirements.
- **Adaptive changes**: A change is adaptive if it can be attributed to new constraints, findings, or a context change. This type of change usually originates outside the project (e.g., legislation).
- **Exceptional changes**: A change is an exceptional change if it can be attributed to damaging behavior or would lead to damaging behavior. This type of change must be implemented as quickly as possible to limit the damage. It can be both corrective and adaptive.

To implement a change to a requirement, change requests must be submitted. These are then evaluated and processed by a change management process (see Section 5.2). Amongst other things, a change request covers the desired changes to the content of existing requirements (i.e., to the current requirements baseline).

Note: Not every change leads to an adjustment to the requirements. For example, software errors do not lead to a change to the actual requirements; instead, they lead exclusively to a change or correction of the implementation with reference to the (correct) requirements. From a customer perspective, the example of the incorrect type of transmission can clearly be classified as a "bug", as the customer's requirement (automatic transmission) was clearly documented.

These changes can be characterized as follows:

- The change requires the **integration of a new** requirement (usually a scope enhancement).
- The change requires the **deletion of an existing** requirement (usually a scope reduction).
- The change requires a **change to an existing** requirement, by means of addition, reduction, or a change to the content (scope change).

For your change management process (see Section 0), note that there are different types of changes that can require changes in "your" requirements. Define which change requests you want to discuss in your project cycle, who is permitted to submit these change requests, and what the request must look like. For example, as a corrective change, a change request to rectify a defect can look different to a change request for an innovation (that is, an adaptive change).

For his project, Peter Reber defines the types of changes he wishes to discuss in the project, what he understands under those types of changes, and who is permitted to submit these change requests and in what form. To do so, Peter uses the above-mentioned classification, although he uses different designations which have become established in the company.

Spec. error: *A "spec. error" is a corrective change and describes an error in the product which can be attributed to an incorrect description in the requirement specification.* These changes must be channeled exclusively *via the IT Service Desk* and are reported *as a change request via the ticket system*.

Scope change: *A "scope change" is an adaptive change and describes new requirements for the system from a user, company, or legal perspective*. These changes are usually submitted *via product marketing* and must be documented via the *template for change requests*.

Tuning request: *A "tuning request is an adaptive change and describes new technological requirements for the system to improve operability*. These changes are generally submitted *via the IT department* and must be documented using the *template for IT change requests*.

5.2.3 Analyzing and Documenting the Stability of Requirements

To move forward in a project, you have to finalize your requirements within a specified time frame so that you can complete your project *on time*, *within budget*, and *in quality*. To enable the client to see results and profit from their investment as quickly as possible, a phased or release-based approach is often selected in which the desired product is taken into production in stages. To do this, however, you have to know which requirements have already been agreed and are stable so that you can hand them over to development (see Figure 11 in Section 5.1.3).

For this selection, requirements should be classified with regard to their stability, and thus with regard to the probability of the current version of the requirement being changed. On the one hand, this type of classification for selecting requirements for a specific phase can be done solely by evaluating the stability (see Chapter 3), or alternatively, a corresponding prioritization technique (see Chapter 4) can be used which, in addition to the stability, includes other aspects—such as the expected benefits from the requirement—in the evaluation. The stability of the requirement should always be considered in the evaluation when selecting requirements for a target release because the stability is relevant, amongst other things, for estimating the risk of releasing a selected requirements configuration (requirements baseline) for implementation.

Practical tip: As mentioned at the beginning, a change rate of 1–5% per month can be expected after the phase of eliciting and documenting requirements. This means that if you have 1,000 project requirements, it is not uncommon for 20 requirements to change per month. If more than 10% of the requirements in your project change per month after the requirements have been released, together with the client, you should think seriously about the project goal to avoid a creeping scope extension.

At this point, you will justifiably ask how you can select a requirements configuration or how you can establish that a set of requirements (requirements configuration) has progressed so far that a requirements baseline can be created and handed over to development without the first changes to these requirements being submitted just a short time later. The answer is that nobody can predict this precisely.

However, even if you are not psychic, you can make a statement about the probability of changes to your requirements. The following rules (heuristics) will help you to evaluate the probability of changes to requirement groups in a short time with limited knowledge and incomplete information (see [VanL2009]):

- Requirement groups that serve the same goal and are generally highly stable (measured by the frequency of changes) have a lower likelihood of change than individual requirements.
- Goals are generally more stable than solution-oriented requirements.
- Functional requirements that meet the core goals are generally more stable than quality requirements.
- Functional requirements that repeatedly appear in the set of requirements (as amalgamations, extensions, or variants) are usually considered as stable requirements.
- Requirements describing alternative choices should be handled with particular caution and are generally less stable than the above, as decisions are often based on incomplete knowledge and assumptions.
- Requirements that are assigned to a variant or enhancement of the system are more stable than requirements that have not yet been assigned.
- Requirements that were frequently changed until very recently are unlikely to be stable.

Define, therefore, for yourself and for your team, the criteria according to which requirements baselines are to be created: that is, which requirements may flow into a requirements baseline (in the sense of defined evaluation criteria). Make sure that corresponding attributes for documenting the requirement status, the stability, the urgency, etc. are created at an early stage (see Chapter 3) and in particular that they are maintained so that at any point in time, you can select the correct requirements for stable requirements baselines.

5.3 Change Management Process

According to ITIL, change management ensures that changes are implemented within the IT infrastructure in a controlled manner. "The purpose of change management is to 'control the lifecycle of all changes, enabling beneficial changes to be made with minimum disruption to IT services'...by following a well-defined...process" within the organization [Oran2013].

The change management process achieves this by defining activities, responsibilities, and necessary artifacts that describe a clear procedure for handling change requests for requirements.

In most change management processes, the Change Control Board (CCB) plays an important role in the change process. In ITIL, it is called the CAB = Change Advisory Board. [WiBe2013] describes the Change Control Board (change committee) as a group of persons with different interests (e.g., project manager, developers, testers, IT department, Help desk) which, for every change request, decides whether and when it should be implemented.

The CCB decides whether, based on the impact analysis conducted, a change request is accepted, rejected, or postponed (see [WiBe2013]). The aim is to identify the effect that a change has on all directly and indirectly affected systems and processes.

The IT of the example bank where Peter Reber is employed works according to a company-specific project process. Therefore, Peter has a good basis for establishing a change management process and for defining interfaces to supplying and implementing processes.

In the following model, Peter has outlined the interfaces to Change Management. In the illustration, we can see that problems that are identified by customers and in the IT department are first evaluated by Problem Management before a change request is submitted. The change management process itself is implemented by the Change Control Board (CCB). Members of the CCB include the project manager, user and IT representatives, and Peter Reber as the requirements manager. Change Management receives change requests from the departments (e.g., product marketing, the legal department) as well as from Problem Management. Changes that are accepted and implemented by the CCB are handed over to Release and Deployment Management as implementation requests. Peter Reber's task is to obtain all the required information from the experts before the CCB meeting—for example, a cost evaluation, the importance of the change, effects on usability and security.

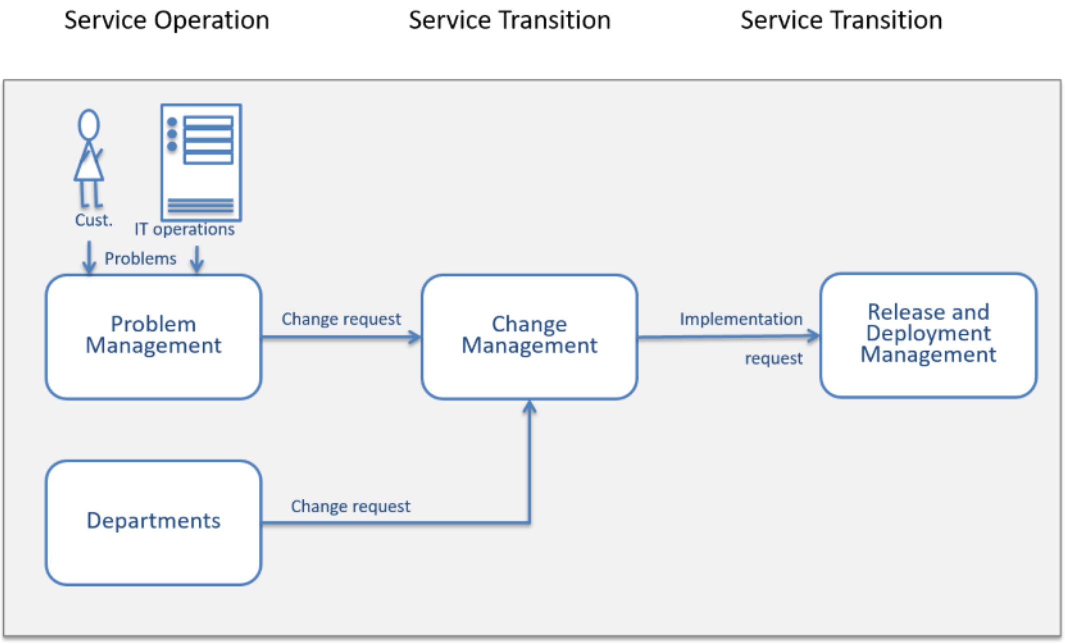


Figure 13: Example interfaces to Change Management

In the following we will concentrate on the change process itself. To ensure that you can integrate changes purposefully, plan a simple and efficient change process for your project. [WiBe2013] provides a few useful tips in this regard:

- Define the goal of the change management process.
- Define the roles and responsibilities in the change management process.
- Define the input criteria for change requests.
- Define the unique statuses and status transitions that a change request can progress through.
- Define a "lean" change management process.
- Define output criteria for the process.
- Define how changes are to be reported.

Proposals for change management processes can be found in the CPRE Foundation Level and in many other literature sources: [PoRu2011], [PMI2013], [VanL2009], [WiBe2013].

Due to the wide variety of properties and differences in processes, there is no one unique answer as to which process is most suitable for your project. Above all, the process you select must fit with the processes executed in the company and must be accepted. However, there is no fundamental difference in the basic activities of a change management process.

The trigger for change management is always the receipt of a change request (for further details about the change request, see Section 5.3.1). The main activities of a change management process can be summarized as follows:

- **Step 1: Preparing the change request**
- **Step 2: Formal check of the request:** This checks whether the change request meets the defined input criteria.
- **Step 3: Classification of the change request**: Classifying a change request involves determining whether the change is a corrective, adaptive, or exceptional change. The requirements manager is involved in the evaluation to determine the cause of a change.
- **Step 4: Impact analysis for the change**: The goal of the impact analysis is to estimate and document the consequences of changes. These consequences must be evaluated not only for other requirements, but also for other artifacts (architecture, source code, test cases, training materials). Use the documented traceability information for this evaluation (see Chapter 6). The goal is to determine the required adjustment effort for the changes requested.
- **Step 5: Decision about the implementation of the change request**: The results of the impact analysis are used by the Change Control Board to determine whether to approve or reject the change request. It is not always reasonable to accept and implement a change request. Reasons for a possible rejection of a change request are, for example:
    - The change is too costly and is not justified in relation to the effort required for its implementation or its expected benefit.
    - The desired change contradicts other requirements.
    - Implementation of the change would lead to too high a risk with regard to the stability of the (software) system under consideration.
    - The change is not covered by a contract.

  For reasons of traceability and of achieving agreement among the stakeholders involved, it is essential to document the decisions of the Change Control Board.
- **Step 6: Prioritization of the change requests:** The change requests accepted are prioritized by the Change Control Board (e.g., according to cost and benefit for adaptive changes, or frequency and effect of the error for corrective changes) (see also Chapter 4).
- **Step 7: Scheduling of the change requests for implementation**: Accepted change requests are scheduled for implementation, for example, via a project, release, etc., and are then implemented.

The actual change begins after the change management process. It is implemented either via an ongoing project or a new project. The responsibility for implementation generally lies with Change Management.

For requirements management, at this point it is relevant that the required changes to the requirements artifacts are performed carefully and, after the change, the requirements specification is in a consistent state again.

To perform the change, use the existing traceability to identify all artifacts to be changed. When changing the requirements artifacts, remember that for the changed requirements, you have to:

- Create a new version (e.g., V07, on 08/12/2014)
- Update the status of the new requirement version (e.g., deleted or changed)
- Document the type of change (e.g., corrective change)
- Document the reason for the change (e.g., requirement obsolete due to CR-1287)
- Update the existing traceability relationships

Note: You should also create evidence of what was changed and why, so that any other person can trace why a change was performed and which changes a specific change request led to. Consider the change request as a new artifact and create new traceability relationships between the change request and the changed requirements artifacts.

5.3.1 The Change Request

Change requirements and requirements changes are described by a change request (CR). As part of your requirements management plan, you should define a template for a change request. An example of a template for a change request is shown in Table 6. Depending on the company and the project, however, complete document templates may also be used. Regardless of the form, make sure that the template contains all attributes relevant for the change request. You can use the attributes proposed for a change request in Table 6 as a basis.

Contents	Description
Project name	Designation of the project that the requested change applies to
Request number	Sequential number of change requests within a project
Title	Title of the desired change
Date	Date of the change request
Requester	Name of the requester
Origin	Source or origin of the change (e.g., marketing, management, customer, test)
Functional responsibility	Name or department with functional responsibility for the original functionality
Change type	Type of change request (e.g., defect, innovation, tuning)
Status	Current status of the change request (e.g., evaluated, accepted, rejected)
Requester's priority	Priority of the change from the perspective of the requester
Implementation priority	Priority of the change from the perspective of the change committee
Tester of the change request	Name of the person who tests the execution of the change (including effects)

Contents	Description
Tester of the change request	
Update date	Date of the last update to the change request
Version	Version number of the change document
Release	Assignment of the release for which the change is to be implemented
Specification effort	Forecast specification effort for the change
Implementation effort	Forecast implementation effort for the change
Description of the change	Description of the change(s) to be executed
Comments	Comments about the change request

Table 6: Attributes for a change request (based on [Pohl2010])

5.4 Content for the Requirements Management Plan

In your requirements management plan, document how you want to version requirements and documents in your project. Define the statuses that a requirement may take, how the status transitions are to take place, and who is permitted to change the status of requirements artifacts (see Figure 9). In addition, define the basis for creating a requirements baseline and what the creation of such a baseline means for the subsequent requirements management process—for example, following a requirements baseline, changes are accepted only via a change management process. In the requirements management plan, define how you want to handle changes in the project, how changes are to be documented, whether there is a change committee, who makes up this change committee, etc.

You can use the requirements management plan to explicitly inform all stakeholders about the planned methodological procedure to ensure that the process you have worked out is actually put into practice. A requirements management plan also gives participants who join the project at a later date the opportunity to become acquainted with the organizational and methodological processes.

5.5 Literature for Further Reading

[Eber2012] C. Ebert: Systematisches Requirements Engineering. Dpunkt, 4th edition, 2012 (available in German only).

[Pohl2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, 2010.

[RuSo2009] C. Rupp & die SOPHISTen: Requirements-Engineering und –Management, Hanser, 5th edition, updated and extended, 2009. Chapter 15 (available in German only).

[VanL2009] A. van Lamsweerde: Requirements Engineering – from System Goals to UML Models to Software Specifications. John Wiley and Sons, 2009.

[WiBe2013] K. Wiegers and J. Beatty: Software Requirements, 3rd Edition. Microsoft Press, 2013.

[BSB2008] Christoph Bommer, Markus Spindler, Volkert Barr: Softwarewartung - Grundlagen, Management und Wartungstechniken. Dpunkt.verlag, 2008 (available in German only).

6 Requirements Traceability

"The overall objective of traceability management is to support consistency maintenance in the presence of changes, by ensuring that the impact of changes is easily localizable for change evaluation and propagation." [VanL2009]

6.1 Reasons for Requirements Traceability

As you have already learned in the Foundation Level CPRE [IREB2015], traceability is very important for requirements management. Amongst other things, implementing traceability enables the following:

- Recognition of dependencies between requirements artifacts
- Recognition of dependencies between requirements artifacts and other development and quality assurance artifacts
- Provision of evidence of the implementation and quality assurance of a requirement
- Analysis and performance of required changes as part of change management

Implementing traceability essentially means maintaining references or links to document relationships between different requirements artifacts as well as relationships with predecessor (e.g., business goals) and successor artifacts (e.g., test cases).

Before we continue, let us take a brief look at the different terms used for requirements traceability in the underlying professional literature. Literature contains different terms for "traceability": verifiability, traceability, requirements traceability, etc. In this learning unit, we use the term traceability unless we refer to a specific reference in literature.

6.1.1 What Does Requirements Traceability Mean?

Definition 6-1: Traceability according to the IREB: Traceability is the ability to trace a requirement (1) back to its origin (stakeholders, documents, justifications, etc.), (2) forwards up to the architecture design and code artifacts, as well as (3) to other requirements that this requirement is dependent on.

As the definition above already states, traceability refers to the ability to trace the dependencies between requirements as well as the dependency of requirements on predecessor and successor artifacts. The following definition also explicitly addresses the traceability of a requirements artifact or development artifact over its entire development cycle or lifecycle.

Definition 6-2: Traceability according to [RuSo2009]: "Traceability is the ability to trace connections and dependencies between information that arise during the development, creation, maintenance, and further development of a system at any time."

When we refer to requirements traceability in the following, we are referring to the ability to trace the dependencies between requirements as well as the dependency of requirements on predecessor and successor artifacts over their entire development cycle or lifecycle using documented traceability relationships.

6.1.2 Why Traceability between Requirements and Other Development Artifacts Is Important

Traceability of requirements is not usually a project goal, but rather a means to an end. A number of reasons motivating traceability between artifacts can be found in literature, see [HJD2011], [IREB2015], [WiBe2013], [VanL2009]:

- Demonstrability of how goals and requirements are to be achieved
- Verifiability as to why, if and how a requirement was implemented
- Identification of unnecessary requirements and properties of the system (gold plated solutions)
- Identification of missing artifacts (e.g. missing test cases)
- Simplification of assignment of development efforts to requirements
- Support for reusability of artifacts
- Support for maintenance, admnistration and further development of systems

Requirements traceability helps to answer important questions in the everyday life of a project: for example, what effect changing certain requirements has, the level of implementation effort expected, or how a requirement was implemented or tested.

As the requirements manager, traceability supports you in particular with the following four analyses (see [HJD2011], [PMI2013]):

- **Impact analysis**: analysis of which artifacts are affected by a change (reduction or extension of scope) (see Change Management)
- **Source analysis**: analysis of why a certain artifact (e.g., requirement) exists in order to identify and avoid unnecessary requirements, for example
- **Coverage analysis**: analysis of whether all requirements and subsequent development artifacts have been considered so that the desired product can be completely recorded, developed, and tested
- **Earned value analysis**: analysis to determine work progress (performance value), in order to compare this against the original project plan and, if necessary, take appropriate action (see also Chapter 8)

Furthermore, traceability between requirements and other artifacts (e.g., business processes, legal texts, test cases) is essential to meet certain maturity levels for reference models (e.g., CMMI), standards/guidelines (e.g., ISO 12207), or legal regulations (e.g., SOX).

6.2 Different Traceability Views

[VanL2009] describes traceability as follows: "...*traceability relies on the existence of links between items that we can follow backwards, towards source items, and forwards, towards target items*..."—that is, the ability to navigate between predecessor and successor artifacts.

[GoFi1994] differentiates traceability from the perspective of the requirements specification as follows:

- **Pre-requirements specification traceability** is the traceability of requirements to their origin, for example to the upstream goals and visions or other sources of requirements from the system context, such as a reference to existing business rules or stakeholders.
- **Post-requirements specification traceability** is the traceability of requirements to successor development artifacts such as system architecture components, code fragments, test cases.

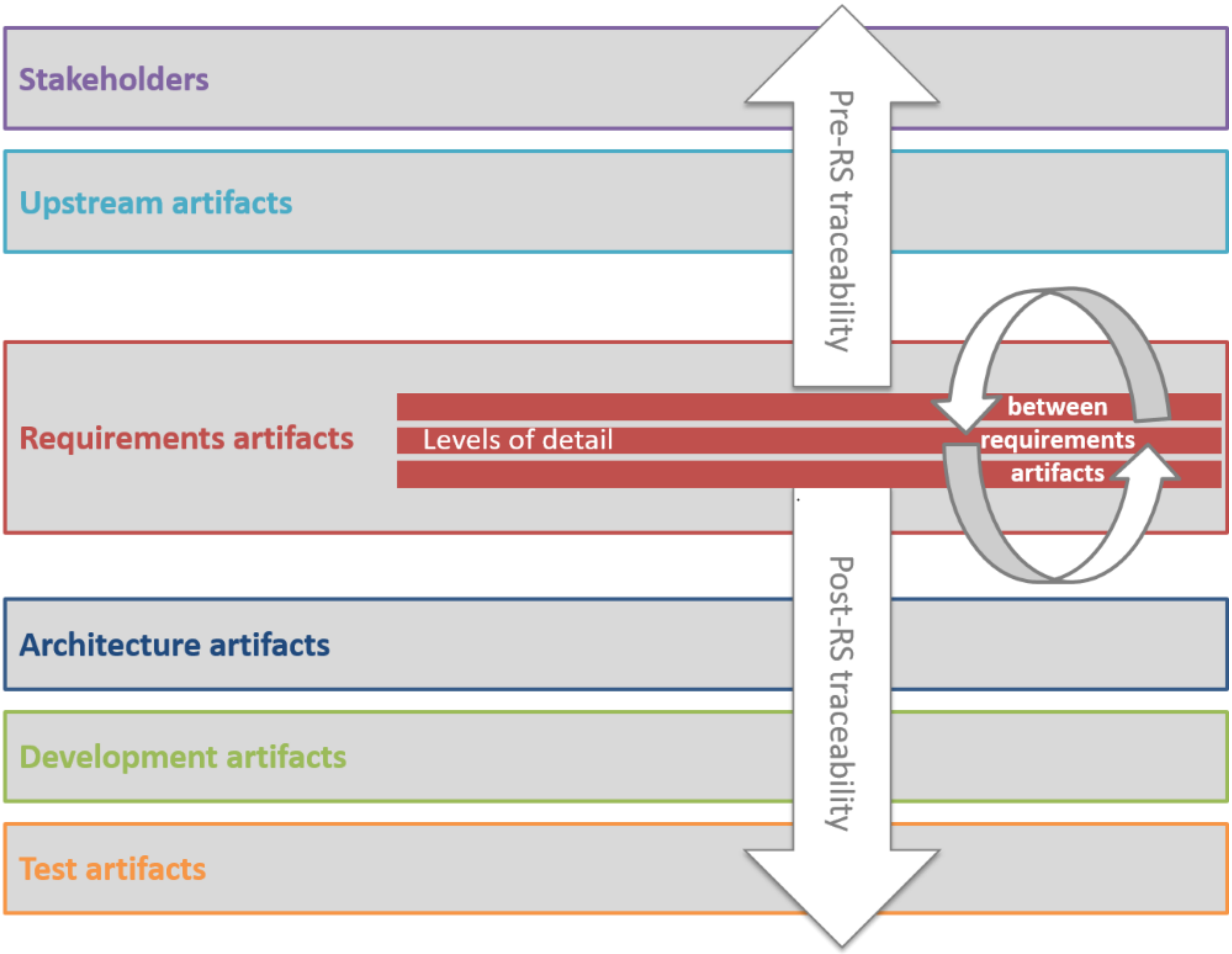


Figure 14: Extended pre- and post-requirements specification traceability

[Pohl2010] shows the extended view of the pre- and post-requirements specification traceability, with the additional traceability between requirements artifacts (e.g., logical dependencies between two functional requirements), referring to this as the **extended pre- and post-requirements specification traceability**.

Figure 14 illustrates the extended pre- and post-requirements specification traceability. The extension focuses on the traceability between the requirements artifacts.

In Section 2.1.3, we discussed that in practice, requirements are closely linked to architecture decisions (twin peaks model). We recommended that to document your requirements in a structured way, you should introduce different levels of detail. In principle, this aspect is also addressed by traceability between requirements, although here there is no explicit differentiation between, for example, logical relationships between requirements at one level, or a detailing at a deeper level. However, we want to explicitly include this differentiation in our examination.

From this point on, in our examination of requirements traceability, we differentiate between the following **dimensions of traceability**:

- **Traceability between requirements at the same level of detail:** This type of traceability describes, for example, content-related dependencies between functional requirements.
- **Traceability between requirements at different levels of detail:** This type of traceability describes, for example, the detailing of legal requirements for system requirements (see Section 2.1.3).
- **Traceability between versions of requirements**: This type describes the traceability of the evolution of a requirement over time. A special feature of this view is that there is only one valid version at a given time.
- **Forwards traceability from requirements to downstream development artifacts**: This type of traceability describes, for example, dependencies that document the implementation/realization of a requirement up to the system component or test case.
- **Backwards traceability between requirements and upstream artifacts:** This type of traceability describes the justification or source of a requirement.

6.3 Relationship Types for Traceability Relationships

"*Traceability links are thus aimed at localizing items, their origin, rationale and impact. To enable item tracing, such links must be made explicit and documented*." [VanL2009]

This excerpt from [VanL2009] states that traceability relationships between artifacts that are dependent on one another must be documented explicitly so that these dependencies can be traced at a later point in time.

[VanL2009] also describes the basic principle of traceability as follows: "*In a production chain, an item is traceable if we can fully figure out where the item comes from, why it comes from there, and where it goes to – that is, what it will be used for and how it will be used*“.

To enable requirements artifacts to be traced back to their origin and to their successor development artifacts, and for the traceability relationships to clearly indicate why this dependency exists, different types of traceability relationships are required.

Of course, traceability relationships could be described in principle by one single relationship type: for example, "dependent_on".
However, in this case, the actual reason for the relationship would not be clear from the use of the documented traceability relationships, which means: does this traceability relationship express that there is a logical dependency between two requirements, does it express that a requirement is detailed by another requirement, or does it even express that two requirements exclude each other because they are different variants? The background to the traceability relationship is missing. This background makes a subsequent use of the documented traceability relationships valuable for impact analyses, for example.

Here, [VanL2009] states: "*The more specialized the dependency, the more specific the reason for it, the more accurate the link, the easier its correct establishment and the more accurate its analysis for multiple uses in traceability management.*" Where traceability relationships are used and defined, however, there is no uniform definition or recommendation for the use of relationship types in literature.

6.3.1 Classes of Relationship Types for Traceability

[Pohl2010] forms five classes of relationship types for documenting traceability and these can be used dependent on the traceability goal:

Condition: The class "**condition**" groups traceability relationships to describe content-related dependencies between two artifacts. This class includes the following relationship types, for example:

- *Limitation*: This relationship expresses that there is a limitation between a source artifact and a target artifact.
- *Precondition*: This relationship expresses that a source artifact is a precondition for a target artifact; that is, one requirement is the precondition for the fulfillment of the other.

Content: The class "**content**" groups traceability relationships that describe content-based comparisons between two artifacts. This class includes the following relationship types, for example:

- *Equality*: This relationship expresses that a source artifact and a target artifact are identical from a content perspective.
- *Contradiction*: This relationship expresses that a source artifact and target artifact contradict one another, which leads to a logical or content-based inconsistency.
- *Conflict*: This relationship expresses that a source artifact is in conflict with a target artifact. However, this conflict does not necessarily lead to a contradiction; it merely hinders the realization of the target artifact.

Documentation: The class "**Documentation**" groups traceability relationships that provide further information about an artifact. This class includes the following relationship types, for example:

- *Example_for*: This relationship expresses that a source artifact represents an example for a target artifact—for example, a scenario for a solution-based requirement.
- *Test_case_for*: This relationship expresses that a source artifact is a test case for a target artifact—for example, a test case for a solution-based requirement.

- *Responsible_for*: This relationship expresses that the person or the role of a source artifact is responsible for a target artifact—for example, the role "Customer support" is responsible for the scenario "Cancel account".
- *Background*: This relationship expresses that a source artifact provides background information for a target artifact—for example, a company guideline for security requirements provides the background for the requirement for customer authentication.

Abstraction: The class "**abstraction**" groups traceability relationships that describe abstraction relationships between two artifacts. This class includes the following relationship types, for example:

- *Classification*: This relationship expresses that a source artifact provides a classification for a target artifact—for example, the scenario "Retrieve account balance" belongs to the class of administrative scenarios.
- *Aggregation*: This relationship expresses that a source artifact provides an aggregation across multiple target artifacts—for example, the scenario "Authenticate customer" is an aggregation of "Customer login" and "Mobile TAN".
- *Generalization*: This relationship expresses that a source artifact provides a generalization for a target artifact—for example, the scenarios "Retrieve postings for the last 30 days" and "Retrieve postings for the period" are grouped under "Retrieve postings".

Evolution: The class "**Evolution**" groups traceability relationships that describe the way in which a requirement is further developed (e.g., fulfilled, refined, replaced, extended). This class includes the following relationship types, for example:

- *Is_the_basis_for*: This relationship expresses that a source artifact has provided a basis for a target artifact—for example, the use of cell phones is the basis for the quality requirement "Use of a mobile TAN procedure".
- *Formalizes*: This relationship expresses that a source artifact provides a formalization for a target artifact—for example, an activity diagram formalizes a textual scenario description.
- *Refines*: This relationship expresses that a source artifact refines a target artifact—for example, a functional requirement "Customer must be authorized with a valid password" is refined by a quality requirement "A valid password must be alphanumeric and must contain 8–20 characters".

Unfortunately, there is no one single answer to the question of which of these or other relationship types that exist in professional literature are actually useful and necessary for your development project. What is important for your requirements management plan, however, (that is, for the planning for your requirements engineering process) is that traceability relationship types are selected and used according to the traceability goal (see Section 6.1).

Tip: Do not be misled into using all possible relationship types from literature. Keep the number of different types as low as possible to achieve your traceability goal. A large number of relationship types may allow the greatest possible flexibility and accuracy, but also requires a much higher effort. What is important is that before you start requirements specification, you define the relationship types to be used in your project.

The traceability dimensions introduced in Section 6.2 can be one means of support when selecting the relevant relationship types for your traceability goal. For example, if you only need to prove how a requirement is implemented and tested, considering **traceability relationships for documenting forwards traceability of requirements artifacts to downstream development artifacts** is sufficient.

6.3.2 Dimensions and Relationship Types

In this section, we use a couple of example assignments to show which relationship types can be used for which traceability dimensions. In the examples, all relationship types are specified from the perspective of the requirements artifact.

Types of traceability relationships for documenting forwards traceability of requirements artifacts to downstream development artifacts.

- Is_tested_by: This type documents that a requirements artifact is verified by a specific test case. This relationship is generally maintained by the test manager who creates the test case.
- Is_realized_by: This type documents that a requirement is realized or reflected by a specific software component or system component. This relationship is generally maintained by the system architect who creates the architecture design artifact.
- Is_implemented_by: This type documents that a requirement is implemented, for example, by a specific function, class, component, etc. This relationship is generally maintained by the developer.

Types of traceability relationships for documenting backwards traceability of requirements artifacts and upstream development artifacts.

- Fulfills: This type documents that a requirement contributes to the fulfillment of an upstream artifact (e.g., a business process). This relationship is generally created by the requirements engineer.
- Excludes: This type documents that a requirement excludes the fulfillment of an upstream artifact (e.g., business goal). This relationship is generally created by the requirements engineer.
- Is_in_conflict_with: This type documents that a requirement is in conflict with an upstream artifact (e.g., a legal requirement). Here, conflict means that the implementation of the system requirement restricts, but does not exclude, the fulfillment of the legal requirement. This relationship is generally created by the requirements engineer.
- Is_explained_by: This type documents that there is additional background information for a requirement that is not contained within the requirement itself. This relationship is generally created by the requirements engineer (e.g., from a user requirement for a statutory requirement for handling SEPA mandates).

Types of traceability relationships for documenting traceability between requirements artifacts at one level of detail.

- *Is_dependent_on:* This type documents that a requirement is dependent on the fulfillment of another requirement from a technical, logical, or content perspective (e.g., relationship between a requirement for the creation of a bank transfer and the requirement for the visual representation of the process).
- *Is_an_example_for:* This type documents that a requirement represents an example for another requirement. This relationship type can be used, for example, to describe a relationship between a solution-based functional requirement or a quality requirement and a descriptive scenario or mock-up.
- *Is_in_conflict_with:* This type documents that two requirements are in conflict with one another and the implementation of one requirement restricts, but does not exclude, the fulfillment of the other requirement. It allows the derivation of limitations that have to be described as part of a project.
- *Contradicts:* This type documents that two requirements are contradictory from a content perspective and therefore exclude each other in a consistent solution. These requirements can actually both be required because they are to be implemented in different products. If both are not required, this relationship indicates contradictions that must be resolved.
- *Is_a_variant_for:* This type documents that a requirement is a variant of another requirement which, for example, is to be evaluated as an alternative solution variant. (**Note**: An alternative is the explicit modeling of variability via feature modeling, see Section 7.3.)

Types of traceability relationships for documenting traceability between requirements artifacts at different levels of detail.

- *Formalizes*: This type documents that a mathematical description formalizes an informal requirement (e.g., textual business rule). This can also be the formalization between a scenario description in prose form and a template-based use case description. Details on modeling requirements can be found in the IREB Certified Professional for Requirements Engineering Advanced Level "Requirements Modeling" [CHQW14].
- Details: This type documents that one or more requirements at a lower level of detail (e.g., system requirement) extends (details) a requirement at a higher level of detail (e.g., user requirement) to the extent that all relevant aspects for implementation have been described.

6.4 Forms of Presentation for Traceability Relationships

To document or implement traceability, traceability relationships are used to document relationships between artifacts (e.g., **requirement** *is tested by* **test case** or **textual requirement** *is formalized by* **requirements model**).

The goal to be achieved with the traceability defines the artifacts between which traceability is to be documented and the types of traceability relationships to be used. Depending on the traceability goal, not every one of the above-mentioned traceability dimensions has to be considered. For example, if traceability is used to ensure that all business requirements in a project have been covered by system requirements, or that a system requirement serves at least one business requirement, then a simple bidirectional traceability relationship of the type *"is implemented by"* between these artifacts may be sufficient. However, if traceability has to be realized according to a specific security standard (e.g., in the aerospace sector), a consistent traceability from the origin of the requirement right up to code artifacts and test artifacts may be required, for example.

6.4.1 Implicit and Explicit Documentation of Traceability

Traceability can be documented implicitly or explicitly.

- **Implicit documentation of traceability**: Implicit traceability can be achieved, for example, through naming conventions, document structures, glossaries, references, etc.
- **Explicit documentation of traceability**: Explicit traceability is achieved through defined and deliberately established traceability relationships between artifacts that are dependent on one another (see Section 6.4.3).

Implicit traceability is understood as the ability to recognize relationships between requirements and to predecessor and successor artifacts via structural or stylistic conventions.

Implicit traceability can be achieved through identical document structures (e.g., in the customer requirements/system requirements specification and test concept). For example, a structuring according to the user-centered functionality across the customer requirements/system requirements specification and test concept will allow you at least to see, across different development phases, how a set of requirements (for a functionality) from the customer requirements specification is implemented and how the quality is assured.

Within a specification, just like inside a book, relationships (and dependencies) to previous and subsequent chapters, definitions, illustrations, etc. can be described.

This means that you can also enable traceability at least at a low granular level within a specification. For example, within a specification, references from user requirements to quality requirements or to requirements for the user interface can be documented.

Furthermore, if identical terms (or process verbs) are defined by means of a glossary and used consistently, in addition to the reference to a chapter, you can also find the corresponding place in the specification that is actually being referenced.

Nevertheless, implicit traceability documentation is not a sufficient approach for enabling requirements traceability in the sense of our understanding (see Section 6.1.1).

Therefore, we see the structuring (and thus the implicit documentation) not as a replacement for documenting traceability, but rather as a supplement to enable traceability within and between different specifications for the reader of these documents.

However, explicit documentation of traceability is also no substitute for a well-structured, legible, and understandable requirements specification. In fact, we would go so far as to say that an understandable structure must never be omitted for the explicit documentation of traceability. Ultimately, a specification is intended to be read and understood by humans! In contrast, traceability is more of a means to an end, for example, to provide evidence of implementation or to analyze the effect of changes (see Section 6.1.2).

6.4.2 Bidirectional and Unidirectional Traceability Relationships

When implementing traceability relationships, we can differentiate between unidirectional (directed) and bidirectional (not directed) relationship types.

- **Unidirectional traceability relationships**: allow traceability from one artifact to another, but not vice versa. For example, the reference from a test requirement to a system requirement allows you to check why the test requirement exists or what it depends on. However, no unique reference from the system requirement to a test requirement will be found. This type of relationship is often found in document-based techniques, where relationships are maintained manually, for example by means of textual references, and refer to either the predecessor or successor artifact. With regard to the documentation direction, it is important to note that reference is made to the artifact to which a dependency exists.
- **Bidirectional traceability relationships**: allow traceability from one artifact to another and vice versa. Unlike the unidirectional relationship, here you can navigate between the artifacts, for example from a requirement to a test case (for example, through a textual reference to a test case) and vice versa, from a test case to the corresponding requirement that is to be checked with this test case. This type of relationship allows you to consider the predecessor and successor artifacts (pre- and post-requirements specification traceability). In requirements management tools, bidirectional relationships are usually created automatically as soon as a traceability relationship is created. The tool thus supports navigation or impact analysis in both directions. For purely textual references, however, explicit maintenance is required for each artifact involved.

Note: In practice, however, and particularly with document-based specifications, we often encounter unidirectional traceability relationships in which, for example, a system requirement refers exclusively to a business requirement, but the business requirement has no reference to the successor artifacts.

6.4.3 Forms of Presentation for Traceability Relationships

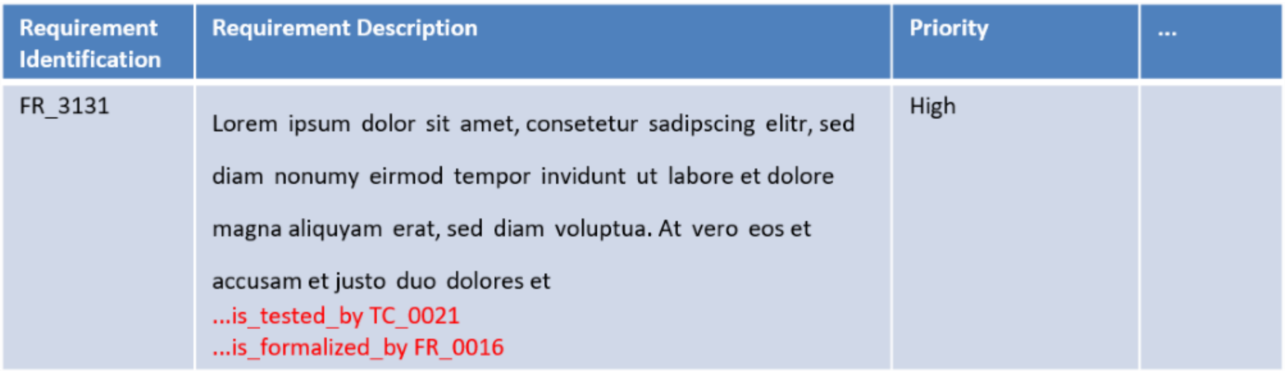
A certain amount of effort must be calculated into the project for documenting traceability (making it usable). This effort is dependent on the traceability goal (forwards/backwards traceability, traceability between requirements artifacts), on the number of relationship types to be considered (see Section 6.3), on the number of requirements in the project, and last but not least, on the form of presentation selected.

There are various forms of presentation for documenting traceability. In this section, we present the most common forms, see [Pohl2010], [RuSo2009], [VanL2009].

6.4.3.1 Textual references

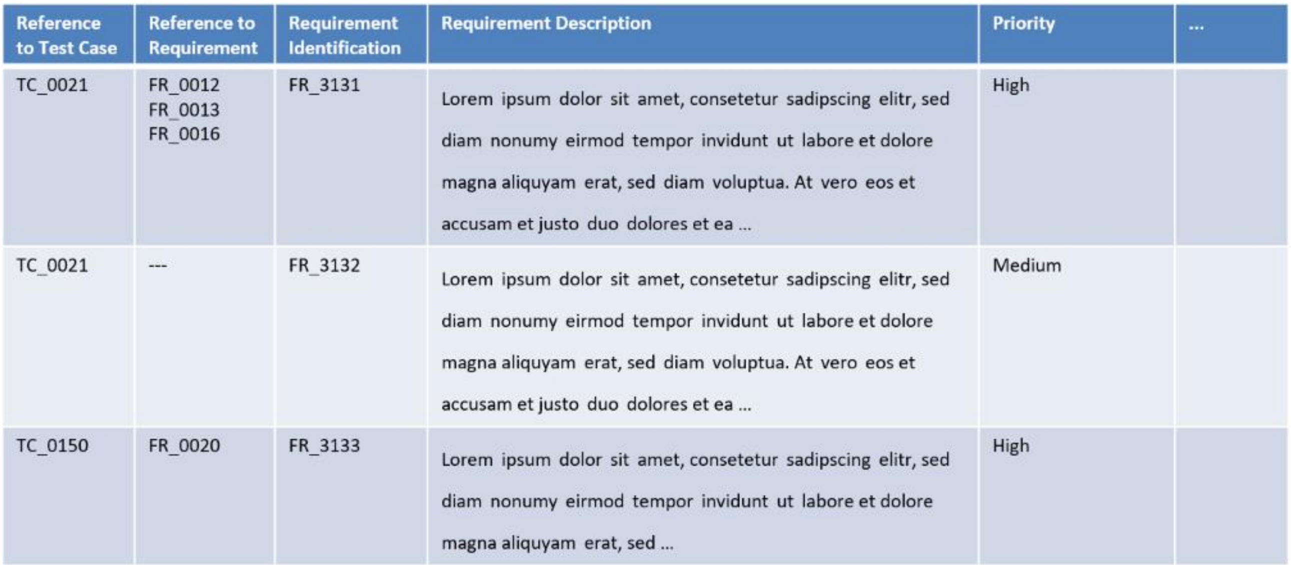
Documenting textual references is the easiest way to implement traceability relationships between artifacts. The relationship describes the relationship type and a unique ID of the artifact to which the relationship refers (e.g., [TC_0021 tests --> FR_3131]). This type of presentation has the decisive advantage that it can be used independently of a requirements management tool and is easy to understand. It is usually documented directly in an artifact, meaning that in a test case, for example, there is a reference to the requirement.

The documentation can be implemented either in the requirement text itself (Figure 15) or using attributes intended for this purpose (e.g., "Reference to Test Case" and "Reference to Requirement"), see Figure 16.



Requirement Identification	Requirement Description	Priority	...
FR_3131	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ...is_tested_by TC_0021 ...is_formalized_by FR_0016	High	

Figure 15: Traceability by means of textual references in the requirement text



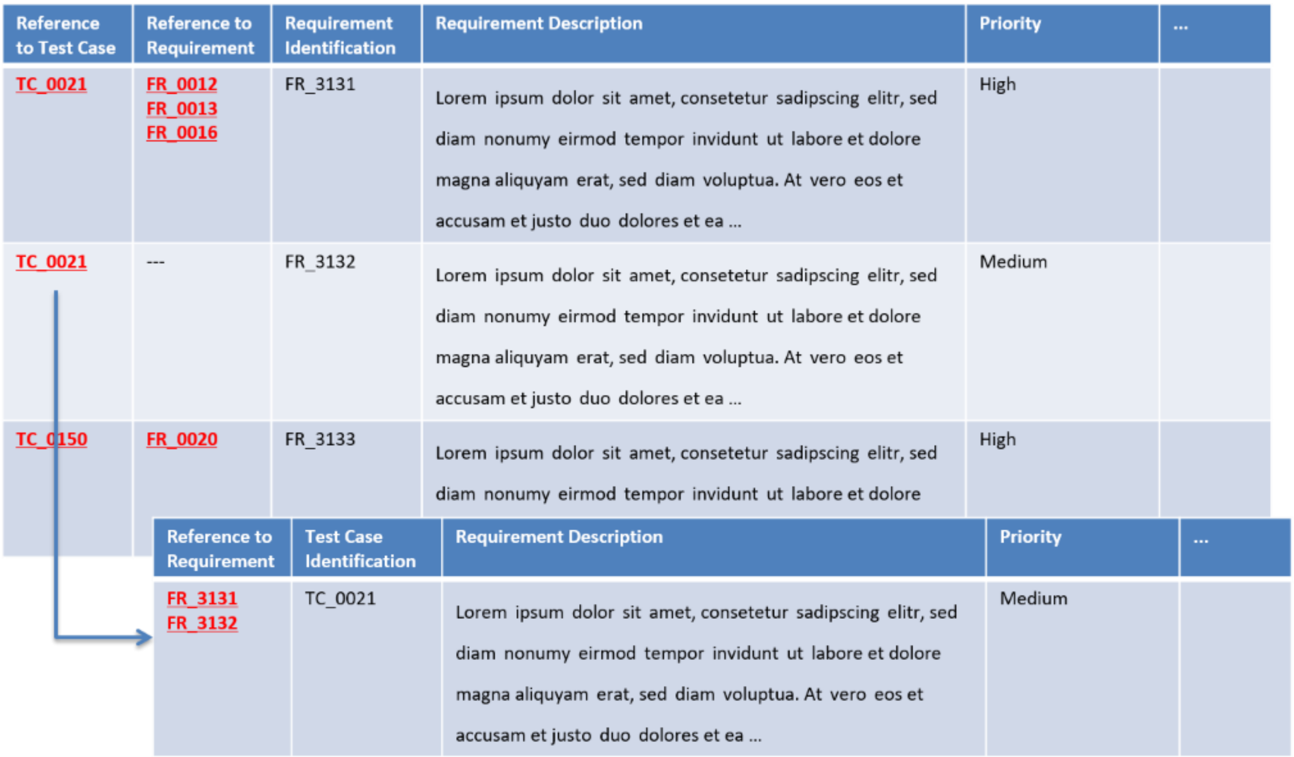
Reference to Test Case	Reference to Requirement	Requirement Identification	Requirement Description	Priority	...
TC_0021	FR_0012 FR_0013 FR_0016	FR_3131	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea ...	High	
TC_0021	---	FR_3132	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea ...	Medium	
TC_0150	FR_0020	FR_3133	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed ...	High	

Figure 16: Traceability by means of textual references with a separate attribute

6.4.3.2 Hyperlinks

Unlike textual references, hyperlinks allow direct navigation to the target artifact. Hyperlinks are created from the source artifact to the target artifact (e.g., from the requirement to a test case). Bidirectional relationships can be created by cross-referencing.

Compared to simple textual references, using hyperlinks has the decisive advantage that you can "jump" directly to the referenced artifacts (see the example in Figure 17). The example shows a hyperlink from the functional requirement FR_3132 to the test case TC_0021 (as forwards traceability from the requirement to the test case). It shows the implementation of a bidirectional traceability relationship: from the requirement to a test case, and from the test case to the original requirements artifact or to the two requirements artifacts (FR_3131 and FR_3132).



Reference to Test Case	Reference to Requirement	Requirement Identification	Requirement Description	Priority	...
TC_0021	FR_0012 FR_0013 FR_0016	FR_3131	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea ...	High	
TC_0021	---	FR_3132	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea ...	Medium	
TC_0150	FR_0020	FR_3133	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore	High	

Reference to Requirement	Test Case Identification	Requirement Description	Priority	...
FR_3131 FR_3132	TC_0021	Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea ...	Medium	

Figure 17: Traceability via hyperlinks

Note: Hyperlinks can generally only be used within one tool or between tools from the same provider.

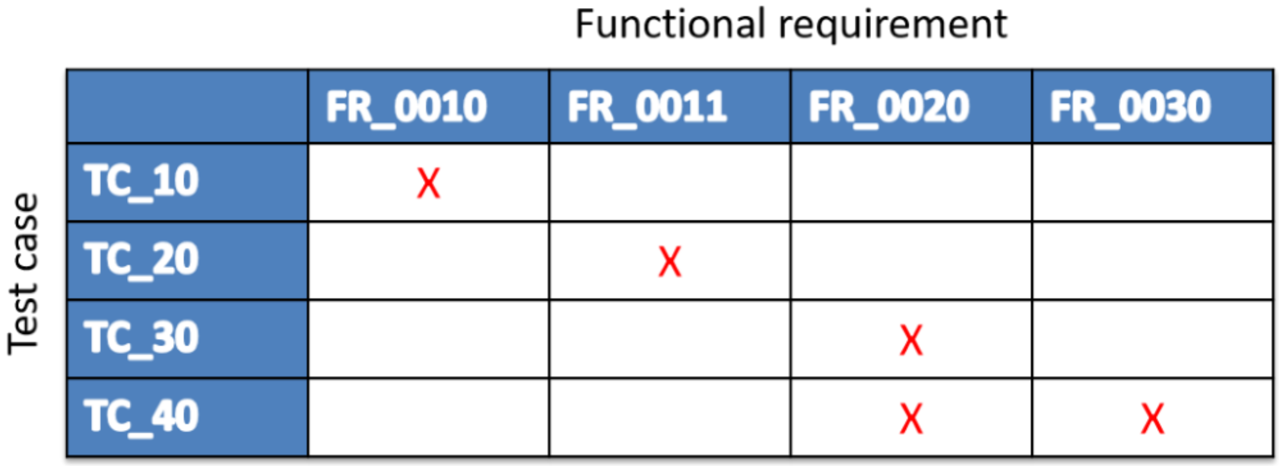
6.4.3.3 Traceability Matrices

Traceability matrices present traceability relationships via references in the cells of a matrix. One source artifact is documented in each row horizontally. Vertically, one target artifact is documented for each column. This means that in the resulting matrix, for each cell, the relationship from the source artifact to the target artifact can be documented. This type of presentation allows an abstract representation of the dependencies between two types of artifacts in a matrix.

Traceability matrices are often used to document precisely one relationship type (e.g., fulfills) between the source and target artifact (see Figure 18).

The traceability relationship is then documented, for example, as a simple "x" in the respective cell. In this example, it is a backwards traceability from the test case (TC) to the requirement (FR), maintained by the person who created the test case.

In the example shown, test case TC_10 tests functional requirement FR_0010; test case TC_20 tests functional requirement FR_0011; test cases TC_30 and TC_40 test functional requirement FR_0020, and test case TC_40 also tests functional requirement FR_0030. Therefore, here we have an N to M relationship.



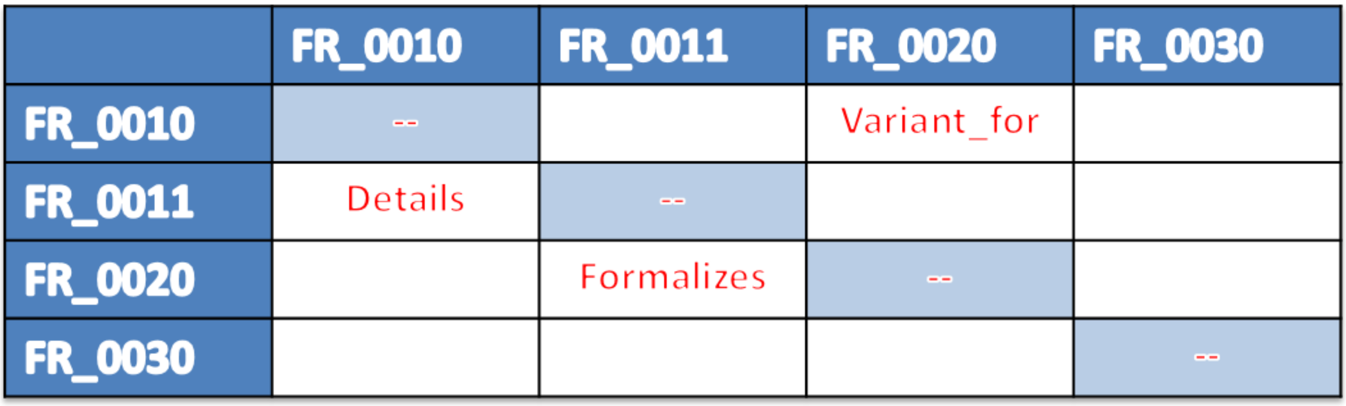
Functional requirement

Test case

	FR_0010	FR_0011	FR_0020	FR_0030
TC_10	X			
TC_20		X		
TC_30			X	
TC_40			X	X

Figure 18: Traceability matrix with one relationship type (FR = functional requirement, TC = test case)

If different relationship types between two artifacts are to be documented (e.g., between requirements at one level of detail), the respective relationship types can also be documented in the cells (see Figure 19).



	FR_0010	FR_0011	FR_0020	FR_0030
FR_0010	--		Variant_for	
FR_0011	Details	--		
FR_0020		Formalizes	--	
FR_0030				--

Figure 19: Traceability matrix with multiple relationship types (FR = functional requirement, TC = test case)

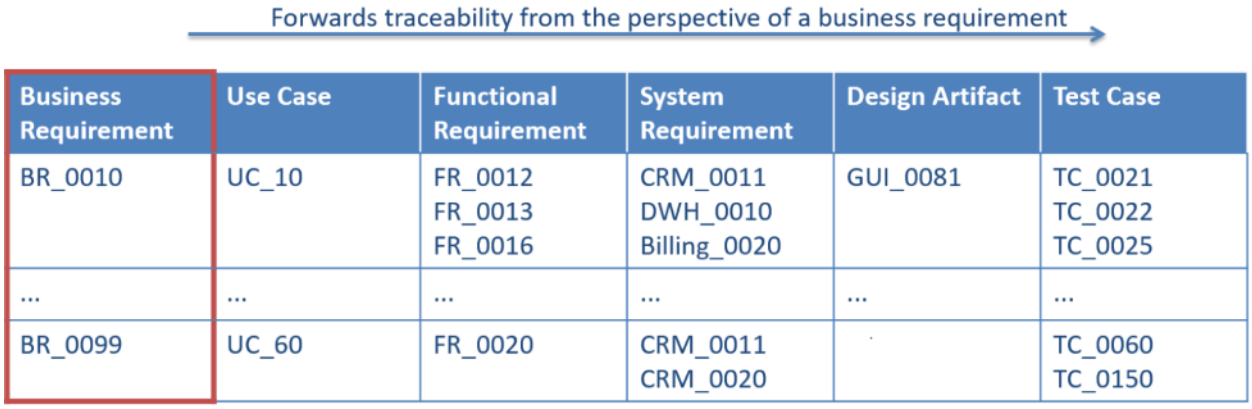
The illustration shows an example for the use of different relationship types in one traceability matrix. The matrix should be read from the row (source artifact) to the column (target artifact): FR_0011 "details" FR_0010; FR_0020 "formalizes" FR_0011; and FR_0010 is a "variant for" FR_0020.

Requirements management tools such as DOORS create traceability matrices automatically based on previously created traceability relationships between artifacts. In practice, however, such matrices quickly become very large and, due to their size, they are difficult to read and maintain.

6.4.3.4 Traceability Tables

Unlike traceability matrices, traceability tables enable you to describe traceability relationships between all artifacts at different levels of detail.

They thus offer a powerful tool for documenting traceability from goals, through use cases and functional requirements, to test cases. Traceability tables can be used independently of a specialized requirements management tool to document traceability between artifacts which are themselves documented in different tools (e.g., Rational Rose, Visual Paradigm, Quality Center, etc.) and Office applications (e.g., Word, Excel).



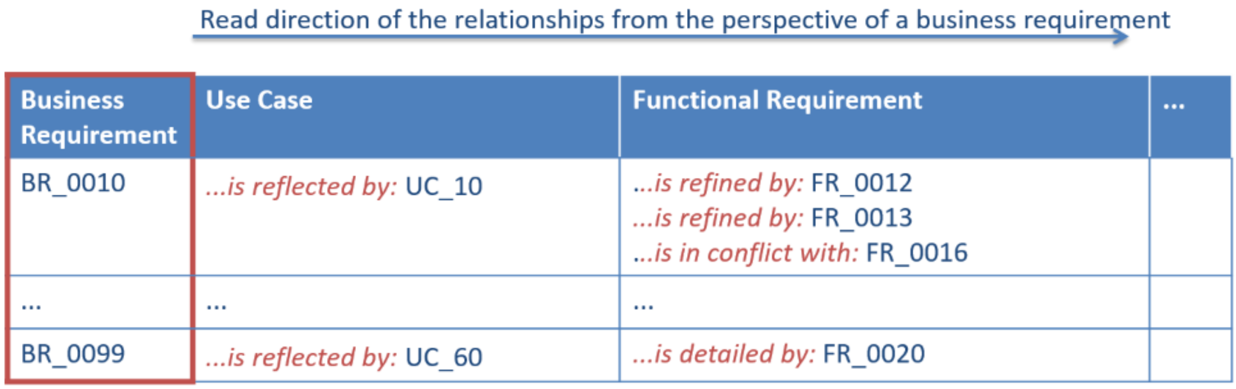
Forwards traceability from the perspective of a business requirement

Business Requirement	Use Case	Functional Requirement	System Requirement	Design Artifact	Test Case
BR_0010	UC_10	FR_0012 FR_0013 FR_0016	CRM_0011 DWH_0010 Billing_0020	GUI_0081	TC_0021 TC_0022 TC_0025
…	…	…	…	…	…
BR_0099	UC_60	FR_0020	CRM_0011 CRM_0020		TC_0060 TC_0150

Figure 20: Traceability table (BR = business requirement, UC = use case, FR = functional requirement, CRM = Customer Relationship Management, DWH = data warehouse, GUI = graphical user interface, TC = test case)

The illustration (**Fehler! Verweisquelle konnte nicht gefunden werden.**) shows which artifacts have a relationship with a business requirement (here, the source). Thus, BR_0010 has a traceability relationship to UC_10; to functional requirements FR_0012, FR_0013, FR_0016; to system requirements CRM_0011, DWH_0010, Billing_0020; to the architecture design artifact GUI_0081; and to test cases TC_0021, TC_0022, TC_0025. What is not recognizable in this example is the underlying relationship type between these artifacts. However, as the traceability relationship always refers to the one source artifact, a corresponding extension to add the relationship type to the respective target artifact would be feasible.

For example, with a supplement for FR_0012, we could describe that business requirement BR_0010 is refined by functional requirement FR_0012: "*is refined by:* FR_0012“. Via this extension, we can even use different relationship types for each source-target relationship (see Figure 21).



Read direction of the relationships from the perspective of a business requirement

Business Requirement	Use Case	Functional Requirement	…
BR_0010	*…is reflected by:* UC_10	*…is refined by:* FR_0012 *…is refined by:* FR_0013 *…is in conflict with:* FR_0016	
…	…	…	
BR_0099	*…is reflected by:* UC_60	*…is detailed by:* FR_0020	

Figure 21: Traceability table with relationship types

6.4.3.5 Traceability Graphs

Another type of presentation for traceability is traceability graphs. In a traceability graph, the nodes represent the relevant artifacts and the edges represent the relationships between the artifacts. To be able to distinguish between the different development artifacts (e.g., scenario, requirement, test case) and relationship types (e.g., refines, implements, tests) at a glance, the recommendation is to define a corresponding form of notation for the nodes and edge types. However, the use of traceability graphs is recommended only if these graphs can be created with a tool automatically based on the artifacts and relationships. In reality, creating and maintaining such graphs manually is too time-consuming. In principle, however, traceability graphs provide an easy-to-understand way of checking dependencies and navigating between the different artifacts. However, similar to traceability matrices and tables, here the actual artifacts are missing—which is why the context of the traceability relationship is lost. The following illustration shows an example of a traceability graph (Figure 22).

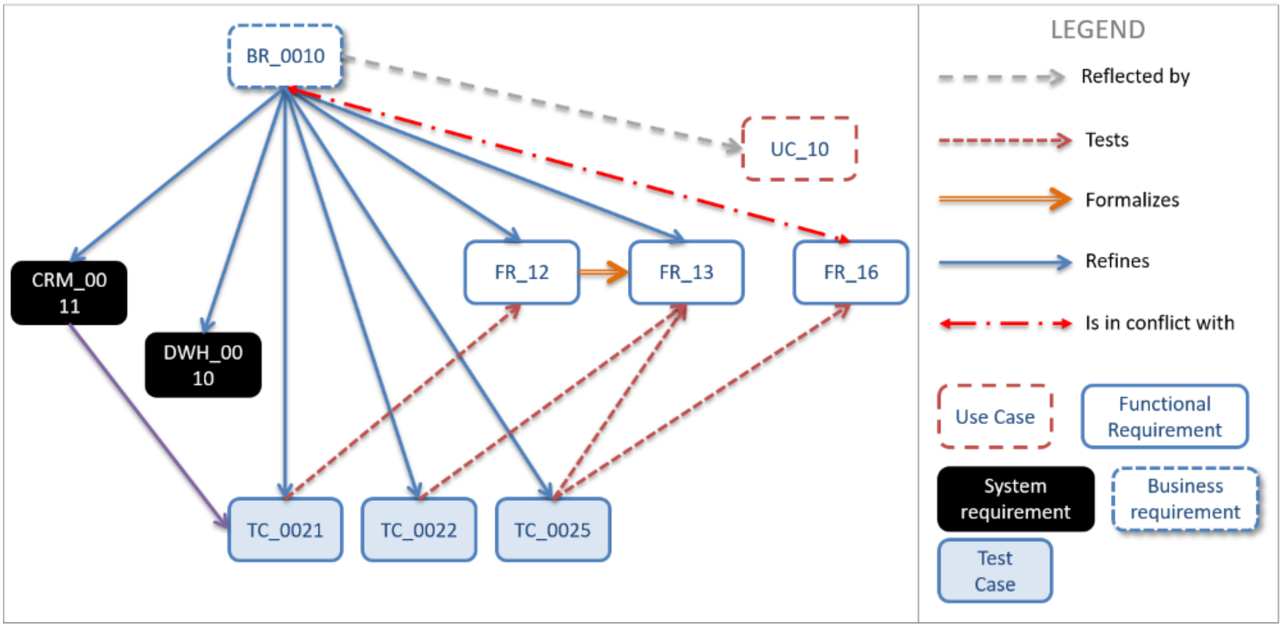


Figure 22: Traceability graph

The illustration shows traceability relationships between different development artifacts as nodes (business requirements, use cases, functional requirements, system requirements, and test cases) and different relationship types as edges (reflected by: tests, formalizes, refines, is in conflict with).

As we can see from the illustration, traceability graphs provide a graphical option for representing relationships between different artifacts. However, if you use these graphs, make sure that you do not select too many artifacts and relationships. With five different artifacts and relationships, this example is already at the limit of traceability. Ultimately, these graphical presentations should be used to identify the artifacts between which dependencies exist. In practice, therefore, the presentation is often reduced to one relationship type.

These dependencies are usually complex enough to avoid bringing additional complexity into the model with a high number of artifacts and relationships. However, if you use tools, you can use filters to display or hide certain artifacts or relationship types so that you are only ever "confronted" with the necessary complexity.

6.4.3.6 Comparison of the Different Forms of Presentation for Traceability

The table below (Table 7) compares the forms of presentation we have discussed thus far and identifies their advantages and disadvantages. Table 7 classifies the different forms of presentation into inline presentation and orthogonal presentation. Inline documentation includes the forms of presentation "textual references" and "hyperlinks", as here, the traceability relationships are directly connected to the requirements specification—they are therefore presented in context. In the case of orthogonal documentation via traceability matrices, traceability tables, and traceability graphs, the knowledge about the relationships is generally presented separately from the requirements specification as these descriptions usually abstract from the artifacts themselves.

Form of Presentation	Positive	Negative	Suitable For
Inline documentation of traceability			
Textual references	Can be implemented independently of tools and comprehensively Relationship is visible in the artifact as plain text	Traceability analyses are very time-consuming	Representing traceability in paper-based textual specifications
Hyperlinks	Relationship is visible in the artifact as plain text Easy navigation between artifacts to detect direct dependencies	Traceability between different tools is not always possible without a lot of effort	Representing traceability in electronic specifications
Orthogonal documentation of traceability			
Traceability matrices	Dependency between two artifacts is visible quickly and easily	Manual creation of traceability matrices is time-consuming and leads to large, only poorly populated matrices	Representing only one single relationship type between two specific artifact types (e.g., use cases and requirements)
Traceability tables	Can be implemented independently of tools Enable clear presentation of the extended pre- & post requirements specification traceability Allow diverse traceability analyses	Highly complex to create	Representing traceability between textual and model-based artifacts in different documents/tools
Traceability graphs	Graphical presentation of traceability allows "abstract" presentation of traceability relationships between artifacts	Can only be used with appropriate tool support	Representing complex traceability between artifacts in a requirements management tool

Table 7: Forms of presentation for traceability relationships

6.5 Developing a Strategy for Project-Specific Traceability

The creation and use of traceability in a project must be planned specifically. It is not usually appropriate to document all possible relationships between all artifacts. Instead, at the beginning of the project, you should think about why traceability is necessary in this project and at what points which kind of traceability will be required to fulfill this goal. To create a traceability strategy, you have to answer the following questions:

- Traceability goal: why or for what purpose must traceability be implemented in this project? This question must be answered by the traceability goal which we have already mentioned several times.
- Usage strategy: what should the documented traceability be used for? This question must be answered by a usage strategy.
- Recording strategy: who is responsible for documenting traceability? This question must be answered by a recording strategy.
- Project-specific traceability model: which are the artifacts between which traceability should be documented, and how should it be documented? This question must be answered by a strategy for documenting traceability.

Note: In addition to answering these questions, you must in particular make sure that all participants know the strategy, that they understand it, and that above all they accept it. Otherwise, regardless of how sophisticated the strategy is, it will disappear without a trace in everyday project life.

6.5.1 The Traceability Goal

The traceability goal should answer the question of why traceability is required or should be established in the respective project. The necessity for traceability can either be due to external reasons (e.g., to fulfill standards) or reasons internal to the project (e.g., to be able to process change requests more quickly and more correctly).

Traceability goals triggered by external factors include:

- Guidelines or development standards specified by the company to fulfill certifications: for example, CMMI (SEI capability maturity model integration) [SEI1999], [SEI2010]; ISO 9000/ISO 9001 [ISO9000], ISO 12207 [ISO12207]
- Legal regulations prescribed by laws or ordinances in certain markets and domains: for example, SOX (Sarbanes-Oxley Act) [USCo2002]
- Guidelines specified by certain domains: for example, IEC DIN EN 61508 [DIN61508], Department of Defense DOD-STD-2167A

Traceability goals triggered by internal (project-driven) factors include:

- To support verifiability to the client: for example, why a requirement was implemented, how a requirement was implemented, the fact that a requirement was implemented
- Quality assurance of specifications through identification of unnecessary requirements in a specification (without a source) or missing test cases

- Support for maintenance, administration, and further development of a system: for example, through identification of the requirements and successor artifacts to be changed

6.5.2 The Usage Strategy

The usage strategy should explain how the traceability information recorded is to be used.

The usage strategy defines how the traceability information documented is to be used by the team. For example, a usage strategy could refer to the impact analysis where traceability relationships are used to determine which requirements and successor artifacts are affected by a change. The usage strategy also defines who is permitted to or should perform analyses of which artifact types and relationship types.

The impact analysis is generally performed by the requirements manager. Based on the requirements artifacts to be changed and the traceability relationships documented, the requirements manager checks which other goals, requirements, architecture design artifacts, test cases, etc. are affected by the change.

In contrast, with a test coverage analysis from the requirements view, the focus is on requirements artifacts and test cases to check whether all requirements are covered by test cases. This analysis can be performed by either the requirements manager or a test manager.

Possible uses of traceability information that is included in a usage strategy are:

- **Impact analysis**: traceability is used to identify the extent of change in requirements and successor artifacts
- **Test coverage analysis**: traceability is used to identify the missing test coverage for requirements
- **Reusability**: traceability is used to identify reusable artifacts
- **Frequency of change**: traceability is used to identify the frequency and the background to changes to requirements
- **Proof of implementation**: traceability is used to prove the implementation of requirements

Note: You generally define the usage strategy based on the goals. Think about what you want to use the traceability information for, who should use it, and which relationship types and artifacts are relevant for this use.

6.5.3 The Recording Strategy

The recording strategy should answer the question of who implements the required traceability relationships and keeps them up to date. It defines the responsibility for documenting traceability relationships. In the recording strategy, for each relationship type between two artifacts, you define who is responsible for maintaining this relationship and when they should do so.

A recording strategy can be, for example, the chronological documentation of traceability relationships proposed by [HJD2011] or [WiBe2013]. The relationship between two artifacts is created as soon as a new artifact is created (e.g., the relationship type *"details"* for a user requirement to a business requirement is maintained by the business analyst as soon as a user requirement is created for a business requirement).

The advantage of this is that there is a clear responsibility for setting traceability relationships and that traceability relationships can be created when an artifact is created.

For example, if a business analyst were responsible for maintaining the traceability relationships between test cases and requirements, they could not do so until the test cases were created. The business analyst would also have to make assumptions about which test cases should be used to test the implementation of which requirements. We would have double the effort here and a corresponding error rate, which is why we also recommend the chronological documentation of traceability relationships. In this case, you "only" have to define which person or role in your project is responsible for maintaining the relationship types you have defined in the project-specific traceability model.

6.5.4 The Project-Specific Traceability Model

The aim of the project-specific traceability model is to answer the question of how (that is, using which form of presentation, see Section 0) and between which artifact types traceability should be documented. Therefore, before you document traceability relationships, before documenting your requirements, you must be clear about the artifact types that you want or have to document traceability between (see Section 6.5.1). You describe these specifications either textually, as an independent information model (for an example, see Figure 23), or as supplementary information in your requirements information model.

A project-specific traceability model describes the permissible relationship types between the relevant requirements artifact types. It also describes how (i.e., with which form of presentation) traceability must be documented (see Section 6.4.3). The creation and use of a project-specific traceability model is described in Section 6.6.

6.6 Creating and Using Project-Specific Traceability Models

The specification of the documentation strategy—that is, the traceability model, with its permissible artifact types and the permissible relationship types and the form of presentation—allows a clear presentation to all project participants of which artifact types and relationship types exist and how they must be maintained (see [Pohl1996], [Pohl2010], [MGP2009], [MJZC2013]). The person(s) responsible for maintaining this information and the point in time at which this must be done are defined by the recording strategy (see Section 6.4.3).

Note: For the actual implementation and use of a project-specific traceability model that has been developed for the project or company, the recommendation is generally to use a requirements management tool that maps the corresponding artifacts, the permissible relationships, and the corresponding stakeholder roles. Of course, all methodological constructs can be implemented with conventional Office applications, but often, these lack the option of analyzing manually set traceability relationships automatically or creating required impact analyses.

6.6.1 Creating a Project-Specific Traceability Model

In a project-specific traceability model, you define which relationship types (e.g., *is_refined_by*; *is_tested_by*) should or may exist between which artifact types (e.g., requirements and test cases).

The following describes a sample process for defining a project-specific traceability model.

- **Selection of a reference schema**: The first step should be to check whether an existing traceability model can be reused and adapted. An effective way to define a project-specific traceability model is to reuse an existing traceability model from a similar project or a company-wide traceability model. This type of traceability model can serve as a basis for defining the project-specific traceability model and will usually already contain a large number of the artifacts and dependencies to be defined.
- **Selection of the artifact types**: In this second step, you define the artifacts between which traceability should be ensured in order to support the goal set in the traceability strategy and the usage scenarios—for example, traceability between use case and functional requirement and between requirement and test case.
- **Definition of permissible relationship types between artifacts**: Here you must define which traceability relationships (see Section 6.3) are allowed between two artifact types—for example, a valid relationship between requirement and test case is "is validated by".
- **Specification of the number of traceability relationships**: Here you define the minimum number of relationships expected between the real artifacts (at instance level of the traceability model)—for example, each requirement requires one traceability relationship to a test case.
- **Definition of the dependency between artifacts**: Here you define which artifact is dependent on another artifact—for example, a test case depends on the content of the requirement. When using unidirectional relationships, pay attention to referencing (see Section 6.4.2)

The example traceability model (Figure 23) presents the different traceability types permissible between the different artifact types. For example, a requirement can detail another requirement or a business goal. A requirement is realized by a design element. A requirement is tested by a test case. In this model, for example, it would not be permissible for a test case to be connected with a business goal via the relationship type *details*.

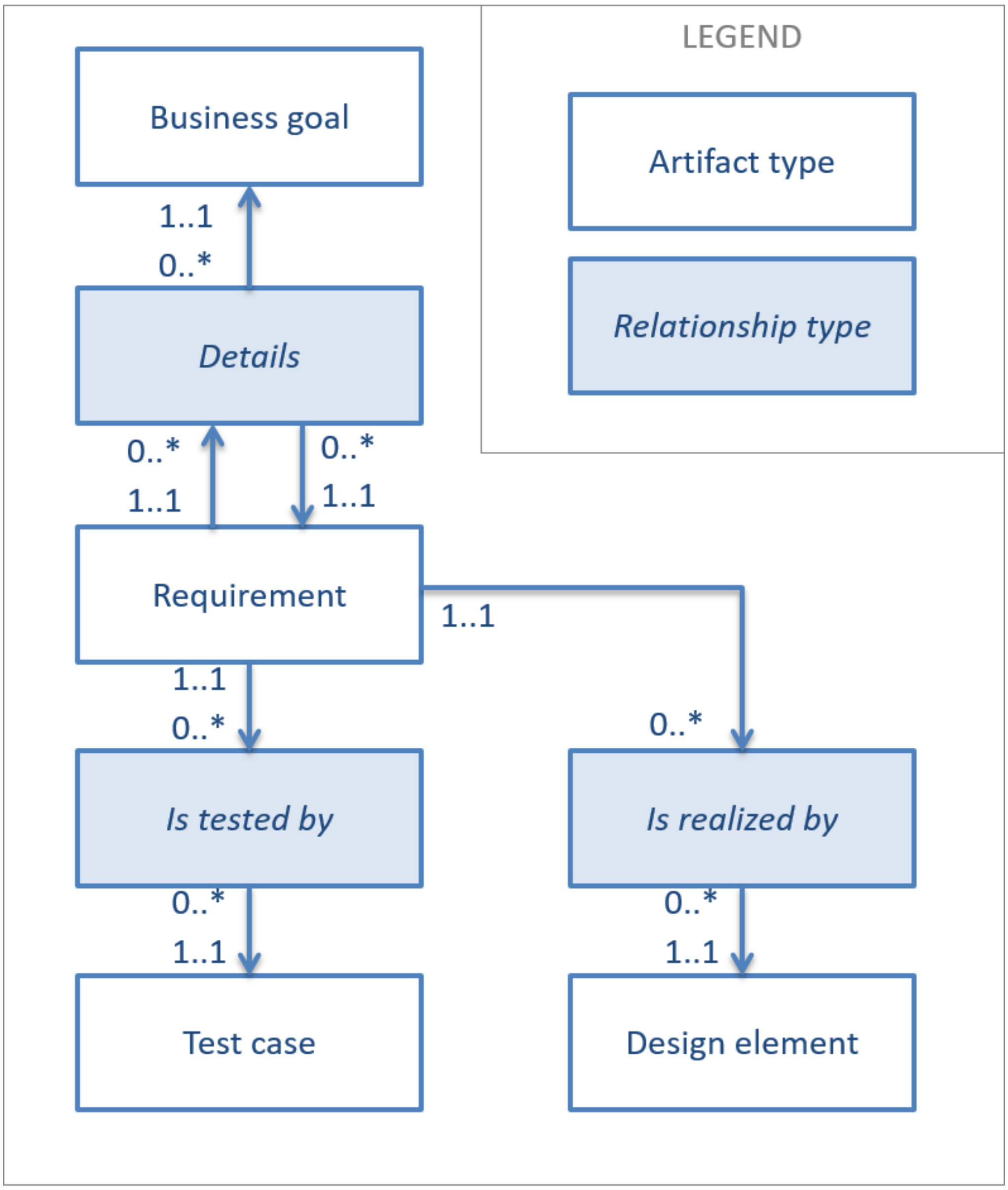


Figure 23: Example of a simple traceability model

With this specification, you can create a clear picture for all participants of the artifact types between which traceability should be realized and what the valid relationship types are. In practice, this model can and will be significantly more detailed and more extensive than the example shown, as there are often more artifact types and relationship types.

Note: To create specific traceability based on a project-specific traceability model, the requirement model elements and relationships in the traceability model (information model) are instantiated and documented according to the artifacts and relationships defined in the traceability model.

The traceability model can either be integrated into the requirements information model (see Chapter 2) or be created as a separate information model. An argument in favor of a separate traceability model is the focus on the relevant artifacts during the creation of a traceability strategy (e.g., when the responsibilities are defined). However, an argument in favor of a joint information model is the central maintenance of, for example, artifact designations, new artifacts, etc. in the event of changes.

6.6.2 Using a Project-Specific Traceability Model

In addition to defining artifacts and traceability relationships, for example in an information model, further aspects have to be considered for the implementation and use of a project-specific traceability model:

Definition of the form of presentation

After defining which relationships between which artifacts should be documented, you must clarify the form of presentation to be used to document traceability relationships. The selection of the form of presentation for traceability relationships is generally influenced by the form of presentation of the artifacts—that is, if the requirements artifacts have been recorded purely in text form, you will probably also document the traceability relationships as textual references or hyperlinks rather than via traceability graphs. (See Forms of Presentation, Section 6.4.3).

Providing support for recording data

Recording traceability relationships between artifacts represents an additional effort, which usually serves other stakeholders (e.g., project managers). Therefore, it is very helpful if the documentation of traceability relationships is supported as far as possible. This can be done on the one hand by requirements management tools, or by self-programmed solutions for example with Word macros.

Creating an alignment between tool artifacts and project artifacts

When using a requirements management tool, a translation into the existing terminology of the tool is usually required. In this step, identifiers of the artifacts and relationship types defined in the model are linked to identifiers offered by the tool and referenced uniquely. For example, if the tool offers only one artifact type "Requirement", but the traceability model distinguishes between "User requirement" and "System requirement", then an appropriate mapping and, if necessary, assignment of an additional attribute is needed here, allowing later differentiation.

Peter Reber is now faced with the challenge of developing a traceability strategy for his project. For this purpose, he has defined the following for himself and his team:

1. Traceability goal

For Peter Reber, the use of traceability is driven by two things: (a) the software development unit should reach the next CMMI level, and (b) Peter would like to have the ability to provide evidence that only requirements requested by management have been implemented.

In the past, there was often unnecessary discussion about the effort involved here, as management had the impression that IT only implements things that nobody needs and that is why everything is so expensive.

2. Usage strategy

Peter wants to use traceability essentially for the following purposes:

1) As evidence that only things directly and indirectly justified by business requirements for the project are developed
2) As evidence that a test case was planned for each requirement
3) To analyze the effects of changes on existing requirements

(1) As evidence that only requirements requested by management in the requirements specification in the project were implemented. This evaluation is to be realized via a dedicated relationship type from the development artifact to the requirement. The evaluation that no development artifacts are created without a dedicated requirement at business level will be created by the developer. This presentation should contain all development artifacts with the associated requirements artifacts. If the evaluation contains development artifacts that cannot be assigned to a requirement at business level, these artifacts must be clarified with the designer and the developer to prevent unnecessary and undesired functionality being implemented.

(2) To check the test coverage, an evaluation of a dedicated relationship type from the requirement to the test artifact is to be used to ensure that all requirements are covered by test cases.

This evaluation will be created by the requirements engineer and should contain all requirements artifacts and the associated test cases. If requirements have no relationship with a test case, the test manager must check these requirements.

(3) To support changes to requirements with a targeted change analysis, three dedicated relationship types are to be introduced: (1) between business and user requirements, (2) between user and system requirements, and (3) between the requirements themselves to document logical and content-based dependencies. This evaluation will be triggered by the requirements manager (Peter himself) when changes occur. The result should present all predecessor and successor artifacts for a selected set of requirements. For each requirements artifact, it should be clearly recognizable which predecessor and which successor artifacts have a relationship with the requirement. This evaluation then helps with the assessment of the impact this change actually has on the predecessor and successor requirements artifacts. This means that it allows you to evaluate the expense (in the sense of person days and costs) created by the change and whether any particular difficulties (e.g., architecture changes) are to be expected.

3. Recording strategy

Relationship types between development artifacts and requirements artifacts (system requirements) are to be maintained by the respective developer as soon as a functionality is implemented for a requirement.

Relationship types between test cases and requirements artifacts must be maintained by the test manager as soon as a test case is created for a requirement. Traceability must be documented as a bidirectional relationship, which is why the test manager requires restricted write access to the requirements to record the forwards relationship to the test case.

Relationships between the different requirements artifacts must be maintained by the requirements engineer and business analysts as soon as a new requirement is created that (a) represents a detailing of or (b) has a logical or content-based dependency to an existing requirement and influences this existing requirement in some way.

4. Creation of a traceability model (documentation strategy)

To implement traceability across different documentation tools and documents, textual references with attributes intended for that purpose (see Figure 16) and traceability matrices are to be used.

Peter Reber has documented the traceability relationships to be maintained between artifacts in the following traceability model.

In the project, there are three levels (classes) of requirements: business requirements, user requirements, and system requirements.

Business requirements can be detailed either by user requirements or directly by system requirements. User requirements are always detailed by at least one system requirement. Requirements themselves (see the abstract class "Requirement") can be in a relationship with one another via an "*influences*" relationship type if they are dependent on each other logically or from a content perspective. Further detailing of the content is not currently planned. Each requirements artifact will be tested by a test case and ultimately, every system requirement will be implemented by a development artifact. There must be NO development artifacts that do not realize a system requirement. However, there may be test cases that have not been assigned to new requirements, that is, they are not used directly to check this requirement—these are regression test cases, for example.

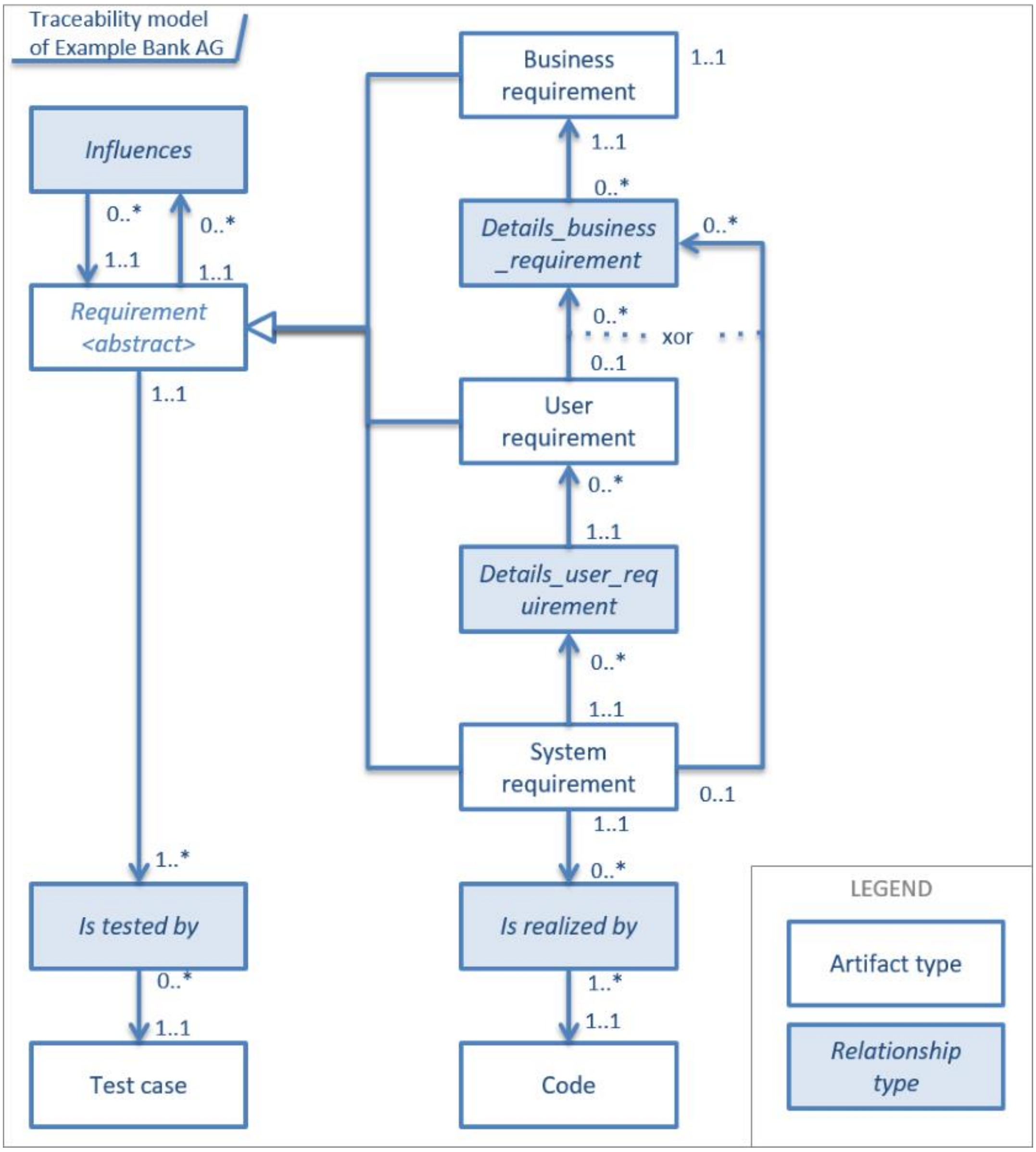


Figure 24: Traceability model for our example Bank AG

6.7 Measures for Evaluating Implemented Traceability

In the previous section, we looked at how you can document traceability and how you can set up a traceability strategy for your project. However, the traceability strategy that you introduce must also be put into practice and must not exist merely on paper. Therefore, during requirements management, at some point the question will be raised as to whether the traceability strategy set up is being or has been followed, and how completely traceability relationships between the artifacts have actually been documented.

For this purpose, perform a check to ensure, on the one hand, the quality of the current documentation with regard to traceability, and on the other hand, to identify problems in the traceability strategy. Checking traceability information provides an insight into the quality of the current documentation.

The following measures can support you in checking the completeness and quality of the traceability relationships:

- Ratio of the number of correct traceability relationships (e.g., has the correct relationship type been used, does the referenced artifact still exist) to the total number of traceability relationships (correctness)
- Ratio of the number of existing traceability relationships to the total number of traceability relationships required (completeness)
- Ratio of the number of requirements with traceability relationships to the total number of requirements (density)
- Ratio of the number of test cases with traceability relationships to requirements to the total number of test cases (backwards traceability, test case to requirement)
- Ratio of the number of requirements with traceability relationships to a test case to the total number of requirements
- Ratio of the number of documents with correct references to the total number of documents (e.g., does the document exist in the specified directory)

Note: Note that the checks for correctness in particular cannot be fully automated. Content checks in particular require a human inspection. Automated checks can be used, for example, to check the existence of artifacts or documents. Furthermore, restricted statements about the correctness of relationship types would be possible if the check of the relationships used were based on the traceability model created and, for example, a relationship type is detected between two requirements that may only be set between test cases and requirements.

A low number of traceability relationships compared to the number of artifacts suggests that the relationships have not been maintained consistently and completely. On the other hand, a low number of correct relationships in relation to the total number of relationships suggests that either relationships were negligently maintained, or that changes were not consistently applied to all the artifacts concerned.

Any deviation may have different reasons that need to be discussed. For example, create a threshold value for each dimension that you want to achieve. If this threshold value is not met, you should check why.

Furthermore, based on your usage strategy, check whether suitable results are achieved. If you do not get the results you want, this can be for at least two reasons: (1) the recording and documentation strategy was not followed satisfactorily, or (2) the documentation strategy was not extensive enough to fulfill your usage strategy.

Note: Follow up and check whether your traceability strategy is actually being put into practice or whether it was just an ideological definition. If you find out that the traceability strategy is not being followed, or is not being followed satisfactorily, find out why and try to remove the obstacles (too complicated, not understood, too time-consuming, no tool support, etc.).

Possible reasons for missing or incorrect documentation of traceability are:

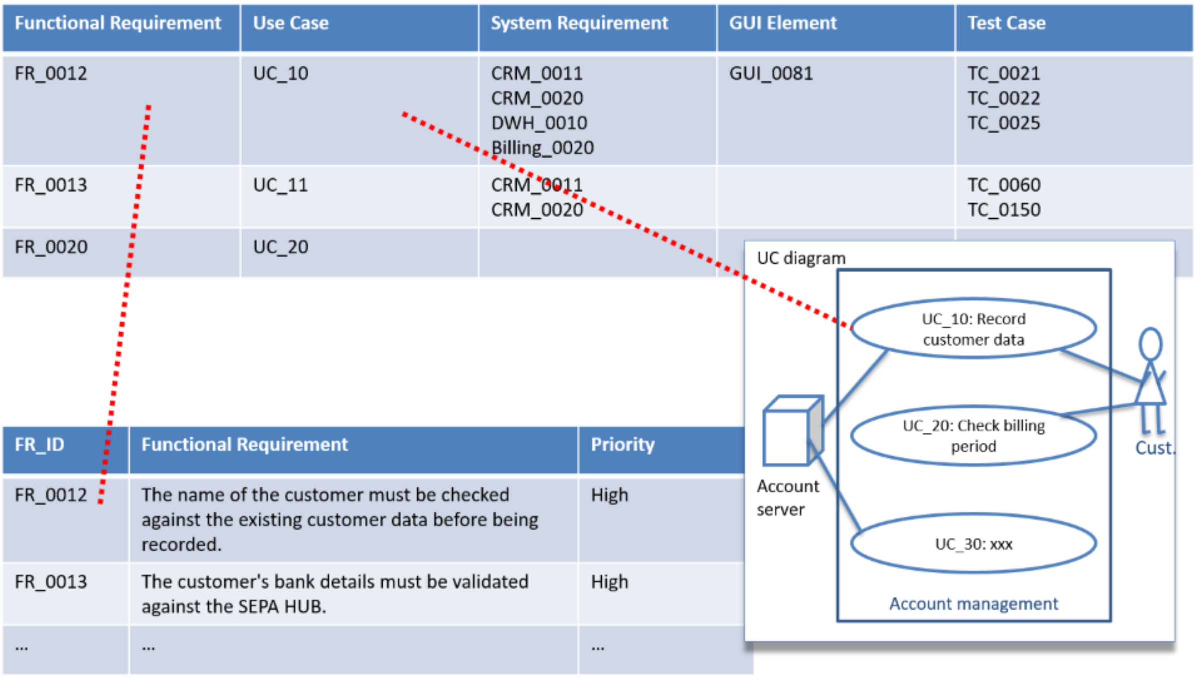
- The necessity of documenting traceability is not known within the team (the benefits may not have been understood)

- Missing traceability strategy within the team or the traceability strategy has not been understood
- Time constraints in the project do not allow documentation of traceability
- There is no agreed and accepted traceability model within the project team
- Insufficient tool support for recording traceability relationships

6.8 Challenges for Traceability between Textual and Model-Based Artifacts

Traceability between textual artifacts (e.g., functional requirements) and model-based artifacts (e.g., activities in UML activity diagrams), or between model-based artifacts themselves can only be achieved with high effort and is therefore not put into practice frequently in real life.

The reasons are generally a lack of integration between model-based and text-based requirements engineering and requirements management tools, as well as the missing unique (at least visible) reference for model elements (e.g., link from a textual requirement to a class in a UML class diagram). Of course, this class has a unique identifier somewhere within the tool or in the properties, but it is difficult for a user to find this. Even though today's tools do not offer complete, high-performance support for linking model artifacts with textual requirements artifacts, there are options for establishing traceability across these different artifacts. Possible solutions include either using separate labels in the identifiers here or creating unique textual identifiers via glossaries that can be referenced. Figure 25 shows an example for a tool-independent implementation of traceability between a use case model and textual requirements.

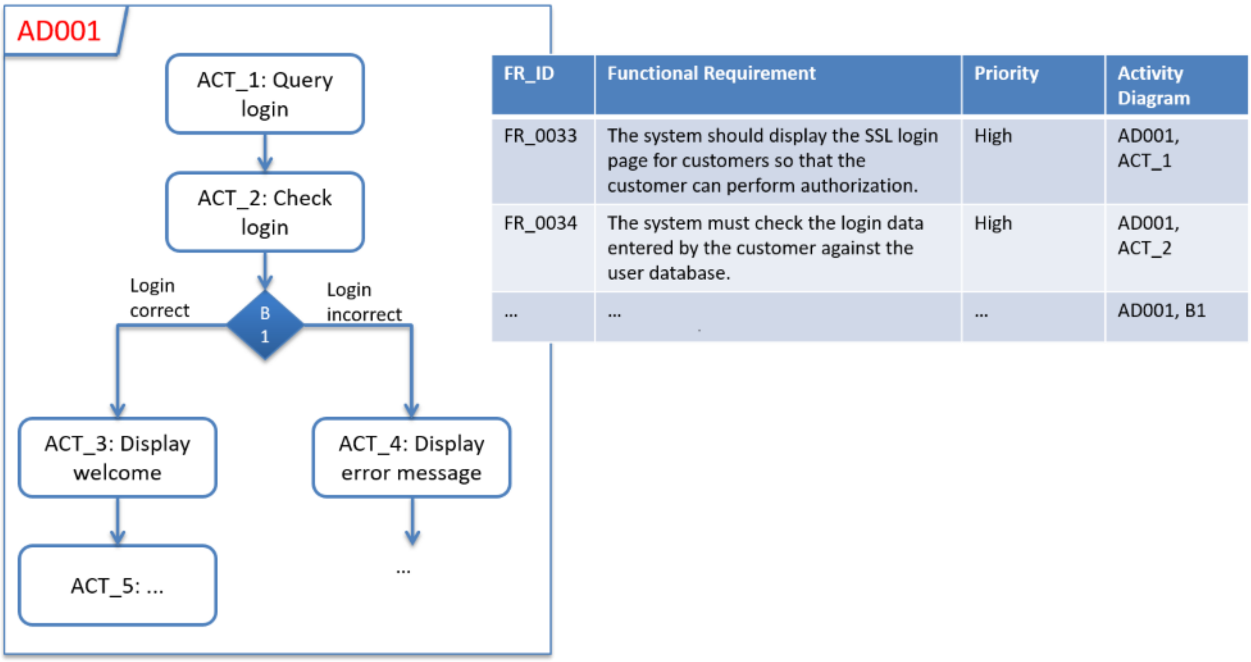


Functional Requirement	Use Case	System Requirement	GUI Element	Test Case
FR_0012	UC_10	CRM_0011 CRM_0020 DWH_0010 Billing_0020	GUI_0081	TC_0021 TC_0022 TC_0025
FR_0013	UC_11	CRM_0011 CRM_0020		TC_0060 TC_0150
FR_0020	UC_20			

FR_ID	Functional Requirement	Priority
FR_0012	The name of the customer must be checked against the existing customer data before being recorded.	High
FR_0013	The customer's bank details must be validated against the SEPA HUB.	High
...	...	...

Figure 25: Traceability between textual and model-based artifacts

Figure 25 shows an example of a traceability table. Here you can see that there is a traceability relationship between functional requirement FR_0012 and use case UC_10. In principle, the relationship type could be added to the traceability table (see Section 6.4.3.4).



FR_ID	Functional Requirement	Priority	Activity Diagram
FR_0033	The system should display the SSL login page for customers so that the customer can perform authorization.	High	AD001, ACT_1
FR_0034	The system must check the login data entered by the customer against the user database.	High	AD001, ACT_2
...	...	...	AD001, B1

Figure 26: Traceability between textual and model-based artifacts

Figure 26 above shows a further example for traceability between textual and model-based artifacts. In this example, there is a traceability relationship from textual requirements to activities in an activity diagram. In activity diagrams in particular, this type of traceability can be used for a better description of the individual activities and conditions. For this purpose, in the example, every activity was described with an identifier (e.g., ACT_00xx) in front of the actual name. Here, the textual requirement references (as a unidirectional relationship) to the activity diagram and the corresponding activity via a textual reference. Bidirectional relationships can also be represented, but this generally makes such models more difficult to read, which means that you have to weigh up what is more important—bidirectional traceability or the legibility of the models.

Practical tip: Some modeling tools support the realization of traceability between models and textual artifacts via word patterns or glossaries.

6.9 Content for the Requirements Management Plan

Document the traceability strategy you define, including the traceability model, in your requirements management plan (see the case study in Section 6.6). At this point, it is less important how (i.e., in which form) you integrate the things into your requirements management plan, and more important that you document your thoughts and definitions as to how you want to record, present, and use traceability in your project in your requirements management plan. This is the only way to discuss and agree these concepts with all stakeholders involved before the project starts.

Furthermore, the explicit documentation of your traceability strategy in a requirements management plan means that participants who join the project at a later stage can quickly familiarize themselves with the project and read the organizational and methodological specifications.

6.10 Literature for Further Reading

[GoFi1994] O.C.Z. Gotel and A.C.W Finkelstein: An Analysis of the Requirements Traceability Problem. In Proceedings of IEEE International Conference on Requirements Engineering, 1994.

[HJD2011] E. Hull, K. Jackson, and J. Dick: Requirements Engineering. Springer, 3rd Ed, 2011.

[MGP2009] P. Mäder, O. Gotel, and I. Philippow: Getting Back to Basics: Promoting the Use of a Traceability Information Model in Practice. In: Proceedings of 5th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE2009), Vancouver, Canada, May 2009.

[MJZC2013] P. Mäder, P.L. Jones, Y. Zhang, and J. Cleland-Huang: Strategic Traceability for Safety-Critical Projects. In: IEEE Software, Volume 30, Issue 3, May/June 2013.

[Pohl2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, 2010.

[VanL2009] A. van Lamsweerde: Requirements Engineering – from System Goals to UML Models to Software Specifications. John Wiley and Sons, 2009.

7 Variant Management for Requirements

Before we look at variant management in the context of requirements management and describe how you document variability in requirements, we will explain a couple of terms from product line development.

We must first distinguish between the terms "product family" and "product line" to strengthen the understanding for product lines.

Definition 7-1: Product family: A product family is a set of connected products that complement each other and cover the requirements of a common application area (e.g., Office suites). These products are generally designed to supplement one another, see [Gabl2014a].

Definition 7-2: Product line: A product line groups different variants of a product. The different products can generally be substituted for one another and differ, for example, in the scope of functions and price (e.g., Apple iPhones). The products in a product line are generally defined such that each of the products meets specific customer wishes, see [Gabl2014b].

A **product line** therefore encompasses a set of specific, differentiated products that all share a common basis (referred to as commonalities). In addition to these commonalities, a product line has a defined variable part that enables different products to be created (referred to as the variability of the product line). Thus, different products can be created through the defined commonalities and the variability of the product line. A product line can encompass hardware and software parts that have been defined as commonalities or variability and can be used in different products.

A requirements pool is a set of requirements that contains more than the set of requirements for a specific product. It can also contain requirements that are not currently considered in any product.

Product line development differentiates between two different processes:

- *Domain engineering*: In domain engineering, the commonalities and variability of existing product variants are identified and used to create a model of the product line.
- *Application engineering*: Here, the product line model is adapted on a product-specific basis, thereby creating product variants.

Definition 7-3 of software product lines (from [ClNo2007]): *"A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way."*

Variability is a term frequently encountered in the context of product lines (see [PBL2005], [Pohl2010]). It enables the specification (and therefore the implementation) of different products through the definition of variation points and variants, without necessitating the creation of a separate specification for every product.

Example of commonalities and variability of a product line: The Apple iPad can be understood as a product line. The commonalities of iPads include the housing, the displays, the processors. Specific product variants (e.g., iPad, 64 GB, black, with Wi-Fi and 4G) are created through the **variation points** "different colors", "different memory sizes", etc. The variability is described via variation points and variants.

Variation points are points (e.g., in the specification) that allow or require the selection of specific variants.

Example of variation points: The variation points "iPad memory" and "iPad color" are variation points that are made more specific by different variants.

Definition 7-4: Variation point: A variation point describes where—at what point—within a product line the requirements vary.

Variants are specific forms of artifacts (e.g., requirements or properties of the product) with reference to a variation point.

Example of variants (of a variation point): The variation point "iPhone memory" has the following variants: 8 GB, 16 GB, 32 GB, 64 GB.

Definition 7-5: Variant: Variants describe two or more possible (permissible) forms of the requirements at a variation point (e.g., 7-inch, 10-inch, 12-inch display).

When we refer to variability below, we are always referring to the differences between different products—that is, the variants that are valid simultaneously in a product line (from which different products can be derived). The changing of requirements artifacts over time is not variability, but rather versioning (see Chapter 5).

Product line development generally differentiates between domain development and application development (see [PBL2005]). Domain development creates the reusable artifacts as commonalities and variability of the product line. Application development creates individual products based on reusable artifacts.

In this case, every product contains all of the commonalities and a selection of variants. To allow executable and consistent products to be derived within the scope of reuse, corresponding selection and combination rules must be considered specially for selecting variants. For this purpose, corresponding rules (dependencies) are defined in domain development for the ability to combine and derive specific products (see Section 7.1).

Even if it is not your intention to operate product line development, the use of variability can be an interesting option for you for the following reasons:

- To allow you to describe different variants for a requirement—which meet the client's goal to different levels of quality—in your requirements specification. At the requirements elicitation stage, the client is often still unsure whether they want solution A (e.g., navigation with voice guidance) or solution B (e.g., navigation with voice and image guidance). The client often wants to make their decision dependent on the expense or the implementation time. Therefore, even in standard product development, you sometimes have to specify different variants.
- To allow you to describe optional requirements within your requirements specification, whereby these optional requirements could be considered as additional requirements and should be evaluated before realization. The reasons for such optional requirements are often analog to the reasons currently listed, that is, uncertainty in the mind of the client about what they actually want.
- To enable you to document different installation and configuration options for an application in a targeted way using variation points and variants. In this case, we are not talking about a product line, but rather about variability in the sense of configuration options.
- To enable a targeted reuse of requirements in similar projects.
- To allow you to develop similar product variants that can become a specific product variant either before implementation or on delivery or licensing.

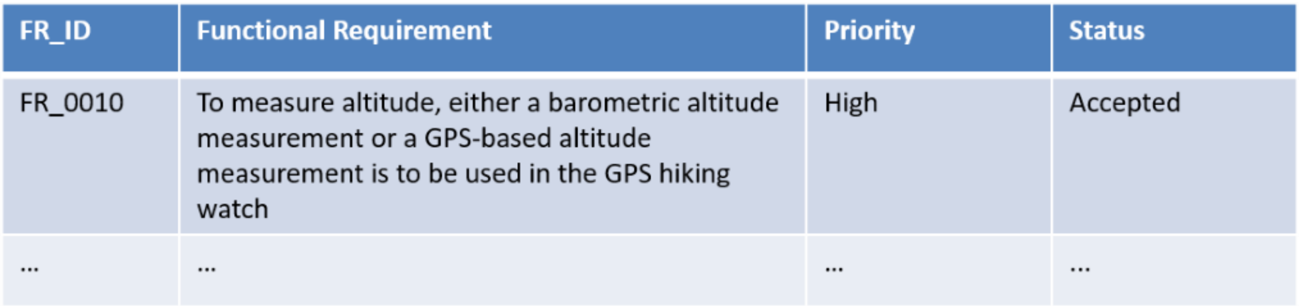
Practical tip: In reality, we encounter this necessity to document variants as alternatives and options as soon as a stakeholder cannot decide what they want specifically, and they want to make their decision dependent on effort, for example. In this situation, via the mechanism of variability, to estimate the effort you can signal to the subsequent development phases that artifacts shown as variants must be considered separately.

7.1 Using Variants of Requirements

As already explained, variants always refer to variation points. A requirements document generally contains a range of variation points and variants, even if we are not in product line development and variability was not explicitly documented.

Variability can be documented implicitly or explicitly. In the case of implicit documentation, it must be clear from the formulation of the requirement that different product variants are possible.

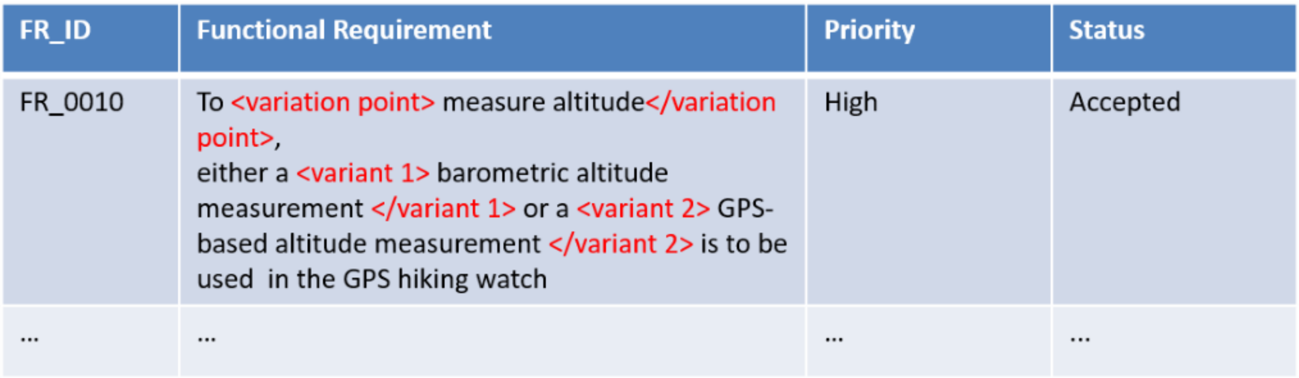
In **implicit** documentation, the word "or", for example, indicates that different product variants are possible (see Figure 27). However, the word "or" is not a reliable indicator of a variation point, as it is also frequently used in logical conditions. Other key terms such as "both ... and" are also generally not clear enough.



FR_ID	Functional Requirement	Priority	Status
FR_0010	To measure altitude, either a barometric altitude measurement or a GPS-based altitude measurement is to be used in the GPS hiking watch	High	Accepted
...	...	...	...

Figure 27: Example of an implicit documentation of variability

In the case of **explicit** documentation of variability, variation points and variants are either integrated into the requirements specification or are created orthogonal to the requirements artifacts (i.e., in a separate model). In the case of textual requirements, both the variation points and the possible variants are explicitly shown in the requirement text in an integrated documentation (Figure 28).



FR_ID	Functional Requirement	Priority	Status
FR_0010	To <variation point> measure altitude</variation point>, either a <variant 1> barometric altitude measurement </variant 1> or a <variant 2> GPS-based altitude measurement </variant 2> is to be used in the GPS hiking watch	High	Accepted
...	...	...	...

Figure 28: Example of an integrated explicit documentation of variability

For explicit documentation of variability in an orthogonal model, the following notation can be used, for example (see Figure 29, [PBL2005], or [Pohl2010]).

In the case of orthogonal documentation, the textual requirement remains untouched. The variation points and variants are documented in a separate model, and the variation points and variants are set in a relationship to the associated requirements artifacts—via a traceability relationship, for example (see Chapter 6).

For the subsequent derivation of specific products as part of application development, when documenting variability, you must take into account that not all variants can be combined freely with one another. There are clear rules about which variants can be or must be combined at a variation point, and which variants may or must be combined across variation points or not.

Here, for example, the orthogonal model (Figure 29) indicates which rules have to be observed when selecting the variants "Barometric altitude measurement" and "GPS-based altitude measurement". In this example, only one of the two variants may be selected for a specific product.

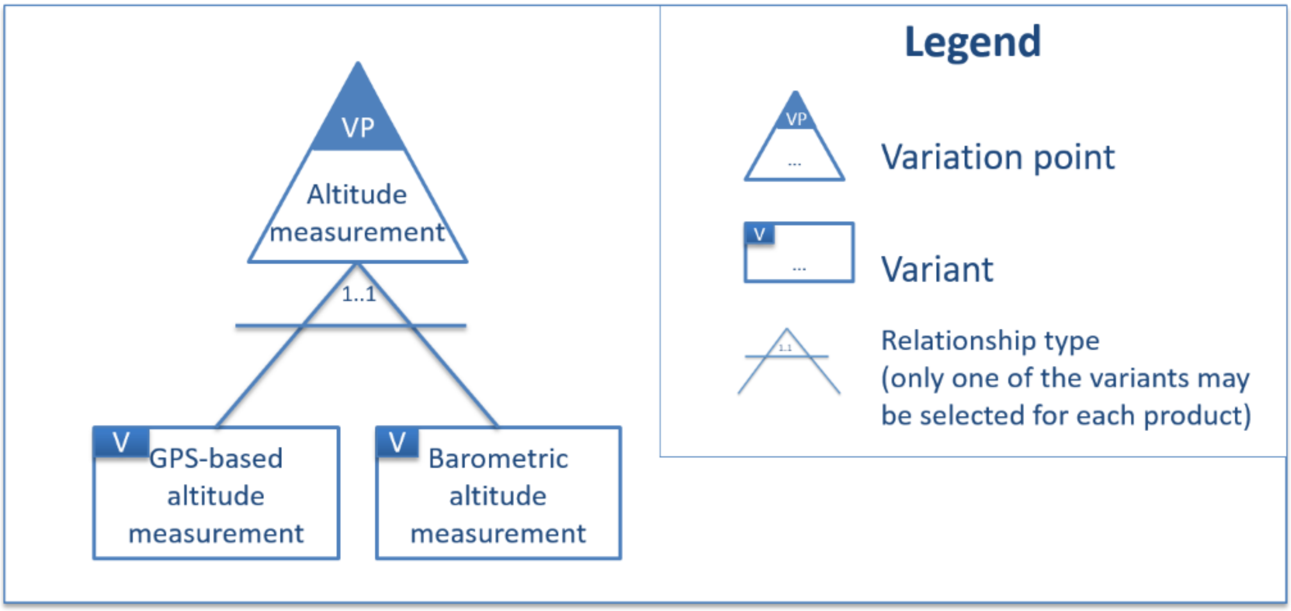

VP
Altitude measurement
1..1
V
GPS-based altitude measurement
V
Barometric altitude measurement
Legend
VP
...
Variation point
V
...
Variant
1..1
Relationship type (only one of the variants may be selected for each product)

Figure 29: Example of explicit documentation of variability as an orthogonal model

The following variability model (see [PBL2005]) describes the dependencies that can exist between variation points and variants. The variability dependency describes how the variants of a variation point can be combined with one another. The following relationship types exist here:

- **Alternative relationships**: to express that at a variation point, either variant 1 (GPS-based altitude measurement) or variant 2 (barometric altitude measurement) must be selected (see Figure 29).
- **Optional relationships**: to express that a variant may be selected at a variation point—for example, saving the altitude difference covered.
- **Mandatory relationships**: to express that a variant must be selected at a variation point (that is, a variant is a mandatory component at this variation point)—for example, the setting of the metric or English measuring system for height measurement.

The relationship dependency describes how variants or variation points can be combined with one another. The following commonly used relationship types exist here:

- **Requires**: to express, for example, that the selection of one variant requires another variant to allow it to be realized in a specific product. To continue our example, the variant for barometric altitude measurement requires an air pressure sensor as well as the GPS module.
- **Excludes**: to express, for example, that one variant is excluded by the selection of another variant as the variants exclude each other mutually for a product. Again, to continue our example, the selection of the GPS-based altitude measurement excludes the selection of the barometric weather forecast as, like the barometric altitude measurement, this can only be realized via an air pressure sensor.

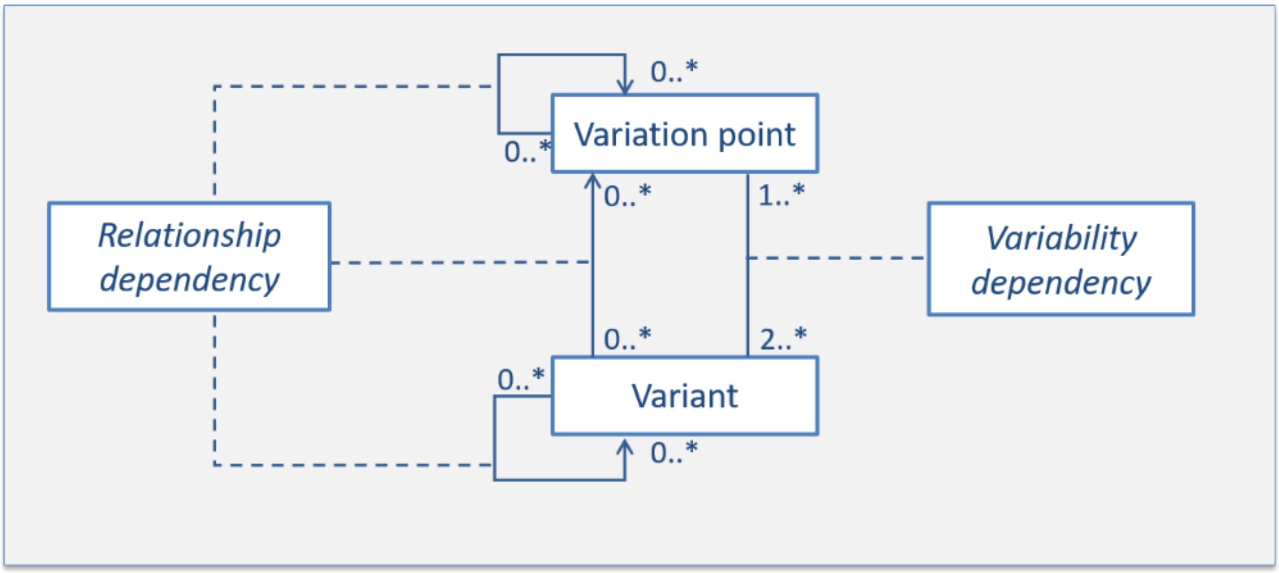


Figure 30: Variability model

Different products can be defined for implementation, taking account of the variability and relationship dependencies. The point at which variability is resolved—that is, the point at which specific variants must be selected to get a specific product—is referred to as the "binding time". According to [CHW1998], variability can be bound before development (i.e., before the creation of the product), on realization (i.e., at implementation), on creation of the software (i.e., at compilation), on initial installation, or even at runtime.

The later that variants are bound to make a product more specific, the more the term variability blurs with the term configuration. The following examples are intended to make this clear:

- **Binding time of a variant before product realization**: This means, for example, that a customer decides on a specific product variant before implementation. For this bound variant, a subsequent change to another variant is no longer possible. For example, a customer wants a hiking watch with barometric altitude measurement.
- **Binding time of a variant during initial installation**: This means, for example, that a customer decides on a specific product variant at installation or commissioning. The variant selected can no longer be changed at runtime.
- **Binding time of a variant during runtime**: This means that at any point during runtime, for example, the customer can select a specific product variant. For example, subsequent purchase of functionality that enables a hiker who has become lost to find their way back to their starting point using a watch.
- **Binding time during runtime as configuration**: Similarly to the last aspect, during runtime, a customer can, for example, make changes to their product—for example, select the colors for the display (monochrome/color), select the language (German, English, French, Portuguese).

We can use the option to document variation points and variants in requirements for more than just product line development. The use of variability also helps us to document real requirements variants for which the stakeholders have not been able to agree on a specific requirement cleanly and to have them evaluated and estimated by the subsequent phases. Furthermore, with variants and variation points, we can also document the configuration settings that can be selected—which are not necessarily attributable to a product line development—for example, changing the language, so that they are easier to understand.

In product line development in general and in requirements management in particular, explicit documentation of variability has the following advantages [Pohl 2010]:

- **Communication**: The explicit documentation of variation points and variants supports communication with the stakeholders as it is easy to see which variants can be selected at which points and under which conditions.
- **Decision support**: The explicit documentation of variability leads on the one hand to more conscious decisions about the points at which variability should be provided. On the other hand, explicit documentation supports the use of variability to select specific variants for a given product.
- **Traceability**: When requirements are changed, the explicit documentation of variability—including the relationships to the respective requirements artifacts—allows the dependent requirement variants to be determined and adapted where necessary. The orthogonal documentation of variability thus gives us the required traceability for requirements variants.

7.2 Forms of Explicit Documentation of Variants and Evaluation of These Forms

As already mentioned at the beginning, in practice, variability is often formulated directly in the requirements. These forms of explicit documentation use the concepts introduced in Section 7.1, such as the variation point, variant, variability dependencies, relationship dependencies, and the documentation of binding times in very different ways.

In practice, there are a number of different textual forms of presentation, and we will look at the following representatives of these more closely in the following sections (see [Bout2011]).

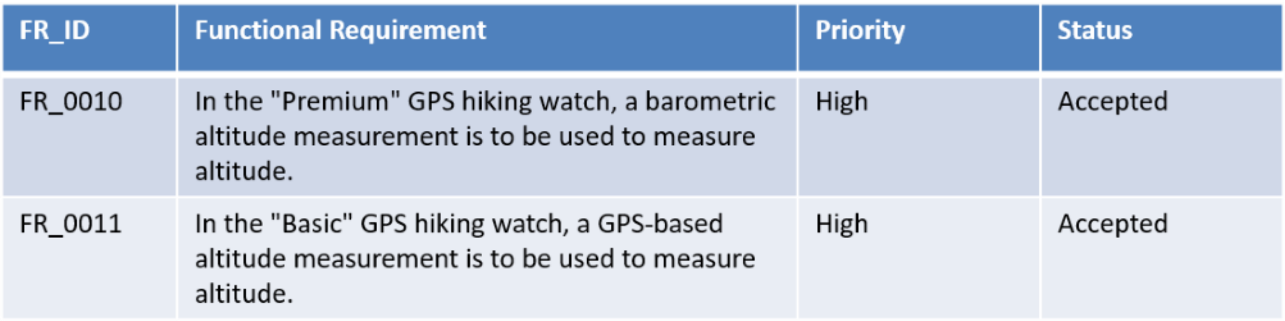
- Textual Assignment of Requirements to Specific Products
- Explicit Assignment of Requirements to Specific Products
- Explicit Assignment of Requirements to Specific Product Features
- Indirect assignment of requirements to features of specific products

We will then analyze these forms of presentation in terms of the concepts for variability presented in Section 7.1 and present additional criteria for evaluating different forms of presentation for variability.

Definition 7-6: Feature: A feature is a property or quality of a system that is visible for the user.

7.2.1 Textual Assignment of Requirements to Specific Products

One form of documenting variants is to document them in individual requirements with the textual assignment of the product which the respective variant is valid for (Figure 31).



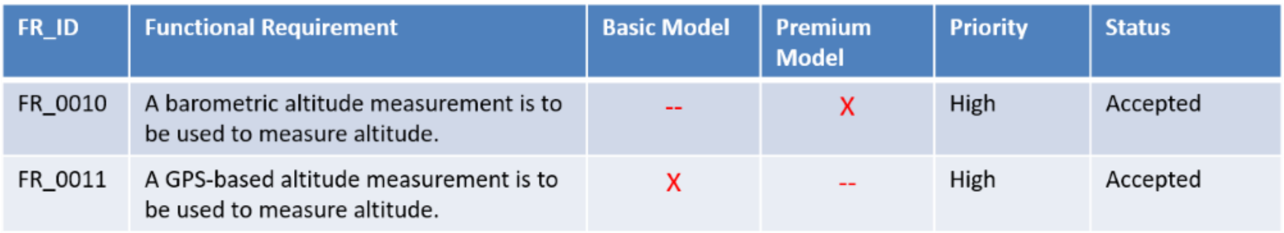
FR_ID	Functional Requirement	Priority	Status
FR_0010	In the "Premium" GPS hiking watch, a barometric altitude measurement is to be used to measure altitude.	High	Accepted
FR_0011	In the "Basic" GPS hiking watch, a GPS-based altitude measurement is to be used to measure altitude.	High	Accepted

Figure 31: Example of a textual assignment to specific products

In this example, we see two requirements in slightly different forms. FR_0010 states that for the premium GPS hiking watch, a barometric altitude measurement should be used. FR_0011 states that for the basic GPS hiking watch, a GPS-based altitude measurement should be used. In the requirement variants, the product names "premium" and "basic" describe which variant should be used in which product. This is already a big added value for the implicit presentation of variability in requirements—think back to our example in Figure 27.

7.2.2 Explicit Assignment of Requirements to Specific Products

Another option for documenting variants is the explicit assignment of requirement variants to specific products, see Figure 32.



FR_ID	Functional Requirement	Basic Model	Premium Model	Priority	Status
FR_0010	A barometric altitude measurement is to be used to measure altitude.	--	X	High	Accepted
FR_0011	A GPS-based altitude measurement is to be used to measure altitude.	X	--	High	Accepted

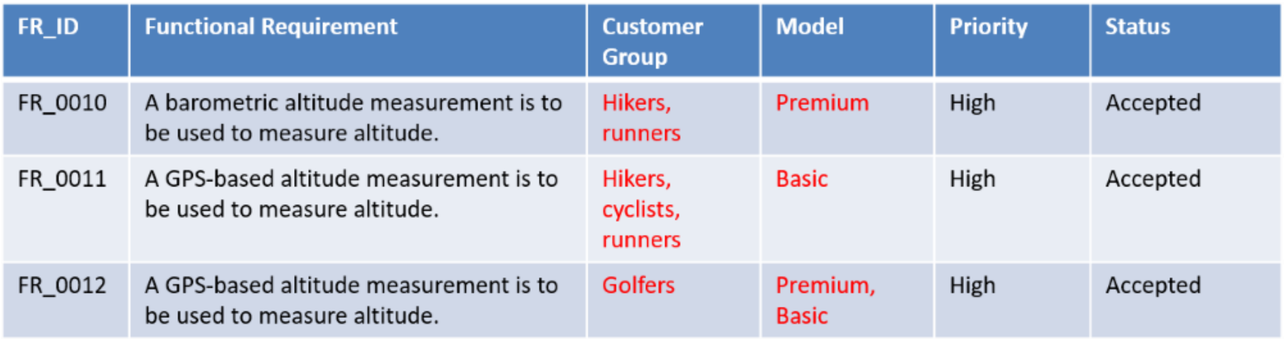
Figure 32: Example of explicit assignment to specific products

In this example, the requirement variants are assigned to the respective product not via a textual designation in the requirement text, but explicitly via a separate attribute. At first glance, therefore, we can already see that each of the two requirement variants is valid only in a specific product. Here, the assignment to products (product variants) is represented by one product attribute in each case. The respective requirement variant is assigned to the respective product with an "X" (e.g., FR_0010 to the product "Premium Model"). Of course, the explicit assignment to products can take another form—for example, via a single attribute "Product" with the respective products as values of the attribute. The specific implementation used depends on the number of possible products and the assignment of the variants.

If specific requirement variants are valid for multiple products, for example, then the assignment via single attributes per product is probably better than the assignment via attribute values.

7.2.3 Explicit Assignment of Requirements to Specific Product Features

In reality, the assignment to specific products often leads to a large number of products. This is because specific products are often defined via multiple dimensions or product features—for example, segments (basic and premium), markets (Europe, USA, Asia), customer groups (hikers, runners, cyclists, golfers). If we assume that all variants can be combined with one another freely, we get 2 x 3 x 4 = 24 products.



FR_ID	Functional Requirement	Customer Group	Model	Priority	Status
FR_0010	A barometric altitude measurement is to be used to measure altitude.	Hikers, runners	Premium	High	Accepted
FR_0011	A GPS-based altitude measurement is to be used to measure altitude.	Hikers, cyclists, runners	Basic	High	Accepted
FR_0012	A GPS-based altitude measurement is to be used to measure altitude.	Golfers	Premium, Basic	High	Accepted

Figure 33: Explicit assignment to specific product features

In the example (Figure 33), we show the option of explicit assignment of requirement variants to product features—for example, to the model and customer group.

Specific products would be, for example, the combinations premium hiker and premium runner. Requirement FR_0010 "barometric altitude measurement" is assigned to both of these products. In contrast, for the customer group "Golfer", only GPS-based altitude measurement is offered, for both the premium and the basic model.

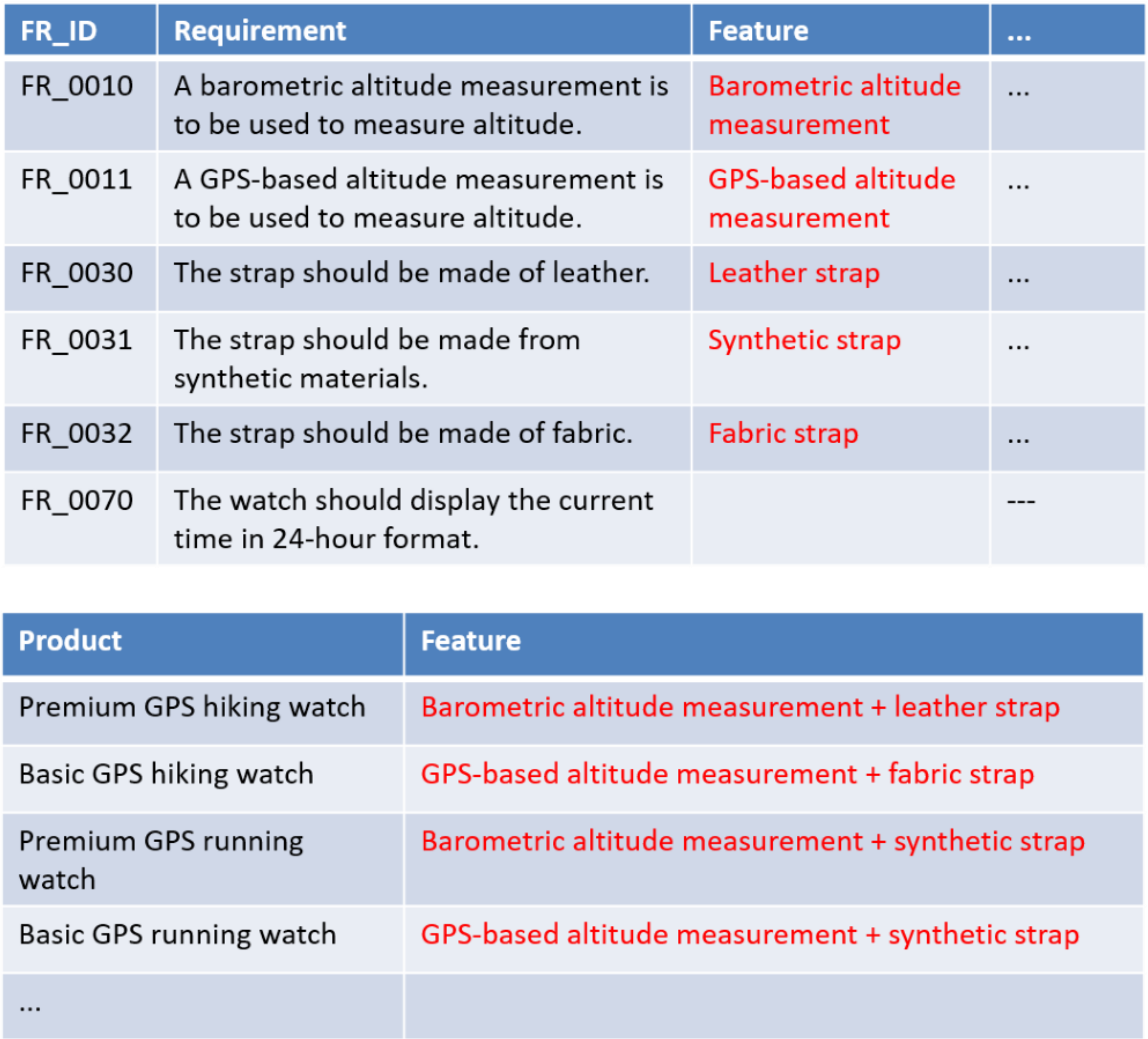
Note about the assignment: Requirements FR_0011 and FR_0012 describe the same requirement from a content perspective. As requirements artifacts, they differ solely in the assignment to the customer group and model. The golfer watch is to have only the barometric altitude measurement—regardless of the model—and therefore requirement FR_0011 was duplicated because a unique assignment would not have been possible otherwise. If we had added the value "Golfer" to the attribute "Customer Group" for FR_0011, and the value "Premium" to the attribute "Model", FR_0011 would be valid for undesired products (e.g., Premium running).

7.2.4 Indirect Assignment of Requirements to Products through Features

Another option for assigning requirement variants to specific products is the assignment of requirements to features. Here, features are special properties of the requirements that describe the variability. In the example shown below (Figure 34), the "Leather strap" is a feature of the associated requirement FR_0030. Here, features abstract from the total requirement and look essentially at the property visible for the user (see Section 7.3).

In the example shown below, the requirement is assigned to a feature—for example, FR_0011 is assigned to the feature "GPS-based altitude measurement". These features can often be found on product packaging or similar, for example, to indicate to the potential customer which features the product has. The second table below in Figure 34 shows the assignment of features to specific products.

For example, the premium GPS hiking watch has a barometric altitude measurement and a leather strap, whereas the basic GPS hiking watch has a GPS-based altitude measurement and a fabric strap. Requirements to which no feature has been assigned (e.g., FR_0070) apply for all derived products—that is, they belong to the common requirements (or commonalities) across all products.



FR_ID	Requirement	Feature	...
FR_0010	A barometric altitude measurement is to be used to measure altitude.	Barometric altitude measurement	...
FR_0011	A GPS-based altitude measurement is to be used to measure altitude.	GPS-based altitude measurement	...
FR_0030	The strap should be made of leather.	Leather strap	...
FR_0031	The strap should be made from synthetic materials.	Synthetic strap	...
FR_0032	The strap should be made of fabric.	Fabric strap	...
FR_0070	The watch should display the current time in 24-hour format.		---

Product	Feature
Premium GPS hiking watch	Barometric altitude measurement + leather strap
Basic GPS hiking watch	GPS-based altitude measurement + fabric strap
Premium GPS running watch	Barometric altitude measurement + synthetic strap
Basic GPS running watch	GPS-based altitude measurement + synthetic strap
...	

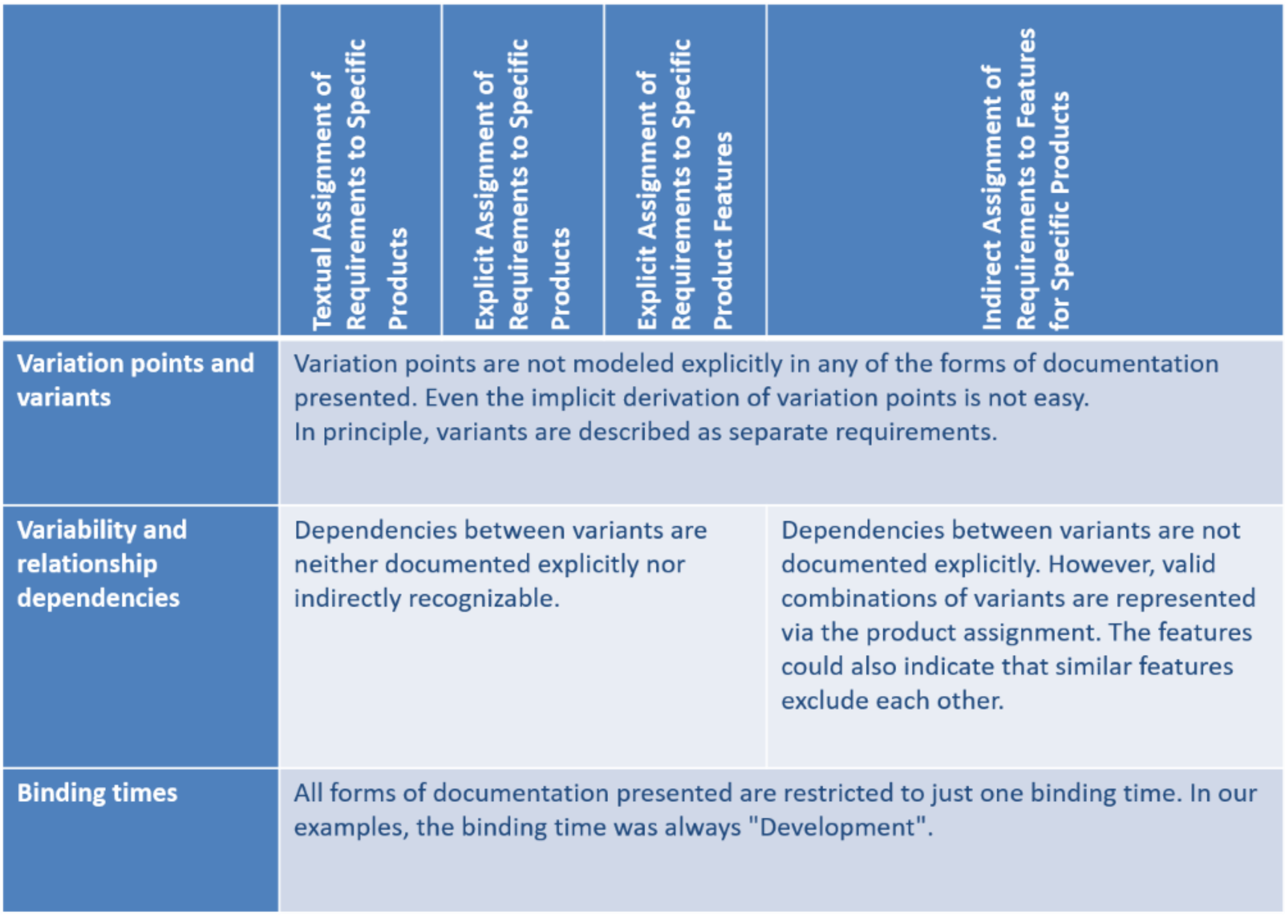
Figure 34: Example assignment of requirements to features of product configurations

7.2.5 Comparison of the Forms of Presentation

To compare the forms of presentation, we will use the aspects for reflecting variability introduced in Section 7.1 and ask the following questions:

- Are variation points and variants differentiated and are they recognizable?
- Are dependencies (variability dependency, relationship dependency) reflected for the permissible variant configurations and are these recognizable?
- Are different binding times considered for variants?

The image below (Figure 35) compares the above three aspects with the four forms of presentation presented. The rows represent the criteria and the columns the forms of presentation.



	Textual Assignment of Requirements to Specific Products	Explicit Assignment of Requirements to Specific Products	Explicit Assignment of Requirements to Specific Product Features	Indirect Assignment of Requirements to Features for Specific Products
Variation points and variants	Variation points are not modeled explicitly in any of the forms of documentation presented. Even the implicit derivation of variation points is not easy. In principle, variants are described as separate requirements.			
Variability and relationship dependencies	Dependencies between variants are neither documented explicitly nor indirectly recognizable.			Dependencies between variants are not documented explicitly. However, valid combinations of variants are represented via the product assignment. The features could also indicate that similar features exclude each other.
Binding times	All forms of documentation presented are restricted to just one binding time. In our examples, the binding time was always "Development".			

Figure 35: Analysis of the forms of presentation

When evaluating a specific form of presentation used for variability, the following criteria are also relevant for practical application of the form of presentation in your projects [Bout2011]:

- **Teachability**: How easily can the chosen form of presentation be taught to non-technical personnel?
- **Scalability**: How easily can the chosen form of presentation be used for a larger number of products?
- **Expandability**: How much effort is necessary to configure a new product from existing and new requirement variants?
- **Migratability**: To what extent can existing requirements documentation be further developed in the direction of the chosen form of presentation without explicit variability information?
- **Verifiability**: To what extent can incorrect configurations in the selected form of presentation be automatically identified?
- **Comparability**: To what extent can requirements of different products be easily compared?

- **Changeability**: How easily can existing requirements for a single product be changed without affecting other products in the product line?

7.3 Feature modeling

Unlike orthogonal modeling of variability (see [Pohl2010], [BLP2004]), feature modeling is an integrated modeling of variability in which both the common product features and the variants, together with their dependencies, are described in one feature model.

The most well-known approach to feature modeling originates from [KCHN1990] and was introduced with FODA (*feature-oriented domain analysis)*. Over the years, the original feature model approach has been slightly modified and developed further [KKLK1998], [KLD2002], [SHT2006].

Analog to our definition 7-6 "Feature", the original definition according to [KCHN1990] is as follows:

Definition 7-7: Feature [KCHN1990]: *"a feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system"*.

7.3.1 Creating Feature Models

The common and variable features of a product line, including their dependencies, are described in a feature model. A feature model can be documented in tabular or model-based form. Feature models are typically presented as a graphical model (feature diagram). Feature diagrams originate from and/or trees. In feature models, variation points and variants cannot be clearly distinguished from one another visually (see Figure 36). Depending on the perspective, features are either a parent feature or a child feature, and therefore either a variation point or a variant. The lowest leaf elements can clearly be identified as variants. In contrast, variation points are all non-leaf elements of the tree.

The descriptive elements of a feature diagram can be divided into the following three categories:

- Basic elements (see FODA [KCHN1990])
- Advanced elements (see [CzEi2000])
- Cardinality-based elements (see [RKGSB2002])

The following model describes a metamodel for feature modeling. On the one hand, the model shows the refinement relationship between parent and child features (basic elements), and on the other hand, the dependency relationships between features (advanced elements).

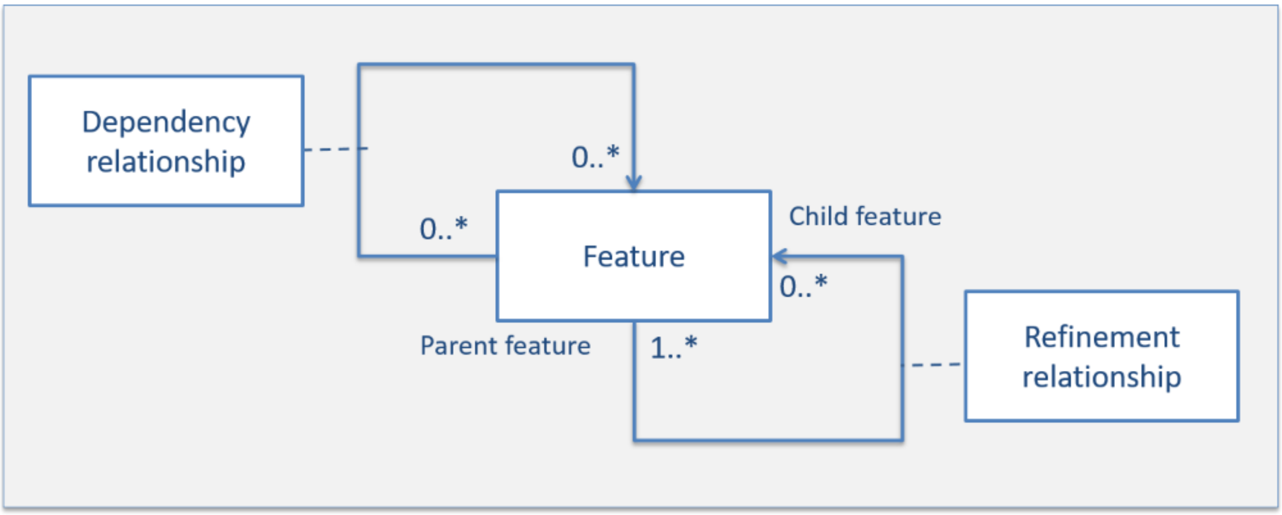


Figure 36: Feature metamodel

The **basic elements** of a feature model describe parent features and their children. The refinement relationship describes which features must be included in the configuration of a specific product and what must be taken into account for the selection of variable features. Child features can have the following relationships with parent features:

- **Mandatory**: The child feature is mandatory for a specific product.
- **Optional**: The child feature can be selected for a specific product.
- **Or**: At least one of the child features in a group must be selected for the creation of a specific product.
- **Alternative**: Exactly one of the child features in a group must be selected for the creation of a specific product.

Using the **advanced elements**, you can define (similarly to in variability models) which additional dependencies have to be taken into account when selecting features. The most common **dependency relationships** are:

- **Requires**: The selection of feature A implies the selection of feature B.
- **Excludes**: Features A and B cannot be contained in the same product.

The notation used for feature models below is based on [CzEi2000]. Figure 37 describes a feature model in which the basic elements are used. The model describes the example we have been using, the "GPS hiking watch". The model shows the "GPS hiking watch" product presented as the parent feature, as well as three direct child features. The feature "Weather forecast" is an optional feature for the GPS hiking watch, expressed via the connection with the empty circle. The two features "Distance measurement" and "Altitude measurement" are mandatory features for the GPS hiking watch. This is expressed via the relationship with the filled circle between the parent and child features. In turn, the feature "Altitude measurement" has refinement relationships to two further child features: "Barometric altitude measurement" and "GPS-based altitude measurement".

These two child features are connected to the parent node via an "alternative" relationship, which means that only one of the two features may be included in a product configuration. "Alternative" relationship types are represented by an empty arc across all child features, of which only one may be selected. In contrast, "or" relationships, which allow the selection of multiple child features, are represented by a filled arc.

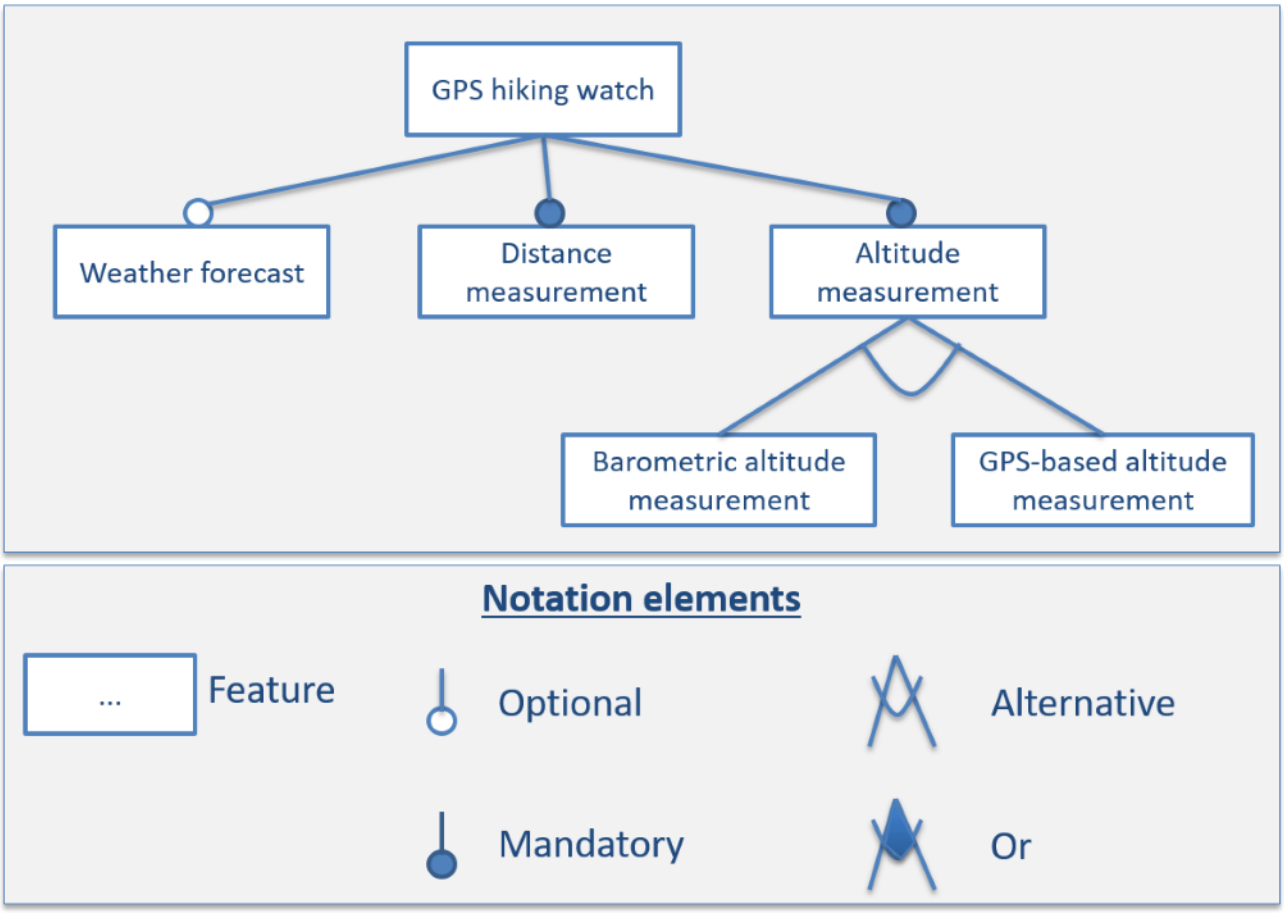


Figure 37: Example feature model with basic elements

Figure 37 shows a normalized form of a feature model. In principle, the single refinement relationships "Optional" and "Mandatory" can be combined with the group refinement relationships "Alternative" and "Or" (see the two examples in Figure 38). Even if the three models appear different at first glance, the meaning of all the models is identical here. The group refinement relationships "Or" and "Alternative" have a higher value here than the single refinement relationships "Mandatory" and "Or".

In Figure 38, we can see from the two examples that the "Alternative" relationship has a higher priority than the "Mandatory" and "Or" relationships between the parent and child features. Here, regardless of the single parent-child relationship ("Optional" or "Mandatory"), only one of the two child features "GPS-based" or "Barometric" can be selected. The single refinement relationship at the child feature can be ignored here. Therefore, in group refinement relationships, we generally find only the single link to the child feature (see Figure 37).

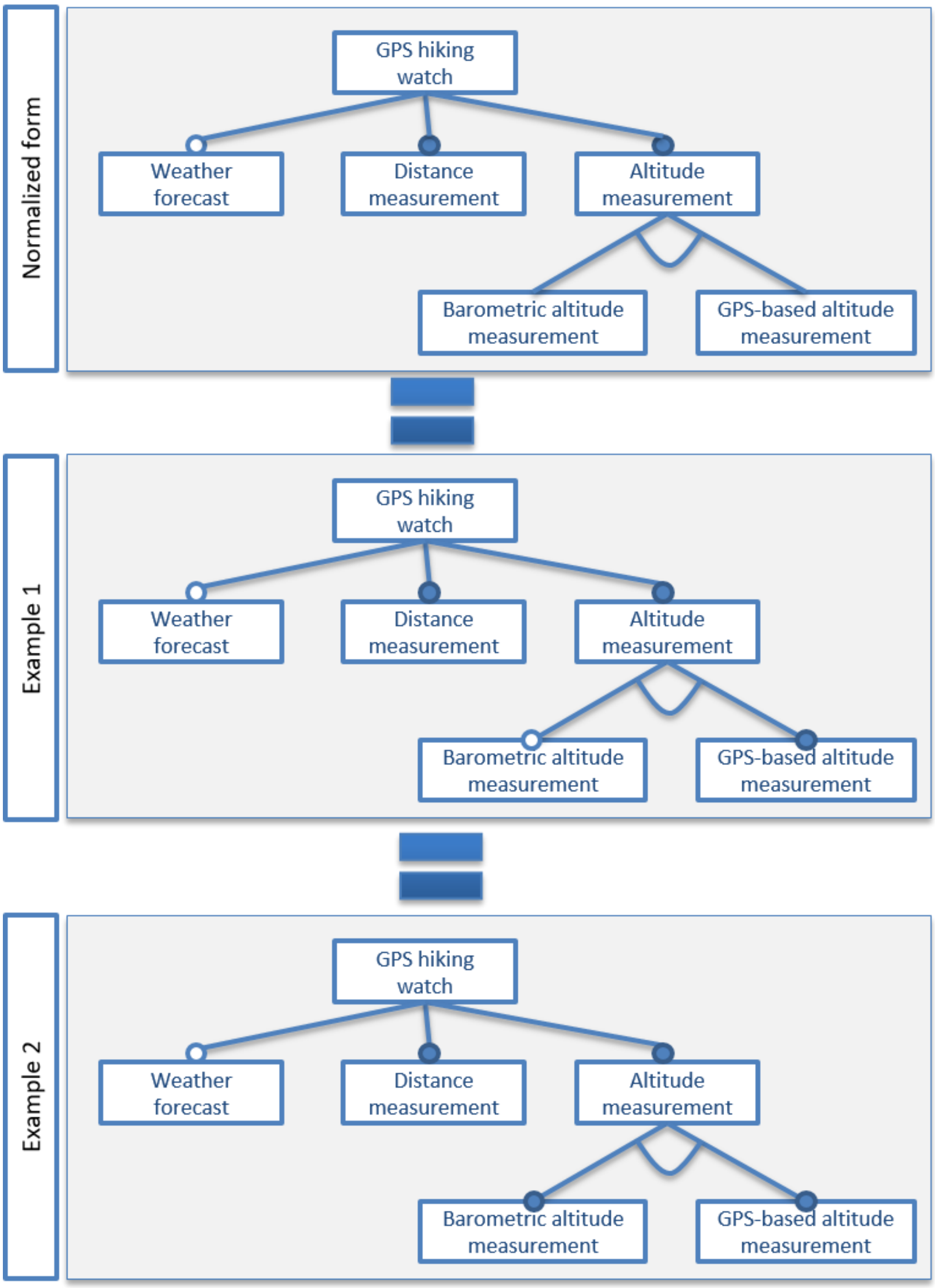


Figure 38: Example combination of "Alternative" with "Optional" and "Mandatory" relationships

Note: The group relationship types "Alternative" and "Or" have a higher priority for the selection, which means that the single parent-child relationships "Optional" and "Mandatory" for the children within a group are not considered.

Building on the example above, the example in Figure 39 shows how the advanced elements (here the dependency relationships) are presented graphically in a feature model. For this purpose, a "Requires" relationship was added to the model between the features "Weather forecast" and "Barometric altitude measurement". This relationship type states that if the optional feature "Weather forecast" is selected, the alternative feature "Barometric altitude measurement" must also be selected.

The "Requires" relationship is represented by a dotted arrow with the tip on the required feature element. The relationship type "Excludes" is represented by a dotted arrow with a closed tip in the direction of both features.

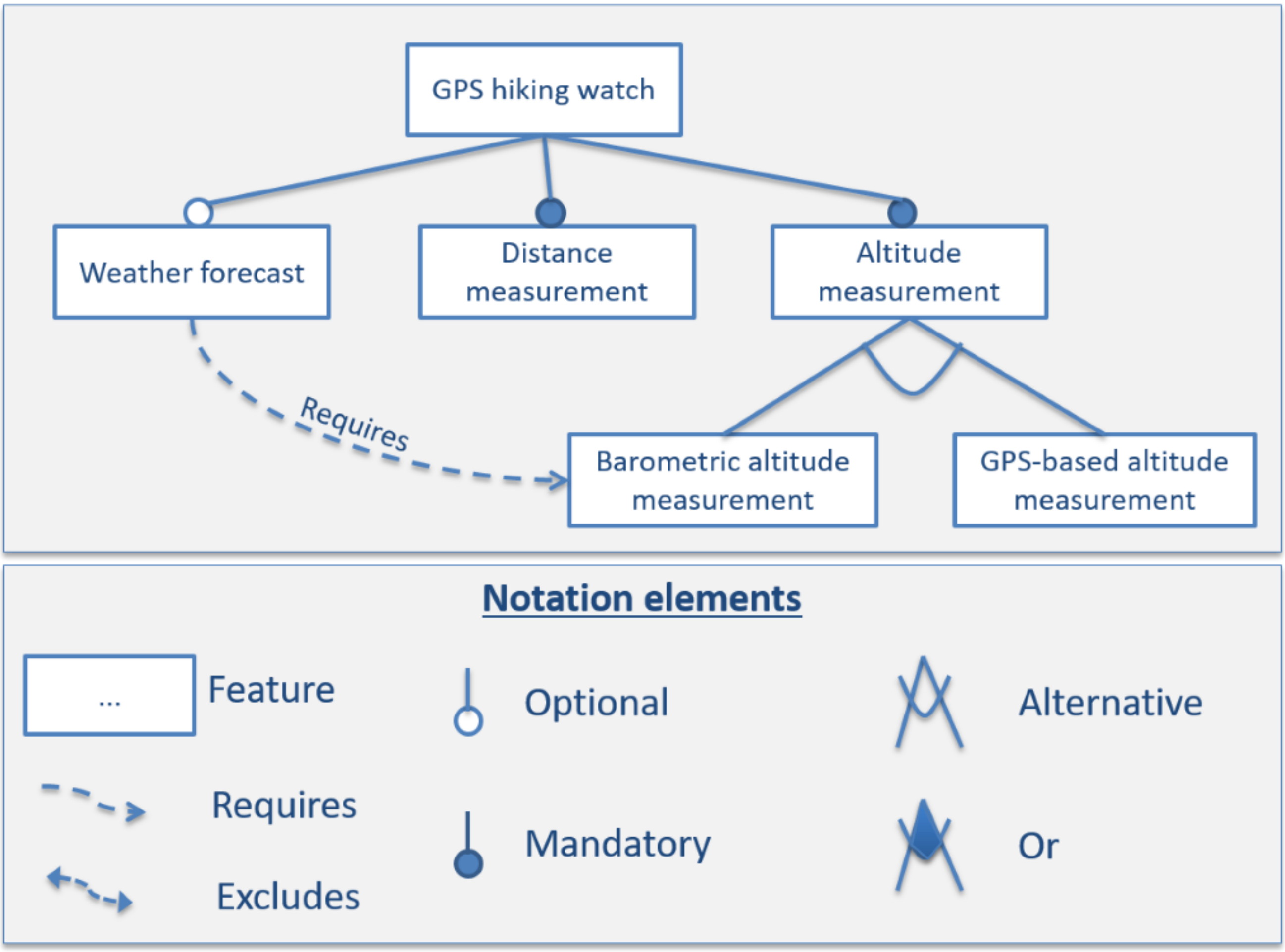


Figure 39: Example feature model with advanced elements

Cardinality-based elements can be used to further specify the **refinement relationships** between basic elements, for example by adding notations such as [min, max] to the parent-child relationships. This allows you to express, for example, that in the event of an "Or" selection, not all but rather a maximum of two child features may be selected (e.g., only two of the 14 languages may be selected). For this purpose, we would add the desired cardinality to the "Or" and "Alternative" relationship types.

7.3.2 Deriving Product Configurations from Feature Models

To create specific products, the variable features of a feature model must be bound at a certain point in time (see Section 7.1). To determine how many different products can be derived from one feature model, the model must be "multiplied out". For this purpose, starting from the root (that is, the uppermost parent feature), all product configurations possible based on the refinement and dependency relationships are defined.

Example: Product configurations of a feature model

Based on the feature models introduced above, the following two examples should show which product configurations the models permit. The example in Figure 37 allows four different products:

- **Product 2: GPS outdoor watch + weather forecast + distance measurement + altitude measurement + barometric altitude measurement**
- **Product 2: GPS hiking watch + distance measurement + altitude measurement + barometric altitude measurement**
- **Product 3: GPS hiking watch + weather forecast + distance measurement + altitude measurement + GPS-based altitude measurement**
- **Product 4: GPS hiking watch + distance measurement + altitude measurement + GPS-based altitude measurement**

In contrast, the example in Figure 39 allows only three different products, as the "Requires" relationship between "Weather forecast" and "Barometric altitude measurement" excludes the combination with the "GPS-based altitude measurement"—that is, product 3.

- **Product 1**: GPS hiking watch + weather forecast + distance measurement + altitude measurement + barometric altitude measurement
- **Product 2**: GPS hiking watch + distance measurement + altitude measurement + barometric altitude measurement
- **Product 3**: GPS hiking watch + distance measurement + altitude measurement + GPS-based altitude measurement

7.3.3 Identifying Features

Features are not generally defined on a "greenfield" basis; they must be identified and defined based on existing system documentation, requirements documents, etc. [BoHo2011] describes a semi-automatable approach for identifying features from existing specifications in four steps.

- **Step 1: Search for nouns**: Requirements texts are examined for nouns as the starting point for identifying features.
- **Step 2: Normalization**: In the next step, the nouns found in step 1 are normalized—that is, they are cleaned up from a language perspective and put into their basic form (e.g., plural nouns are put into the singular form).
- **Step 3: Removal of duplicates**: In the third step, duplicates are removed from the list of normalized nouns.
- **Step 4: Removal of stop words**: Finally, general nouns that have nothing to do with the product itself are removed from the list (e.g., words that deal with contractual or general development aspects in the project) so that the result is a list of candidates for features.

In this approach, the primary objective is to support you in identifying possible variation points and variants from existing, textual requirements so that you can use these as a basis for creating a feature model. This is particularly helpful if you already have documents that you regularly reuse—but with no explicitly documented variability—but have not yet defined a variability model or feature model for your new product line.

Example: Identifying features from a requirement text

The **input for the analysis** is the following requirement:

R_1020: The customer should be able to choose between a metal strap, a fabric strap, or a leather strap as the strap for the GPS hiking watch.

With step 1 (search for nouns), the following nouns were identified: *strap, GPS hiking watch, customer, metal strap, fabric strap, leather strap*.

In our example, there will be no change for **steps 2 and 3** because we are looking at only a singular requirement.

By applying step 4 (removal of stop words), the term *customer* would be removed from the list. This can be identified as a **stop word** here because this noun does not describe a property of the product—even though the customer is the person who is to purchase the product later.

The **output of the analysis** is the following feature candidates: *strap, GPS hiking watch, metal strap, fabric strap, leather strap*.

Based on these noun lists, an expert can then usually quickly identify potential features or variation points and variants. However, one significant disadvantage of this procedure is that in particular, variation points that are not explicitly mentioned in the text cannot be identified. Furthermore, the correct relationship types between the parent and child features generally have to be analyzed from the requirement itself. Variation points (i.e., parent features) and the relationships to the child features can often be identified if the technical expert insistently asks about the reason (i.e., the "why") for the different variants. For example, in response to the question of why the watch sometimes has a fabric strap, sometimes a leather strap, and sometimes a metal strap, the answer is that different customers prefer different straps.

Accordingly, the strap is the corresponding variation point (or rather, the parent feature) and the specific types of strap are the variants (or rather, the child features).

7.3.4 Tool Support

If you want to document variability explicitly, this is difficult to do without using special tools. Of course, you can create feature models as and/or trees with existing modeling tools—however, these generally do not support any relationship types for variability or the derivation of product configurations.

However, there are a number of tools on the market that allow you to:

- Create feature models
- Create product configurations

- Check product configurations for reliability

These tools generally have interfaces to other modeling or requirements management tools in which the actual development artifacts (e.g., requirements) are located. This enables you to place variability models or feature models in relationships with other development artifacts so that you can establish the traceability between the different models.

7.4 Content for the Requirements Management Plan

Where necessary, the aspects of variability modeling presented in this chapter can be added to the requirements management plan if you want to either represent variants or even develop a true product line. In your requirements management plan, define how you want to document variability—that is, variation points, variants, and their dependencies—in your requirements. You can do this for example in text form, as an orthogonal model, or as a feature model (e.g., Figure 39). What is important here is that you define explicitly how variability is to be documented (e.g., using feature models) so that you can discuss and agree this with all stakeholders involved before the project starts.

7.5 Literature for Further Reading

[Bout2011] E. Boutkova: Experience with Variability Management in Requirement Specifications. In: D.E. Almeida, T. Kishi, C. Schwanninger, I. John, and K. Schmid (eds): Software Product Lines – 15th International Conference (SPLC), München, 2013, pp. 303-312.

[BoHo2011] E. Boutkova and F. Houdek: Semi-automatic identification of features in requirement specifications. In: Proceedings of the 19th International Requirements Engineering Conference, Trento, Italy, September 2011.

[BLP2004] Bühne, S.; Lauenroth, K.; Pohl, K.: Why is it not Sufficient to Model Requirements Variability with Feature Models. In: Aoyama, M.; Houdek, F.; Shigematsu, T. (eds) Proceedings of Workshop: Automotive Requirements Engineering (AURE04). IEEE Computer Society Press, Los Alamitos 2004.

[CHW1998] J. Coplien, D. Hoffmann, and D. Weiss: Commonality and Variability in Software Engineering. In: IEEE Software, Volume 15, Issue 6, 1998.

[ClNo2007] P. Clements and L. Northrop: Software Product Lines: Practices and Patterns. Addison Wesley, Boston, 6th Edition, 2007.

[CzEi2000] K. Czarnecki and U.W. Eisenecker: Generative Programming: Methods, Tools, and Applications. Addison Wesley, 2000.

[KCHN1990] C. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson: Feature-Oriented Domain Analysis (FODA) - Feasibility Study. Software Engineering Institute, 1990.

[KKLK1998] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures," Annals of Software Engineering. vol. 5, 1998, pp. 143–168.

[KLD2002] K. Kang, J. Lee, P. Donohoe: Feature-Oriented Product Line Engineering. IEEE Software 19(4): 58-65 (2002).

[PBL2005] K. Pohl, G. Böckle, F. van der Linden: Software Product Line Engineering - Foundations, Principles, and Techniques. Springer, 2005.

[Pohl2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, 2010.

[SHT2006] P.-Y. Schobbens, P. Heymans, J.C. Trigaux: Feature Diagrams: A Survey and a Formal Semantics. In: Proceedings of the 14th International Requirements Engineering Conference (RE'06), September 2006.

8 Reporting in Requirements Management

8.1 The Goals and Benefits of Reporting in Requirements Management

Reports are part of project and organizational controlling. They serve to collect information about projects or organizational units and to prepare this information appropriately for certain target groups in order to meet their information needs.

Reporting is defined, for example, as "the creation and dissemination of cross-functional reports in the sense of an organized compilation of messages exclusively for management" [Zieg1998]. Another definition emphasizes the preparation and goals of the reporting system: "It can be understood as all persons, facilities, regulations, data and processes used to create and distribute reports. Thereby, reports represent summarized information under an overarching goal, an information purpose." [Küpp2005].

Reports are the specific, technical implementation of views, thus "an extract from an artifact that contains only the content that is currently of interest" (see Chapter 3).

You can use reports to find out how much work has already been completed in a project and the quality of this work. This information is used for project controlling and quality assurance. Specifically, the information supports:

- Knowledge about the status of the project progress
- Transparency about the project progress for management and the team itself
- Early detection of deviations of the actual progress from the target progress
- The ability to make reliable, important management decisions as early as possible (e.g., whether the delivery date has to be postponed and if so, by how much)
- A reduced view of the relevant data that focuses on the essentials

Definition 8-1: Reporting in requirements management is the collection, evaluation, and presentation of information about requirements or the requirements engineering process. The information contained in reports serves not only as pure information but also as a basis for project decisions and for controlling the requirements engineering process.

Definition 8-2: A **report** is a document that combines one or more views for a specific stakeholder and purpose.

In connection with requirements management, in this book we are interested primarily in the contribution the requirements manager makes to reporting. In particular, we demonstrate how requirements-based project controlling can work—that is, observing the project progress using the information in the requirements management tool.

Of course, the prerequisite for creating such a report is that the requirements management tool contains the corresponding information. This information is usually defined in the form of attributes that have precisely the value list required for reporting.

Therefore, when you create the attribute schema, you have to think about the reports that are to be created. Ultimately, the purpose of the attributes lies not in their collection but in their evaluation.

Both product and process key figures are of interest for reporting. In this context, key figures are dimensions for both the scope and quality of the product development results that have already arisen and for the status and quality of the development process: "*The purpose of the Measurement Process is to collect, analyze, and report data relating to the products developed and processes implemented within the organization, to support effective management of the processes, and to objectively demonstrate the quality of the products.*" [ISO29148].

One of the difficulties of reporting is that the people who are responsible for entering the required information in the requirements management tool are not the same people who use this information. The result is that not only do the project team members have little intrinsic motivation for entering the data—because they themselves do not benefit from it—but also that they may even want to sugarcoat the true status of the project to the controlling instance.

The goals and stakeholders of reporting

The primary goal of the further development of online banking is to introduce the new release properly and without any malfunctions. At an early stage (with at least three weeks lead time), a number of persons and parties affected by the development must be informed about which functionality is being introduced at which point in time. These persons and parties include management, the data center, the service representatives in the call center, and the employees in the branches of the bank. At the time promised, the respective functionality must work without any errors.

What is important in reporting, therefore, is a status tracking for the next release. This status tracking must detect deviations from the schedule at an early stage and also ensure that any errors that may have been implemented are discovered and eliminated by the time of delivery.

Here, delivery reliability and quality are more important than costs and the scope of delivery. If there were any doubt about a functionality, it would be omitted rather than being delivered with errors. Furthermore, additional costs in the form of developer days would be invested rather than postponing the deadline.

At the same time, Internal Controlling, which controls the flow of costs across all projects, is keeping an eye on this project and wants to know what costs are incurred each month. They are particularly interested in whether the budget granted has been or will be exceeded or not consumed in its entirety.

The project manager is the person who evaluates the status and quality of the project using the reports and then communicates the conclusions the draw from the reports to the other stakeholders. The project manager creates an extensive status report for the Controlling department, for their own department head, and for the project team.

For the remaining stakeholders, the project manager has set up a newsletter that provides brief information on a weekly basis about the planned release date and the functionality that the release will probably contain. This is a selective extract of the information contained in the extensive status report.

The data for these reports, which refer to the requirements and the requirements engineering process, comes from the requirements management tool that manages the status of the requirements over their entire lifecycle. Therefore, for the project manager, Peter Reber is the person who delivers the requirements-based content for the status report. Further data comes from development and from quality assurance, and these functions each use different tools for managing their content and for creating status information.

8.2 Establishing a Reporting System in Requirements Management

8.2.1 Interfaces

Requirements management is closely integrated with project management, product management, and quality management. Consequently, these interfaces also exist within the reporting system. Therefore, it makes sense to coordinate the reporting of these three areas and their data. Project management will certainly be one of the most important recipients of reporting on requirements management but quality management must also be considered. In some companies, where applicable, it may even make sense to generate reports to cover both requirements management and quality management. This should be checked on an individual basis.

8.2.2 Contents of a Report

Reports can be sent informally in an email text. However, there are often templates to ensure that every report has the same structure. This makes reports easy and efficient to read and create: the same information is always available in the same place on the page. For the author, it is particularly practical if the report can be generated automatically from the tool in which the necessary information is managed —that is, it is practical if the requirements management tool can create status reports.

Reports are important project documents and must therefore be stored such that they can be traced. Here, all rules of good document management apply. Ideally, the report files should have meaningful and unique names that also contain the date of creation of the report in a form that, where alphanumeric sorting is used, leads to files being sorted in chronological order. A good example of a file name is status_20140817.docx. In some project crises, conflicts, and disputes, the reports represent valuable information about the progress of the project and the flow of information—that is, who has informed who about what and when did they do so?

A report not only contains one or more views and key figures—it also documents its own creation and approval process. The report must also clearly state what it refers to—for example, the specific project and reporting period.

Sometimes there are different reports about the same content but with a different purpose, with different titles that provide information about the target group and purpose—for example, the status report or the management summary.

Of course, every type of report contains different information, a different view. A report should contain only the information that the recipient needs to cover their information need.

8.2.2.1 Key Figures in Requirements Management

Key figures or measures are an important part of reports.

"You can't control what you can't measure" is a statement by DeMarco [DeMa1982] that is often quoted. Twenty-seven years later [DeMa2009], DeMarco discusses that not every project needs the same level of control, that not everything can be controlled, that control is not everything and not the most important management task. According to DeMarco, management of human resources is also important, for example. That may be correct, but it is still true that measurement simplifies control. Specific figures and hard facts supplement or correct our intuitive impressions in an engineering-based processing and management of a project. This also applies in particular to the management of requirements and changes to these requirements, that is, requirements management.

Ebert [Eber2012] defines a measure as:

> "(1) A formal, precise, reproducible, objective assignment of a number or symbol to an object to characterize a specific characteristic.
>
> (2) Mathematical: Figure M of an empirical system C and its relations R in a numerical system M.
>
> (3) The use (collection, analysis, evaluation) of a measure. Examples: Measure for a product (for example, errors, duration, deviation from plan) or a process (for example, error costs, efficiency, effectiveness)".

There is a difference between product key figures and process key figures.

Definition 8-3: The **product key figure** measures the scope or quality of the product to be created at a specific point in time.

As we are interested in requirements here, we measure the scope and quality of the product based on requirements: for example, "Which requirements are planned for the next release?" or "How many requirements are ready for delivery?".

Definition 8-4: The **process key figure** measures the progress or quality of the work process.

Here too, we are interested in particular in the requirements perspective, that is, the progress of the development process in terms of the requirements ("How many requirements have already been specified completely?") and particularly the progress of the requirements engineering process ("What proportion of the requirements currently known have already been checked?").

Ebert [Eber2012] differentiates between three types of measures in requirements engineering:

- "Progress (e.g., the number of requirements that have been specified, realized, or tested)
- Requirement quality (e.g., number of errors in the requirements documents)
- Model semantics (e.g., degree of coverage of the requirements by the analysis model)"

The key figure type "model semantics" could also be called the "traceability dimension", as it measures the completeness of traceability between two different requirements specifications. If we consider the requirements to be part of the product, this is also a product key figure.

ISO 29148 ([ISO29148]) makes the following statement about requirements key figures:

"Requirements engineering as a discipline benefits from measuring requirements in both the process and product contexts. More than one measure may be needed to provide the insight into the information needs for the requirements. Practice has consistently proven various useful measures, including:

Requirements volatility – In the process context, requirements volatility can indicate an organization's requirements engineering process will not converge a collection of requirements into a well-formed set. In the product context, a high volatility value can indicate risk early by stakeholders failing to reach consensus on system requirements, putting significant risk on subsequent activities in the life cycle.

Other useful requirements measures include:

- *Requirements trends*
- *Requirements change rate and backlog*
- *Requirements verification*
- *Requirements validation and*
- *TBD and TBR closure progress per plan."*

TBD stands for *"to be determined"* and TBR for *"to be resolved"* or *"to be revised"*, thereby identifying open items directly in the requirements specification.

Various authors recommend the following as requirements-based key figures for tracking the status of the project:

- Status of the requirements, which can be represented over the course of time—for example, the number or proportion of requirements that have been agreed, developed, or completed, see Figure 42
- Change rate = proportion of requirements that have changed in a period, measured over the total scope of the project; measures the stability of the project and its requirements
- Error rate = number of errors per unit (e.g., errors per 1,000 requirements); measures the result of the requirements inspection or software test
- Degree of attribution of the requirements: this key figure measures whether the requirement attributes have been completely filled; the target value is 100%
- Degree of linking between different artifacts
- Requirements coverage: percentage of all requirements that have been validated by at least one test case [SpLi2007]
- Test coverage: criterion for measuring the completeness of the tests executed [SpLi2007]
- Velocity = number of requirements that can be implemented in one iteration in the case of iterative development
- Throughput duration of a change request from application to approval

To determine the key figures above, we need condensed views that calculate totals and subtotals or percentage proportions.

8.2.2.2 Standard Contents of Reports

The standard contents of regular reports in requirements management—for example, reports to project management—are the following:

Project name: The report must specify the project to which it refers. (If the report is about an organizational unit, for example a department, the name of the department is displayed here instead of a project name.)

Date of report creation: The contents presented in the report change daily or even hourly. It is therefore important to specify when the data was extracted, that is, the information status that the report is based on.

Version number: If there are several versions of a report, for example because someone added something, the new version must have a new version number to ensure that the report is unique and to ensure better traceability of changes.

Reporting period: Reports can refer to days, weeks, months, years, or any other time interval. Weekly and monthly reports are the most common, but in critical project phases, reports can also be generated on a daily or half-day basis. Of course, when interpreting the contents of the report, it makes a difference whether it relates to what was achieved within a week or a month.

Creator and recipient(s): A report has a creator (author) and recipient(s) (distribution list). The recipients can also be distinguished between those who receive it for information only and those who have to approve it. The names of these persons are usually mentioned on the report and are thus documented.

Release status: If the report requires a release, this status should be noted here. The report may contain different contents in different release statuses. ***Overall status***: Right at the beginning of the report, a reader in a hurry wants an overview of how critical the project is. Busy managers in particular only read the report if the project is critical. Reports on projects that are running according to plan do not contain any informational value for the supervisor, as their support is not required. Traffic light scales with the following meanings for the colors are popular:

- **Green**: The project is running according to plan. No acute problem, no need for action.
- **Yellow**: The project is not running according to plan. The project team can probably solve the problems themselves. Action must be taken, however.
- **Red**: The project is at risk and the project team cannot or can no longer solve the problems themselves. Urgent support is required from above or outside the project, as well as a decision about, for example, postponing the deadline, increasing the budget, creating a task force.

Statuses can also be specified for individual parts of a project (e.g., phases or work packages) or for individual aspects (e.g., costs, delivery reliability, progress, quality, risk). What is also interesting, particularly just before the delivery deadline, is which of the planned requirements are already ready for delivery and which are not. Here you can also document which milestone has already been reached.

Earned value analysis: The earned value analysis (EVA) is described in more detail in Annex C. As the basis for the earned value analysis, the report specifies the following key figures for the project:

- Budget (budget at completion/planned costs, PC): The budget available for the entire project. The budget is specified in € (or another currency) or in time units (person days or person months). Both details can be specified here, or you can consistently use only one of the two key figures.
- Planned degree of completion: Here, you specify in % the proportion of the project that should be completed at the current point in time. The figure is calculated as the quotient of the planned work volume and the total volume of the project. Here, too, you can use a currency or a time unit. Of course, the work volume and total volume must be measured in the same unit.
- Degree of completion: Here, you specify in % the proportion of the project that is actually completed at the current point in time.
- Costs or effort to date: Here, you specify the costs that have already arisen, in € or in a time unit.
- Cost index: This is the quotient of the costs to date and the total budget of the project in %.

These key figures tell you what part of the result has already been completed and what proportion of the budget has been used to do so. You can therefore calculate whether the project is on schedule and whether work is being performed efficiently—that is, whether the result created matches the budget consumed. Based on these figures, forecasts can be created about whether the project can be completed on time and within budget. This provides the status of the project. For a detailed description of the earned value analysis, see Annex C. These key figures can also be broken down by work packages or requirements. However, this is usually not necessary.

Further key figures: Further key figures can describe the quality of the project results, including the requirements—for example, the number of errors found in the inspection, or the proportion of requirements for which not all attributes have been maintained yet. Further quality assurance results are also interesting—for example, the test coverage, density of errors in the code, and the number of errors or serious errors that are still open. Further examples for key figures can be found in Section 8.2.1.

Of course, the data required for the reports must be available in the requirements information model (see Chapter 2) and in the attribute schema (see Chapter 3).

Evaluation and forecasts: In addition to the figures themselves, the report always requires an evaluation by the creator of the report. The recipients cannot necessarily judge whether a specific value is acceptable or not for this particular project or for the current point in time. Therefore, this evaluation is a significant part of the report.

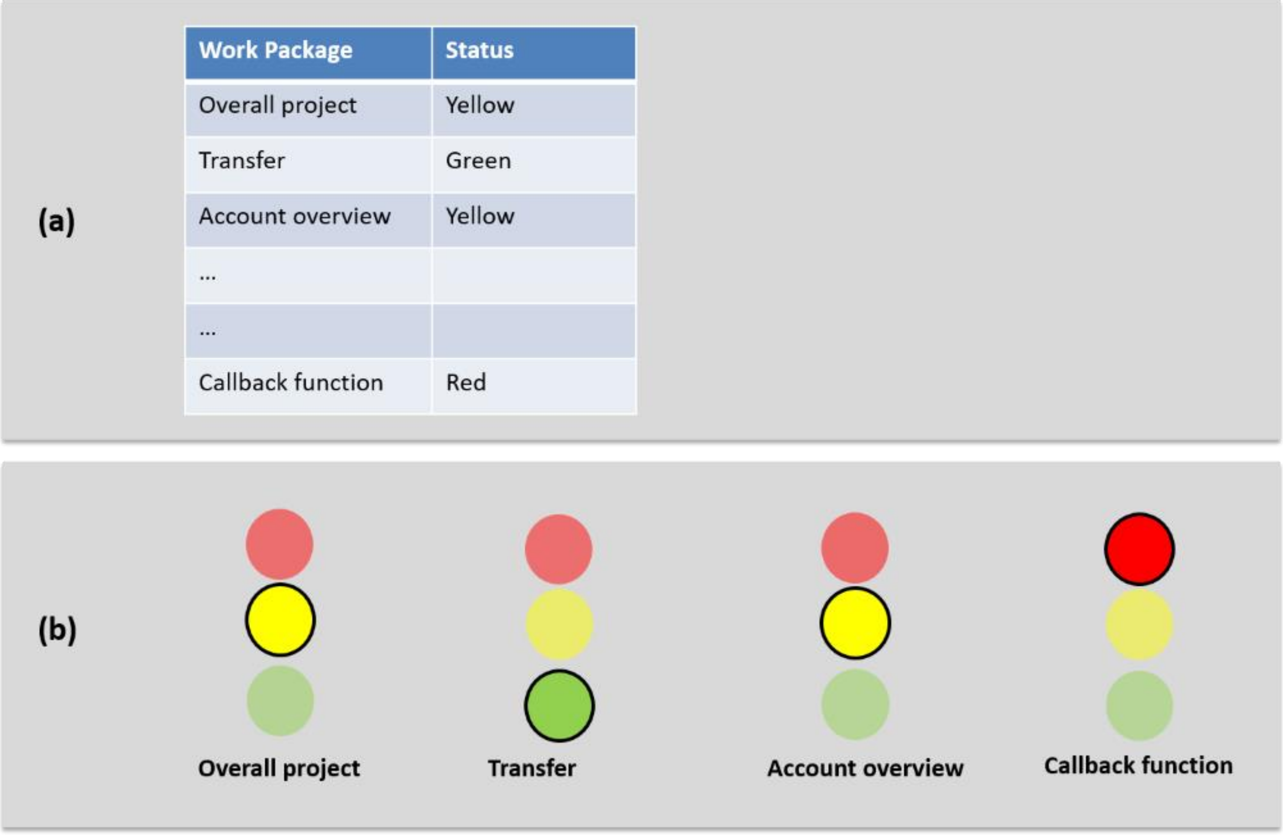
Forecasts are part of this evaluation. The author of a report should create forecasts so that every reader does not have to create the forecasts and draw conclusions themselves. For example, if 25% of the budget has been consumed for a degree of completion of 20% of the work volume, this requires a justification which is in turn important for a forecast. Is the 5% overspend due to a one-time issue—for example, unexpected costs that occurred at the beginning and since then, work has been performed according to plan? In which case, can we hope for the rest of the project that 10% of the budget leads to a 10% degree of completion, meaning that at the end, the costs will be 105% of the budget? However, if the cost overspend can be attributed to the fact that the cost estimation was incorrect, or unforeseen problems are making the work more difficult, there is a fear that this will also apply for the rest of the project. The remaining 80% of the project will then consume a further 100% of the budget, meaning that at the end, the project will cost 125% of the planned budget. In the case of a time delay, the cause can also indicate whether the time can be recovered or whether the final deadline has to be postponed and by how much.

Special events: The figures do not indicate whether anything special has occurred during the reporting period. Special events can be deviations from the plan, risks that have occurred, or extensive change requests. They should be specified here in text form.

Open items: What is still open? What has to be done next, by whom, and by when? The information here is usually only the next tasks, unscheduled tasks, and decisions that have to be taken urgently.

Graphical presentations: In addition to the dry figures, graphical presentations that give an overview at a glance are popular. Some examples are given below.

A colored presentation of the *status as a traffic light:* In particular, if the status of multiple elements (e.g., work packages) is presented, as shown in Figure 40, a graphical traffic light gives a better overview. Compare the tabular part (a) of Figure 40 with the traffic light presentation (b).



(a)

Work Package	Status
Overall project	Yellow
Transfer	Green
Account overview	Yellow
…	
…	
Callback function	Red

Figure 40: Status of the entire project and the requirements as a table (a) and a traffic light (b)

Time diagrams for the costs, the degree of completion, the milestone deadlines, and other project key figures: If we apply these key figures over time, as shown in Figure 41, we get a good overview of their development over the period. In turn, this overview allows us to create forecasts about how the project will develop in the future. A project in which 2% is continuously processed each week will probably—if there are no radical changes—continue to progress only 2% per week, despite all hopes for a miracle. If, after the thirteenth project week, a 26% degree of completion has been achieved with 28% of the budget, the project has consumed too much of the budget.

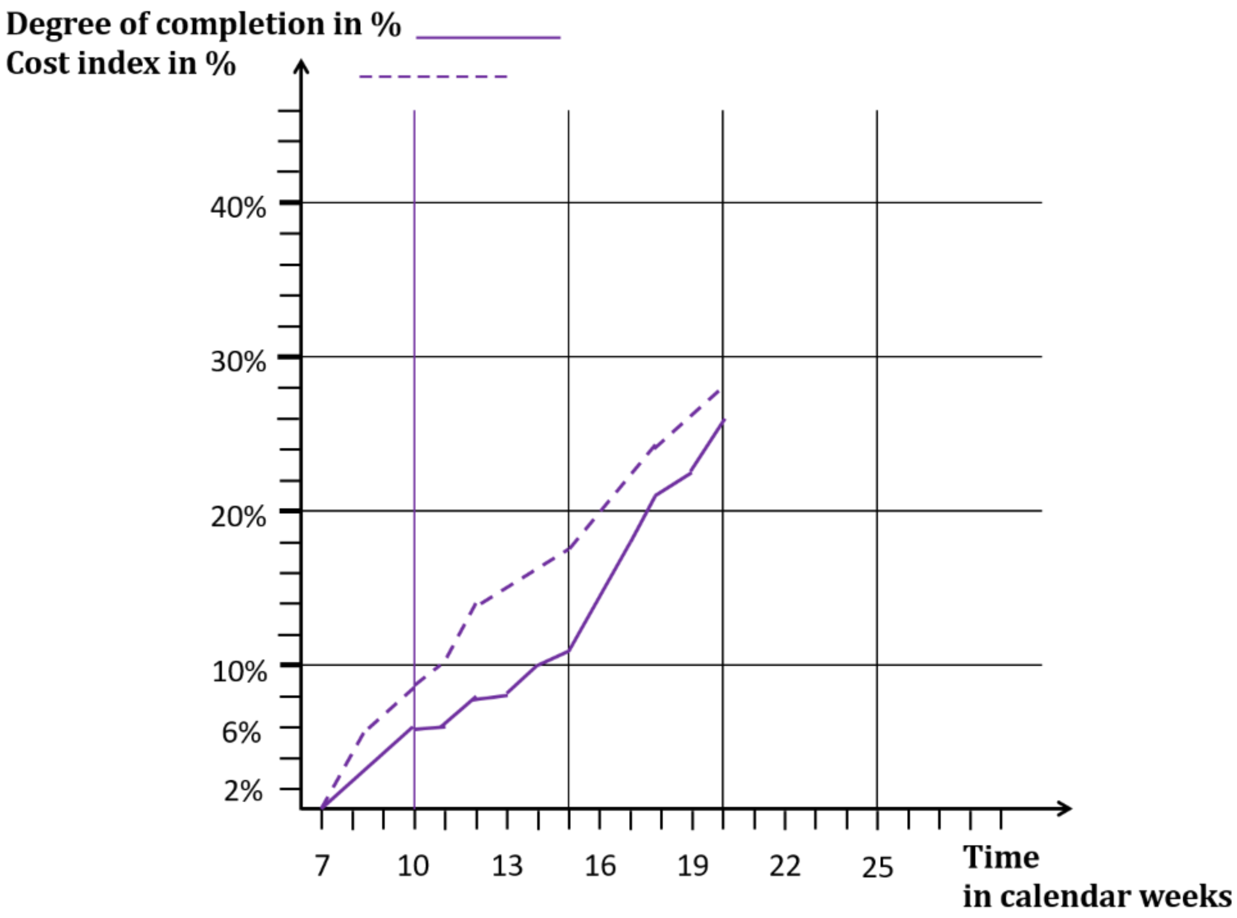


Figure 41: Time diagram for the degree of completion and cost index of a project

Figure 42 shows the development of the status of the requirements over time. This type of graphic can be created very easily in a spreadsheet program if the data is available in tabular form. There is a lot of information in this chart: from the total number of requirements (= overall height of the bar), we can see that in this project, a lot of requirements were elicited in the first weeks, and only a few new requirements were added later. There is therefore very little "requirements creep" (= a creeping increase in the scope of the project). Agreement of the requirements began in calendar week 12 and the required decisions were then taken quickly within a few weeks. Overall, this diagram shows a very satisfactory project progression.

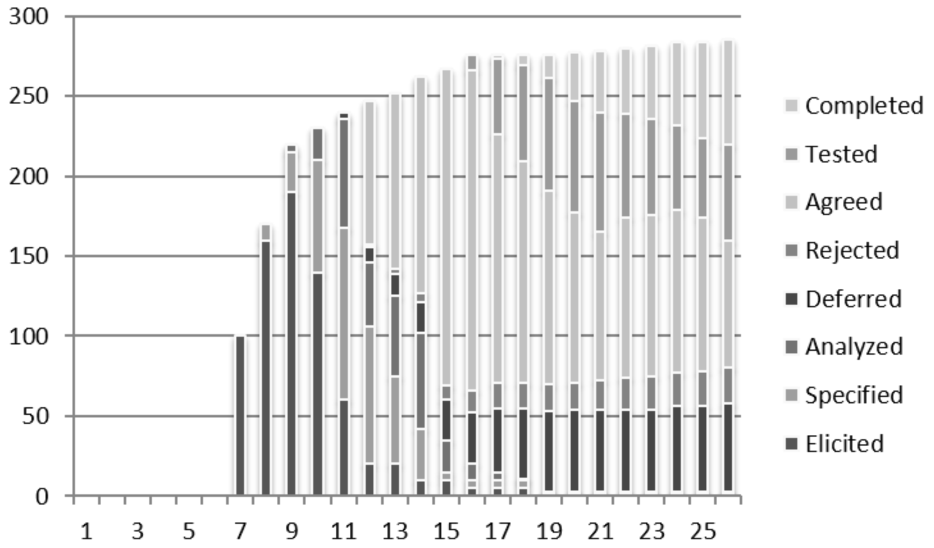


Figure 42: Time diagram for the status of the requirements. The horizontal x axis shows the time in calendar weeks, and the vertical y axis shows the number of requirements. It would also be feasible to show the effort on the y axis, that is, weight every requirement with its estimated effort.

Contents of the status reports

As already stated, our case study is to have two reports: the status report to Controlling, the department head, and the project team, and an abbreviated form for the newsletter to the remaining project stakeholders.

The status report is to be created weekly, and, just before the delivery deadline for the release, daily if necessary. The most important project variables are:

- Overall status of the release: this is yellow as soon as one of the requirements probably cannot be delivered; it is set to red if indispensable requirements cannot be delivered and the go live deadline therefore has to be postponed
- Costs already consumed in the release in € and cost index in %
- Degree of completion with respect to the next planned release in %, also compared with the planned degree of completion
- Status of the requirements, presented as a bar chart over time, as shown in Figure 42
- Number of open errors as a measure for the quality, applied over time
- Forecasts about delivery reliability and the probable delivery deadline

In addition, the report of course also contains the organizational information, such as the project name, date, version number, reporting period, creator, and recipients.

In addition to the organizational data, the newsletter contains the status of the release, the planned release deadline, the degree of completion compared with the planned degree of completion, and the forecast for the delivery deadline.

Practical tip: Less is more! Keep the number of reports to be created and their contents as low as possible. Every time a new field is created, this creates work for the author and the recipients. And too much unnecessary information can even cloud the view of the essentials.

8.2.3 Tips for Developing and Applying Reporting

There are some practical challenges in the development and application of (requirements-based) reporting:

- *Focusing on the essentials*: Even when the stakeholders and the benefits of reporting are known, the art is to focus on the essentials. The report definition process in Section 8.2.4 and the GQM method, which we describe in Section 8.2.5, help here.
- *Reconciliation*: The information required for the report must be provided in the requirements information model (see Chapter 2) and the attribute schema (see Chapter 3). As it is difficult to retrospectively change the information model and attribute schema, and the introduction of a new attribute requires extensive content maintenance, the requirements management data models should be clarified at an early stage, even before the project or work begins. It is helpful to use reference models that have already been coordinated with each other.
- *Data collection*: The people who have to collect the data are not the same people who need the information and create or read the report. The data collectors therefore have no inherent motivation to enter the data. It is therefore even more important that data collection is integrated into daily work processes well and that it is clear who has to enter which data and when.
- *Data quality*: The mere presence of attributes does not necessarily mean that all content is maintained, up to date, and correct. While it does not make sense for an efficient work process to introduce too many mandatory fields, especially since some information is not yet available when a requirement is created, for reporting, it would be important that the attributes are maintained. Missing content leads to incomplete information in the reports. In Section 8.2.7, we describe how you can ensure the data quality or consider missing data in the report.

8.2.4 The Report Definition Process

Defining requirements-based reports requires a comparison with the attribute schema (see Chapter 3). The requirements manager is responsible for this task. If the requirements manager cannot conduct the comparison themselves, they delegate this task to a suitable person.

According to ISO 15288 ([ISO 15288], 6.3.7.3 a) 1) to 4)), a measurement and reporting system is defined in these steps:

1) Description of characteristics of the organization relevant to the measurement
2) Identification and prioritization of information needs
3) Selection and documentation of key figures that meet these information needs
4) Definition of procedures for data collection, analysis, and reporting

In the following sections we split these four steps up further.

Characteristics of the Organization

Characteristics of the organization that are relevant for reporting are, in particular, the organizational chart and the requirements information model, including the attribute schema. The characteristics of the project are also relevant here: scope, schedule, stakeholders.

Identifying Information Needs: Identification of Report Recipients

All stakeholders of the project are potential report recipients. However, report recipients can also be people who are not (yet) on the stakeholder list.

Identifying Information Needs: Determining Goals and/or Risks in the Project

The report recipients want to use the report to achieve their information goals. These goals must now be determined, at best by the report recipients themselves. The goals are often to reduce project risks or to learn about the occurrence of the risk in time to initiate actions.

Examples of goals or risks in product development are:

- The planned delivery time or milestone deadlines must be adhered to (goal).
- The budget must be adhered to (goal).
- Important required functions cannot be provided (risk).
- The budget for eliciting the different stakeholder requirements will not be sufficient (risk).
- The required quality cannot be delivered (risk).

Prioritizing Information Needs

The main goal of the project is still product development and not reporting. Therefore, the report must focus on the most important needs of the most important report recipients.

It is easy to imagine creating a tailor-made, optimized report for the most important report recipients and sending less important report recipients the same report or an extract thereof, even if this is not optimal and only just sufficient for their goals.

When prioritizing the report recipients and their information needs, important criteria include the position of the report recipient in the hierarchy and the criticality of the success of the information need—that is, how important it is for project success that this information need is met.

Selecting Key Figures, Defining Report Content

In Section 8.2.2.1 and Section 8.2.2.2, we proposed some key figures and report content that are recommended in literature and often collected. These can serve as examples and an (incomplete) checklist. However, the content that should actually be included in a specific report depends on the information need and other factors. The GQM method (see Section 8.2.5) describes how you get from the information need to the key figure that supports it.

There are two important criteria for selecting the report content and key figures:

1) The information needs are fulfilled.
2) The data is easily available.

With regard to data availability, ISO 29148 ([ISO29148]) advises focusing on those data and key figures that are collected anyway, and to use these as a checklist for the data to be collected: "*It is good practice to choose measures for which data are readily available through the life cycle. The data collection can then be integrated into the requirements related processes to obtain the data and insight on a regular basis as the requirements engineering proceeds. It is also good practice to review the analyzed requirements related measures collectively, looking for predictive trends and projections that can aid risk management.*"

Both criteria (information need and data availability) must be weighed up against one another. The report must contain only data that meets an existing information need. Of course, it does not make sense to include data in a report simply because it is easily available if nobody actually needs this data. Conversely, it can be useful not to include key figures in a report, even though they would be useful, if collecting them is difficult and requires a disproportionate amount of effort.

When selecting the data to be reported, start with the data that you already have and check whether it fulfills any information needs. Then check whether every information need is fulfilled and if not, which data may have to be collected in addition.

It is feasible that in different project phases, different information needs exist or different data is or should be available.

Defining Procedures for Data Collection, Analysis, and Reporting

Data collection should ideally take place during normal project work, should be integrated in the normal work processes, and should not cause any additional effort. We will look at the procedure for collecting data later on in Section 8.2.6.

For the procedure for data analysis and reporting, you have to clarify the following:

- **Generation cycle**: How often must the report be created? There may be specific points in time at which the status of the project must be determined, for example, the milestones.
- **Tools**: What tools are used to create the report?
- **Report creator**: Who creates the report? In principle, multiple persons involved in the requirements management process can provide content for a report, in the same way that multiple persons involved in the process can receive reports.
- **Report form**: In what form will the report be created (format and template) and how will it be distributed?

Reporting can be implemented manually by a defined report creator, by the requirements manager, or automatically by a requirements management tool. The type of implementation depends on the maturity of the tool environment, the extent to which requirements management has been established in the company, and on the importance of reporting in the company. The automatic generation of reports has a special role particularly for extensive reports and reports that have to be generated regularly and for extensive data. This is because automatic generation significantly reduces the effort involved in generating the reports and the probability of errors in the reports.

As well as being influenced by the tools, the presentation of the report is also influenced by the standards, customs, and expectations that prevail in the respective company.

8.2.5 Goal, Question, Metric Method

In reporting, the "goal, question, metric" (GQM) method [BaWe1984], [Basi1992], [BCR] is one potential method for ensuring that no unnecessary, or, to be more precise, only goal-oriented key figures are defined for reports or report content. GQM is a systematic procedure for identifying such key figures. A suitable key figure is identified by answering the following questions:

- Which goal is to be achieved by the measurement? (Goal)
- What should be measured and which questions should the measurement answer? (Question)
- Which key figure(s) can describe the necessary characteristics? (Metric)

When applying the GQM method to reporting, we start with the report recipients and their information need (= goal). Which question should the report answer and which key figure is suitable for this?

GQM for delivery reliability

Goal: In our example project, we are particularly interested in the delivery reliability.

Question: When will the new release go live?

Metric: The probable delivery deadline is the key figure that is particularly relevant here. It is determined via the earned value analysis, which in turn requires the collection of multiple further key figures.

It is also feasible that one information need leads to multiple questions, or that multiple key figures are needed to answer one question.

If, for example, the goal is a high level of customer satisfaction, customer satisfaction arises not through one single factor, but probably through a mix of hard and soft factors. Figure 43 shows a more complex example in which the key figures initially derived are in turn interpreted as a goal and analyzed further.

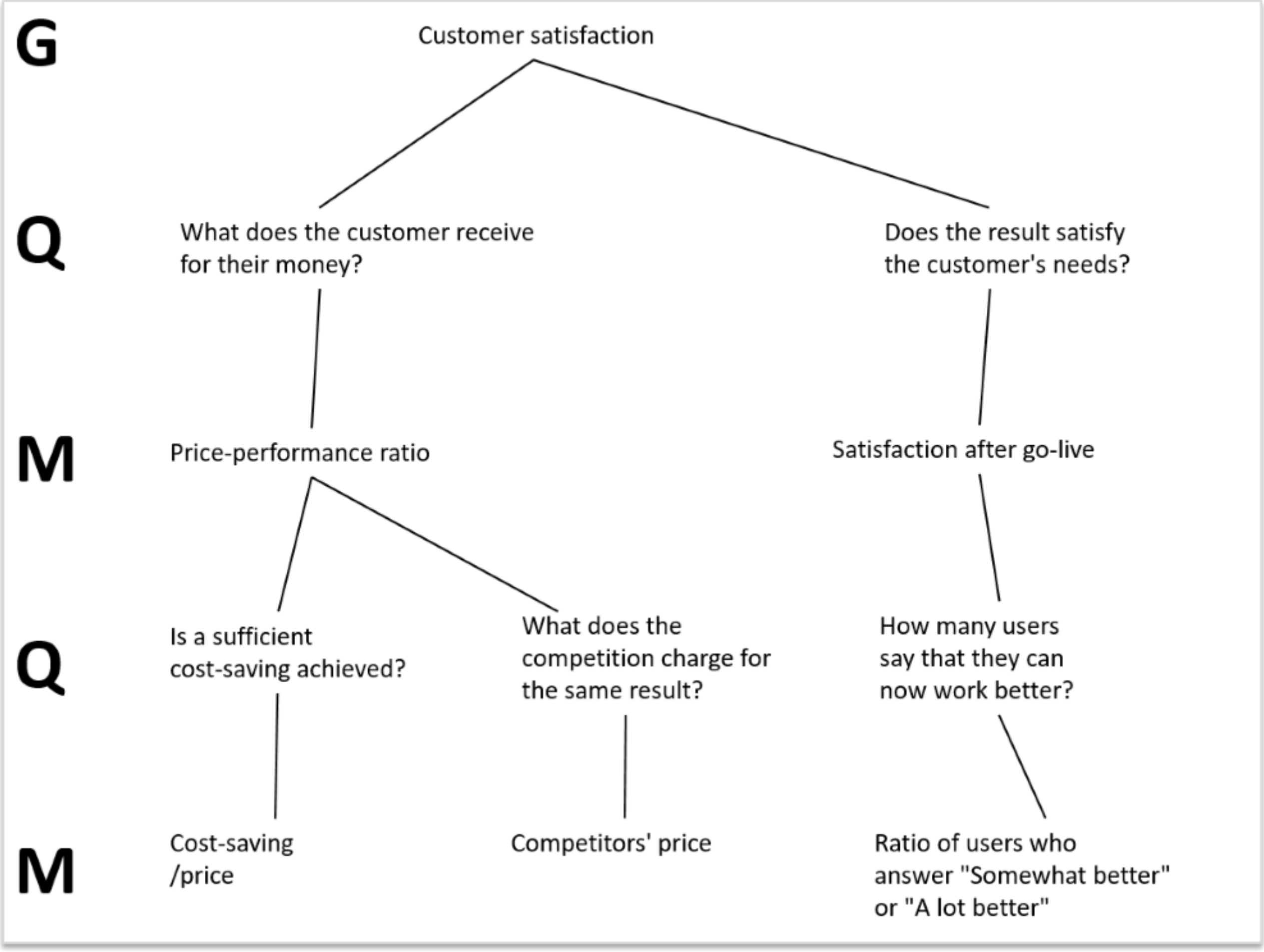

G
Customer satisfaction
Q
What does the customer receive for their money?
Does the result satisfy the customer's needs?
M
Price-performance ratio
Satisfaction after go-live
Q
Is a sufficient cost-saving achieved?
What does the competition charge for the same result?
How many users say that they can now work better?
M
Cost-saving /price
Competitors' price
Ratio of users who answer "Somewhat better" or "A lot better"

Figure 43: Example for the application of the GQM method

8.2.6 Data Collection

The collection of data for reporting covers the following tasks ([ISO15288], 6.3.7.3 b)):

1) Integrate procedures for data creation, collection, analysis, and reporting into the relevant processes: the description of the work processes must include a definition of who collects which data and when for the report. This applies, amongst other things, for the requirements engineering process (see Chapter 9). The earliest point in time for data collection is of course when the data arises—for example, effort estimations or actual efforts. It may, however, also be the case that at certain points in time, data is imported or aggregated from another system because it already exists there. The time for data analysis and reporting must also be defined to ensure that the stakeholders receive up-to-date and correct information regularly. The check of the data quality, which, for requirements-based data, is the responsibility of the requirements manager, must also be planned. Here, the requirements manager checks the data for completeness, plausibility, and quality.

2) Collect, save, and check data: as planned under point (1), the data is then collected and the quality of the data is checked. If applicable, the requirements manager can or should ensure that this actually happens. If the quality is not correct, the requirements manager ensures that the knowledge owners enter their knowledge in the requirements management tool and thus make it available for reporting.

3) Analyze data and create information: as planned under point (1), at the defined points in time, the views and reports are created.
4) Document results and communicate them to the users: the reports are documented (e.g., stored as a file with a timestamp) and distributed to the stakeholders concerned. The documentation can be helpful if you want to track the progression of the project status and the level of knowledge at a later point in time.

8.2.7 Checking the Data Quality

The report recipient receives the available information in the form of reports which they can then use to derive actions and/or decisions. The data quality should therefore be correct, because otherwise, incorrect data quality could result in incorrect decisions being taken. The persons involved should exchange information to increase the quality of reporting. Both the report creator and the report recipient are responsible for this information exchange.

"Information and reporting should not take place exclusively from employees to the project manager. The project manager should also make his knowledge and his information available to the employees involved. Lack of information leads not only to uncoordinated activities, but also has a negative effect on employee motivation" [KuSt2001].

The requirements manager is responsible for checking the data quality. Two criteria must be investigated: completeness and quality.

It is relatively easy to check the completeness of the data. For example, if all the attributes with the name "Effort" have been entered, this data is complete. If filtering or sorting by this attribute leads to the discovery of requirements for which this attribute is empty, the data is incomplete. It may of course be correct for individual items of data to be missing. For example, a requirement that is still being clarified cannot contain a value in the field "Actual effort" because no effort has been spent yet. Alternatively, when determining the degree of completion of the current release, the status of the requirements that have been deferred for later releases is completely irrelevant. Therefore, criteria must be defined for the completeness of the data.

It is more difficult to evaluate the quality of the content. Criteria must also be defined for this. Sometimes, these criteria are already in the attribute schema which, for example, can prohibit implausible or contradictory attribute combinations (see Chapter 3). In this case, it is not possible to enter implausible data. However, it is sometimes not technically feasible to prevent such attribute combinations on collection. You then have to identify them using suitable views.

If instances of missing or implausible data are found, the question arises as to who should or can correct this data and by when it must be corrected. In principle, the attribute owner is responsible for either maintaining the content of the attributes themselves, or for ensuring that someone else does so. The urgency depends on the urgency of the information need. A list of data can be entered retrospectively quite quickly; alternatively, the instruction can be given that the next time a requirement is edited, the data is to be corrected.

8.3 The Risks and Problems of Using Reporting

In practice, there are practical difficulties in gathering and evaluating data that result in reports not adequately reflecting reality. As reports are intended to lead to important management decisions, an incomplete or even deliberately embellished report can have far-reaching consequences.

Evaluation of Data: Condensed Representation of Reality

A report is always a highly condensed model of reality in which similar things are grouped into categories and insignificant details are omitted. It is very difficult to do this in such a way that any future question can be answered well at any time.

This is why the superficiality of a report must always be taken into account. In particular, it is important to avoid drawing false conclusions from the data available. For example, a report that shows 99% traceability for all project requirements does not yet allow a statement on the progress of the project or the quality of the relationships. Requirements that have not yet been linked could be the most important or most time-consuming requirements that contribute significantly to the success of a project. When reducing the complexity of key figures, you should always be aware of this problem. It is often the case that only very rough statements and conclusions are possible.

If really reliable data is required, the question must be asked correctly and the correct key figure evaluated. The GQM method (see Section 8.2.5) can support the derivation of the correct key figure. Further data may have to be collected.

Data Quality

Missing data is usually easy to detect. It is not as easy to evaluate the quality of the data: does the data correspond to reality? Is it up to date? Does it measure exactly what it should, for example, does the attribute "Effort" only measure the implementation effort, although the test effort should also be taken into account? Is the criticality actually the result of an expert survey or has it been set provisionally?

Undiscovered but also known shortcomings in data quality lead to the report not correctly reflecting the reality of the project. It is difficult to make the right management decisions based on this incorrect data. And even if the lack of data quality is known, decisions are difficult to make.

Poor data quality often results from the fact that the parties involved neglect data maintenance because they themselves have little benefit from it. Conversely, sometimes they may even be interested in embellishing the data, or at least in saving time on data maintenance by not performing careful analyses and instead hastily entering data that seems plausible.

However, poor data quality can also result from the fact that not everyone involved has the same vision. In agile development (see Chapter 10), the "definition of done" is an important topic of discussion. The point at which a requirement is considered completed must be clearly defined. Possible criteria for the implementation of a requirement include the following: the code has been created, unit tests have been created and successfully run, the documentation has been adapted and the code convention followed.

Quality defects are difficult to detect if the data has been intentionally poorly maintained. The person maintaining the data ensures that even if the data is not correct, it is plausible.

The data collection and data analysis processes should therefore include steps for quality assurance. Diverse plausibility checks are feasible in which different data is compared. With the earned value analysis, where the project progress and budget consumption are compared with one another, data collection problems can also be identified in addition to real project problems. Therefore, if the project progress and budget consumption do not match, the first step would be to check that the data is correct. Other data can also be cross-checked accordingly—for example, the status of the requirements can be compared with the date of the elicitation of the requirements. If a requirement was elicited a long time ago but still has an early status, then discussion of this requirement has been forgotten or it is simply the case that the status has not been updated according to the processing status.

Evaluating the Performance of Specific Employees

In Germany, employees and their data enjoy legal protection. In particular, measurement of the performance of individual employees must be avoided. If, despite this, a report is still to be created—for example, to collect information about employees' workloads so that work can be redistributed if necessary—any such report must be agreed with the Works Council.

It is therefore better to collect data on a requirement, project, or team basis. Personal data should only be collected when this is absolutely necessary. This is not usually the case.

Data Protection Regulations

Applicable general and company-specific data protection regulations must be followed when defining and implementing the reporting system. If personal data is provided by participants and further communicated within the company in the form of reports without the knowledge of the participants, this can lead to problems. In this context, it is important to clearly agree with the data creators who receives which data within the scope of the decisions to be made. In general, personal and person-related data should be used sparingly or should not be entered in the first place. When defining views, you should also ensure that no statements about individual persons can be made so as not to unintentionally violate data protection regulations.

Inflationary Reporting

If the volume of report information increases constantly, this might also lead to a situation where the report recipients are unable to process this data due to time constraints and important decisions can no longer be made on a sound basis.

Therefore, less is more! Focus on the information that is really necessary. This can also mean that different target groups receive different reports in which only certain aspects are presented, or are presented at various levels of detail.

8.4 Content for the Requirements Management Plan

The requirements management plan defines which (requirements-based) reports are to be created and when they are to be created. For each report, the report recipient and the goal of the report are documented, for example in tabular form. The derivation of report content from goals can be represented graphically as a goal, question, metric tree (as shown in Figure 43). The requirements management plan also defines what content the report contains and how this content can be determined or calculated from which attributes, and how the content is presented (e.g., the graphical form of presentation). The specification can also be documented in the form of a report template or view.

8.5 Literature for Further Reading

[DeMa1982] Tom DeMarco: Controlling Software Projects: Management, Measurement, and Estimation. Prentice Hall/Yourdon Press, 1982.

Earned value analysis for beginners:

[Wann2013a] Roland Wanner: Earned Value Management: The Most Important Methods and Tools for an Effective Project Control. CreateSpace Independent Publishing Platform, 2013.

Earned value analysis for experts:

[Wann2013b] Roland Wanner: Earned Value Management: So machen Sie Ihr Projektcontrolling noch effektiver. CreateSpace Independent Publishing Platform, 3rd edition, 2013 (available in German only).

9 Managing Requirements Engineering Processes

9.1 Requirements Engineering as a Process

A process consists of interdependent activities which each transform input into output [ISO9000]. Each activity is uniquely assigned to the organizational entity responsible for it, for example, a role. As part of the development process, requirements engineering and requirements management can be seen as processes.

The requirements engineering process is understood as a systematic process for developing requirements via an iterative, cooperative process of eliciting, documenting, validating, negotiating, and managing requirements (according to [LoKa1995]).

This requirements engineering process includes the following four types of activities [IREB2015]:

- Eliciting requirements
- Documenting requirements
- Validating and negotiating requirements
- Managing requirements

In each project, there are several elicitation activities such as workshops and meetings with stakeholders, document analysis, analysis of the legacy system, and so on. There are also multiple individual activities for the other types of requirements engineering activities.

The requirements engineering process uses stakeholders' needs and ideas as input information. In addition, the status quo before the project start (e.g., the legacy system) and competing products also play a role. The result of the requirements engineering process is a validated, conflict-free, consistent, prioritized, quality-assured requirements specification that can serve as a reliable basis for further project work.

In general, the four activity types (whose procedure and methods are defined in the CPRE Foundation Level [IREB2015]) have the input and results outlined below, which can of course look different, especially if company-specific requirements or standards have to be met, or according to the given constraints (see Table 8).

These four types of activities must always be performed, regardless of whether they are explicitly documented or implicit. They do not have to be and, in fact, cannot be performed sequentially; instead, they can run iteratively, incrementally, or in parallel. Requirements are always elicited in some way, even if this is through informal discussions. There is usually also documentation—in the worst case, chronological documentation or documentation spread over numerous discussion notes. Documenting requirements implicitly only would of course not conform to the recommendations of the IREB. Standards and company guidelines require different implementation of these activities and define different guidelines with regard to the documents to be created.

Activity Type	Input	Result
Eliciting requirements	Stakeholders and their needs and ideas If applicable: an existing legacy system and its documentation; competitor products	Oral and written requirements including the system vision
Documenting requirements	Oral and written requirements	Written requirements specification (textual or model-based or both)
Validating and negotiating requirements	Written requirements specification	A validated, conflict-free, consistent, prioritized, quality-assured requirements specification
Managing requirements	Written requirements specification + change requests	A constantly up-to-date, validated, conflict-free, consistent, prioritized, quality-assured requirements specification Preparation of requirements for individual stakeholder groups

Table 8: Four activity types for requirements engineering, as well as their input and output (result)

The results of the requirements engineering process must satisfy quality criteria in three independent dimensions: specification, representation, and agreement [Pohl1994]. Requirements should become more mature over time within these dimensions, although there does not have to be a simultaneous, constant increase in all dimensions. For example, growth in the specification dimension (e.g., formalization) can lead to a regression in the agreement dimension because new contradictions have come to light due to the formalization.

- **Specification**: This dimension describes the completeness of the specification or the completeness of the understanding of the requirements. At the beginning of the requirements engineering process, requirements are vague and unclear (opaque). As the process progresses, requirements become more complete in the sense of a thorough coverage of the problem to be solved and a description that is detailed enough to be properly understood. Various standards provide guidelines as to which conditions must be met by the requirements in order for them to be considered complete. However, it is not possible to prove the completeness of requirements.

- **Presentation**: Here, the scale varies from informal to formal. Informal presentation includes sketches, free text, and prototypes. Semi-formal presentation includes graphical models such as class diagrams, state machines, use case diagrams, or data flow diagrams. Use cases presented in tabular form, which strictly follow a given syntactic structure, are also semi-formal. Formal specifications describe requirements completely uniquely using logic languages and formal semantics. Preparation of a formal specification usually begins with informal forms of presentation.

- **Agreement**: Establishing agreement is another goal during the requirements engineering process. In the agreement dimension, you move from the personal view to a common view of the requirements.

The requirements specification has to be optimized in all three dimensions during the requirements engineering process. Here, elicitation activities mainly contribute to improvement in the specification dimension, documentation activities to improvement in the presentation dimension, and validation and negotiation activities to improvement in the agreement dimension. Requirements management aims to maintain the quality level in all three dimensions, even when changes occur.

Various standards (see Section 1.5) propose how the requirements engineering process or the development process can be designed. However, these are merely blueprints that have to be adapted to the circumstances in the respective company. Through tailoring, process parameters, roles, activities, and result types can be adapted to specific needs.

Specifications for the requirements engineering process in the example bank

As a certified CPRE professional, Peter Reber naturally complies with the IREB standard. However, this standard does not specify how the requirements engineering process is to be performed in detail. In fact, quite the opposite is true: this standard shows the wide range of selection options and supports tailoring of the process.

The four activity types are mandatory:

1. Eliciting requirements
2. Documenting requirements
3. Validating and negotiating requirements
4. Managing requirements

The methods that can be used to perform these activities and the criteria for selecting the correct method are described in the CPRE Foundation Level and the respective Advanced Level. We have already defined the requirements landscape for the case study project in this book.

What still has to be defined is who performs the planned activities how and in what order. We discuss these parameters of the requirements engineering process in the following Section 9.2.

9.2 Parameters of the Requirements Engineering Process

Even if the same elicitation, agreement, and documentation methods are used, the requirements engineering process can vary greatly and must be adapted in particular to the given constraints, such as the project size and the skills of the persons involved. Despite the variety of requirements engineering processes that exist in different process models, there are only a certain number of process parameters that can be changed when selecting or adjusting the requirements engineering process:

- Timing of the elicitation (upfront or iterative)
- Level of detail of the documentation, that is, the lowest level of detail used for the specification (heavyweight versus lightweight specification)
- Incorporation of changes, in particular: change request versus product backlog
- Allocation of responsibility

These parameters should be adjusted to the given constraints. Such constraints are:

- The size of the project
- Is it a new implementation or a small enhancement, improvement, or variation to an existing, mature system or product?
- Is the system security-critical?
- Was a fixed price agreed or not?
- Is there a stable team that has been working together for years?
- Availability of people and their qualifications

9.2.1 Timing of the Elicitation (Upfront or Iterative)

Requirements can either be elicited completely at the beginning of the project (upfront) or iteratively (iterative requirements engineering): in the first case (*upfront*), a requirements specification (e.g., a customer requirements specification) is created at the beginning of the project, describing the planned project scope in its entirety, at least at the uppermost level of detail of the requirements. With iterative requirements engineering, the aim is not to define the requirements, or even just the project scope, completely at the beginning, but rather to consider the requirements documentation (e.g., the product backlog) as a preliminary list. Requirements can be added or changed at any time, even during implementation. Caution: there is a difference between iterative requirements engineering and iterative development. It is therefore conceivable to first create a complete requirements specification upfront and subsequently implement the requirements through iterative development.

If the project is a small enhancement, improvement, or variation of an existing, mature system or product, or a small project, then it is to be expected that stable requirements can be defined for the entire project with few surprises expected. This is where upfront requirements elicitation is possible and useful.

However, if the project is very innovative with many uncertainties, is a large project, is in a volatile environment, has changing, undecided or conflicting stakeholders, or there are other risk factors that make a reliable upfront specification impossible, iterative requirements engineering serves to reduce risk. However, iterative requirements engineering requires regular participation of at least the most important stakeholders. If this is not possible, and the only opportunity is a one-time elicitation workshop or an initial elicitation phase, then the requirements must be elicited upfront. An upfront specification is also required if the specification has to be created under more difficult conditions, if the project has a fixed price (meaning that the project scope has to be defined early), the system is a security-critical system for which a security analysis has to be performed in the overall view, or a technology is used that is difficult to change and enhance, that is, it is difficult to consider requirements that arise spontaneously.

9.2.2 Level of Detail of Requirements Documentation

The level of detail of the documentation or specification can vary between heavyweight and lightweight requirements: the heavyweight specification describes all requirements in detail at multiple levels of detail, including all their attributes and traceability relationships. This makes the specification very comprehensive. In agile development, it is common to create lightweight specifications with only a few levels of detail. In this case, requirements are specified only as comprehensively as necessary and not earlier than necessary. The point in time at which certain information is required depends on the process model. What is needed depends on the stakeholders, their needs, and background. A project-specific stakeholder analysis helps to define how detailed the requirements specification must be. Among other things, the purpose of a specification is to enable the developer to understand what stakeholders want. With a lightweight specification, details of the implementation are left to the developer (especially if he is very familiar with the domain), are discussed verbally without being documented, or are refined using a prototype. The lightweight requirements specification describes requirements as user stories, for example. Requirements are only specified in detail when their implementation is about to begin. Even though upfront specification is usually heavyweight (e.g., in the waterfall model and V-Modell XT) and iterative specification is usually lightweight (as in scrum and other agile methods, see Chapter 10), the two parameters timing and level of detail are independent of one another. It is possible to create both a lightweight specification upfront and a heavyweight one iteratively (as in the Rational Unified Process).

Whether the specification is lightweight or heavyweight depends less on the project size or the type of contract, and above all, more on the information and documentation need of the specific project and its stakeholders.

Due to statutory requirements alone, security-critical systems are usually specified in heavyweight form and completely. In principle, a lightweight specification saves unnecessary effort if the information that would have been documented additionally in the heavyweight specification is available to all stakeholders in an undocumented form—for example, in the case of small teams or teams that have been working together for a long time, developers with very good knowledge of the domain and the customer, and for the further development of an existing system.

However, the lightweight method can also be used if it is technically easy to create and change a prototype quickly and the requirements are refined based on the prototype.

The level of detail is defined in the requirements information model (Chapter 2).

9.2.3 Change Management: Incorporation of Changes (Change Request versus Product Backlog)

Requirements change during a project. Some requirements engineering processes integrate new or changed requirements into the requirements specification and development process as change requests. The projects concerned are usually projects with a fixed price and upfront requirements specification, which means that from an organizational and legal point of view, the definition of the project scope and the requirements elicitation are completed at a certain point in time. From a legal perspective, subsequent changes are contractual changes. In legal terms, a change request means a new contract. Normally, the project's contract already specifies how change requests are to be handled. They usually go through a simplified approval procedure with the following steps: analysis (of the requirements and their benefits), impact analysis (i.e., analysis of changes to the system, their costs and risks), decision by the Change Control Board, and then implementation. A change request is often described using a change request template that assigns a unique number and title to the change request, describes the problem to be solved and the proposed solution, quantifies costs, benefits, and risks, and manages the status (requested, accepted, rejected, postponed, implemented) (see Chapter 5).

In iterative requirements engineering, however, requirements are collected in the product backlog and all requirements—old and new—are treated equally. This is made possible by the fact that there is never a commitment to a defined system scope. The advantage of this procedure lies in the flexibility. New, important requirements can be integrated into the project easily. However, this flexibility also has the disadvantage that it is ultimately difficult to define the exact delivery scope. The number of requirements in the product backlog can increase constantly; in the worst case, more quickly than the requirements are implemented.

Nevertheless, it is not mandatory for an upfront requirements specification to treat later requirements as change requests. It would be conceivable to adjust the requirements specification created upfront later without recording and approving changes as change requests. Changes to the requirements artifacts must of course be documented and traceable.

Conversely, an approval for new requirements could also be demanded in iterative requirements engineering. The primary decisive factor here is the form of contract. In the case of a fixed price contract, a new requirement can only be integrated into the project if both contracting parties agree, which requires a more or less extensive approval process. In other cases, how changes are handled is a matter for agreement between the client and the contractor.

From the perspective of requirements management, it definitely makes sense to integrate the change requests into the existing requirements specification sooner or later so that an up-to-date specification of the planned system is available at any point in time.

There should therefore not be two separate specifications—for example, the requirements specification with the status at the start of the project plus a list of chronologically sorted change requests. This type of presentation does not fulfill the modifiability requirement for requirements specifications. The main question from the point of view of requirements engineering is therefore who integrates change requests into the specification, when do they do this, and in what form? Good coordination between project management and requirements management is definitely required. This allows requirements management to provide the decision makers with important information about the probable impact of a change (with regard to impact analyses, see Section 6.5.2 on the usage strategy for traceability).

9.2.4 Allocation of Responsibility

A single role (for example, the requirements manager) can be responsible for requirements engineering in the sense that this role plans, controls, and improves the requirements engineering process. The role either performs the activities involved in the requirements engineering process or ensures that they are performed by someone else. However, there can also be an entire team or several roles responsible for requirements engineering, either for different activities or different content (e.g., functional requirements versus usability requirements). Requirements engineering can also be closely integrated into the development process without a separate requirements engineering process or a requirements engineering role existing. In this case, the development team performs the requirements engineering activities—that is, the team members elicit, document, check, and manage requirements.

The bigger the project, the more it makes sense to define a separate requirements manager role to monitor this area of activity in the project. However, this role can also be defined for small projects, in which case it is not a full-time role. The requirements manager should be the person who is most familiar with requirements engineering and requirements management. In particular, this person must have very good communication skills and must also be in constant contact with all stakeholders. In addition to methodological requirements engineering and requirements management knowledge and technical knowledge, which the requirements engineer needs, the requirements manager also needs management skills to be able to set up, manage, and monitor the requirements engineering process.

Parameters of the requirements engineering process in the example bank

In our case study, the requirements are elicited upfront, as the project is a further development of software that the team knows well. It is therefore very feasible to elicit and describe the requirements at the beginning. In contrast, development will take place iteratively.

As defined in Chapter 2, the requirements are described at multiple levels of detail. This supports a detailed security analysis and a complete documentation of the system for the future.

Change management: the project is an in-house project with a fixed budget. However, as there is no fixed price contract, the content—if not the project scope—allows some flexibility in principle.

When new, important requirements arise, these should be included in the project and other, less important requirements deferred. This is particularly true for changes in law, which have to be considered at short notice. However, these changes to requirements must be checked properly and executed in a controlled manner.

This requires a documented change process in which, during the impact analysis, not only do the implementation costs have to be estimated, but the security team must also submit a risk assessment for the IT security, the business analysts must submit a risk assessment for the business processes, and the usability expert must submit a risk assessment with regard to accessibility. Based on the evaluations, a Change Control Board ultimately decides whether the change to the requirement is accepted, rejected, or deferred.

The responsibilities were defined in Chapter 1: as the requirements manager, Peter Reber plans and monitors the requirements engineering process, while multiple experts perform requirements engineering—that is, they elicit, document, and agree requirements. Several external business analysts analyze the business processes, a team of IT security experts conducts risk analyses, the usability expert designs alternative interface designs and improves accessibility, and a moderator holds an ideas workshop with the Customer Advisory Board.

9.3 Documenting the Requirements Engineering Process

The requirements engineering process consists of numerous activities of the four types mentioned above, such as elicitation workshops, document analyses, specification reviews, etc. Many of these activities are planned in the form of meetings or workshops, as they involve multiple persons. The order of these activities results from the selection of the process parameters, for example, whether requirements are elicited and specified upfront, or how changes to requirements are handled (see Section 9.2).

The following applies regardless of whether you are defining a generic requirements engineering process that is to apply as a company specification for all projects, or whether you are planning the requirements engineering process for a specific project.

The activities and their sequence can be presented as a UML activity diagram. The activity diagram can also show the assignment of activities to roles. You will be familiar with this notation from the CPRE Foundation Level ([PoRu2011].

The assignment of responsibilities for activities to roles can also be presented in more detail using a *RACI matrix* like the following. RACI stands for:

- R = responsible = responsible for the execution
- A = accountable = authorizes, for example, the activity and its budget
- C = consulted = (will be) consulted, especially in terms of technical, content-related responsibility
- I = informed = to be informed, i.e., the person is to be informed about the results

The following table shows an example of an excerpt from a RACI matrix.

Activity	Requirements Manager	Business Analyst	IT Security Expert	Usability Expert	Moderator	Customer Advisory Board
Analysis of business processes	A, I	R	I	C, I	I	
Risk analysis	A, I	C, I	R	C, I		
Interface design	A	C	C	R		
Ideas workshop	A, I	I	I	I	R	C
…						

Table 9: Example of a RACI matrix for requirements engineering

The requirements engineering process

Table 9 shows an excerpt from the RACI matrix for our case study. The activities presented here all belong to the activity type requirements elicitation. Elicitation of the requirements probably also includes further activities not specified in more detail here. There are also activities of the type requirements documentation, validation and negotiation of requirements, and requirements management. However, our goal here is not to plan the requirements engineering process for the entire project completely, but rather to illustrate the corresponding methods of presentation.

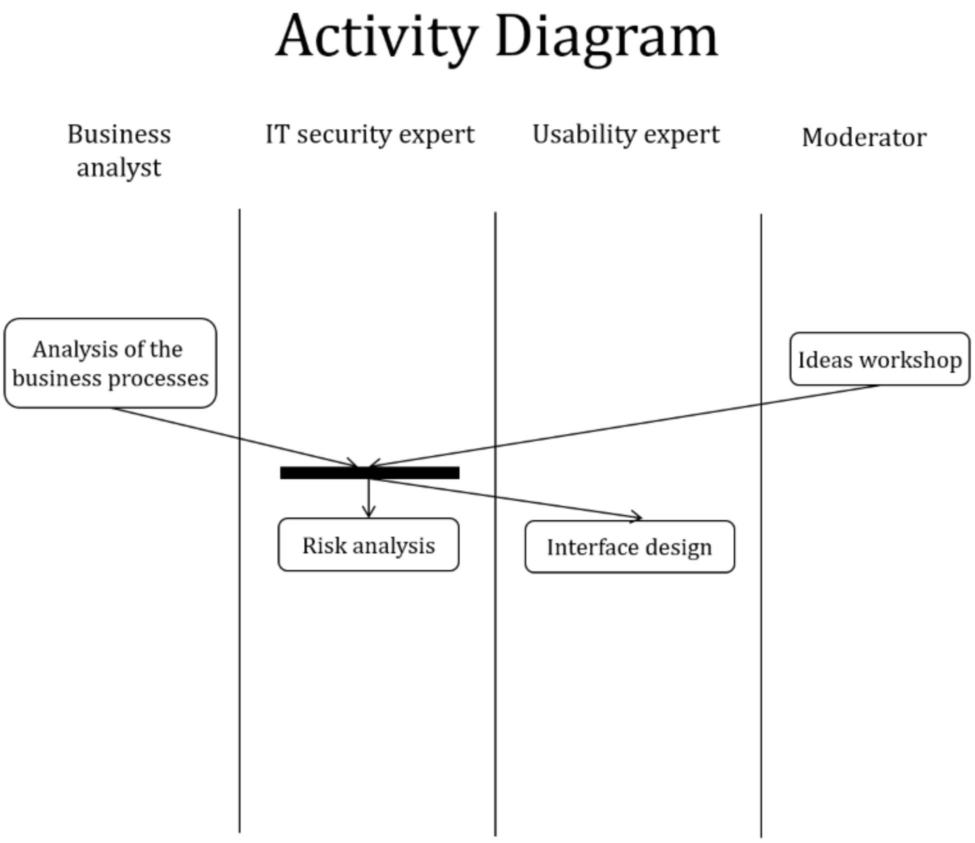


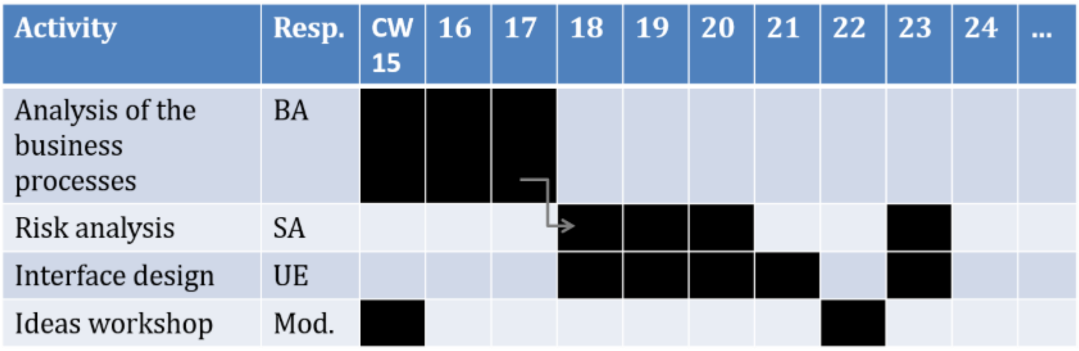
Figure 44: Activity diagram (excerpt) for the requirements engineering process for the case study

Figure 44 presents the same excerpt from the requirements engineering process as an activity diagram. How the two forms of notation differ and supplement each other is clear:

The RACI matrix can present the responsibility of the different roles for the activities in more detail, whereas in the activity diagram, an activity is usually only in one swim lane: in the lane that belongs to the role responsible. Anyone else involved in the activity is not shown.

In contrast, the activity diagram also documents dependencies between the activities, for example, the order of the activities, such as "The risk analysis takes place after the business process analysis (because it builds on the results of the business process analysis)", or "Risk analysis and interface design can take place in parallel".

The Gantt diagram in Figure 45 shows the time progression in even more detail. The person responsible is defined in the column "Resp.", and the other RACI responsibilities could also be presented here. For each activity, the time progression is shown horizontally as a row, with each calendar week (CW) in which work is performed on this activity shown in black. Thus, the duration and repeated work can be presented in more detail than in the activity diagram. In the Gantt diagram, however, the presentation of dependency relationships between the activities is not so clear, even though arrows are used to represent these.



Activity	Resp.	CW 15	16	17	18	19	20	21	22	23	24	...
Analysis of the business processes	BA	■	■	■								
Risk analysis	SA				■	■	■			■		
Interface design	UE				■	■	■	■		■		
Ideas workshop	Mod.	■							■			

Figure 45: Gantt diagram (example). BA stands for business analyst, SA for security analyst (or IT security expert), UE for usability expert, Mod. for moderator.

The three types of presentation therefore complement each other well. They can thus be used together. However, it is more efficient to concentrate on the least number of forms of presentation as possible. It is also feasible to use the Gantt diagram for the rough planning of the work packages, and to use one activity diagram for the fine planning of each work package.

To manage dates and budgets quantitatively, the requirements engineering process can also be presented as a project plan. Other documents that can represent and support the requirements engineering process are: checklists, templates, sample documents and guidelines for the execution of individual activities.

If many people are involved in the requirements engineering process, it also makes sense to support this process with a tool. All workflow management systems in the broadest sense are suitable for this.

9.4 Monitoring and Controlling the Requirements Engineering Process

Monitoring the requirements engineering process means ensuring that all activities are performed and the defined results are delivered on time and that the activities remain within budget. Reports that regularly record dates, budget consumed, status, and degree of completion of the requirements engineering process and its individual activities and compare the actual values with the target values from planning are helpful for this (see Chapter 8).

Controlling the requirements engineering process means executing it according to the plan or, if the process deviates from the plan, taking corrective action. For example, if it becomes apparent that a deadline or budget cannot be met, the consequences for the overall project must be determined and—if appropriate—countermeasures taken. There are two alternative options for correcting the situation: you can adjust the plan to the actual progress, or adjust the progress to the plan. The former is easier, but often difficult for a project with a binding end date and budget. To adjust the ongoing process to the plan, planned activities may have to be omitted, brought forward, or performed with less effort. Careful trade-offs must be made where they cause the least damage: for example, individual stakeholder groups are not interviewed, individual open questions are not clarified, details are not specified, unimportant change requests are rejected, and so on. The prerequisite for setting such a focus is, of course, that the requirements have been prioritized (see Chapter 4). It is important to consider the risk: does the benefit of the savings outweigh the possible damage?

Practical tip: If the defined requirements engineering process cannot be adhered to, there can be various causes. For example, the employees may not know the process or may not have understood it correctly. It may also be the case that the process does not describe the optimal working method and is therefore not adhered to. There may even be resistance to new or certain working methods that is causing the process not to be adhered to. As each of these causes requires a different measure to correct it, it is essential to find out why the process is not being put into practice.

9.5 Process Improvement for the Requirements Engineering Process

A process can always be improved further. The CMMI (*capability maturity model integration*) maturity model requires that a mature development process plans activities for continuous improvement of the work processes. The basis for this is an analysis of the actual process, referred to as an evaluation or audit, that systematically investigates how good the process currently is, where it is already good, and where there is potential for improvement. In an audit, the current process is usually compared to a reference process (e.g., prescribed by a standard) and process key figures are collected (see Section 8.2.2.1). The basis for a process analysis should always be objective, measurable criteria.

When performing an audit, you go through the following steps, for example:

1. Recognize the need for an audit
2. Plan the audit, for example, the purpose and goal, the scope (Process? Product? Which?), the team, the criteria, the resources, and deadlines
3. Perform the audit and document the results
4. Evaluate the results: strengths and weaknesses, required improvements, and the most urgent measures
5. Implement the measures
6. Measure the improvements

Process improvements can be performed either abruptly—a process rearrangement—or continuously. A process rearrangement changes many activities and parameters of the process at the same time. This has the advantage that it is possible to achieve a significant increase in efficiency, which, however, usually only occurs after all participants have become accustomed to the new process. However, there is also the risk that the new process will not prove its worth and will reduce efficiency. Resetting will then again involve great effort.

Continuous process improvement avoids this risk and leads to short-term (mostly small) improvements with little effort. According to the principle of continuous process improvement (CPI), processes are optimized gradually by repeating the following four activities (PDCA) of the Deming cycle [Demi1982] iteratively:

- *Plan*: The actual process and, in particular, the need for improvement are analyzed. Based on this, the desired process is planned and documented.
- *Do*: Improvement actions are developed and tested in a pilot project and accompanied by measurements.

- *Check*: A check determines whether the actions have brought about the desired improvement. The actual values are compared with the planned values.
- *Act*: Based on the results of the actual/plan comparison, improvement actions are introduced continuously or, if necessary, new actions are planned. The implementation of the actions is monitored and accompanied by measurements.

The actual and target process are characterized using measured quantities (see Chapter 8). Such measured quantities can be:

- The *proportion of the project budget* invested in requirements engineering. Both too much and too little can be questionable. Normally it is 10-30% of the project budget.
- The *number of requirements still to be implemented* (weighted according to expected effort) measures the work that is still to be completed up to the end of the project.
- *Number of requirements (or rather, number weighted by implementation effort) implemented per time unit.* Together with effort estimations for the requirements still to be implemented, forecasts can thus be made about the remaining duration of the project.
- *Change rate of the requirements*: A rate of 1-5% of the requirements per month (measured in effort) and, in the worst case, 30-50% over a project duration spanning multiple years is considered normal [Eber2012].

  Fewer changes may mean that no one is really interested in the requirements and stakeholders are not sufficiently involved. Too many changes are also an alarm signal: requirements are not yet stable, stakeholder groups may be too heterogeneous or in conflict, and it is still too early to implement the requirements.
- *Processing time of change requests* from order to implementation.

With the help of benchmarking it is possible to find out which figures make sense and are achievable as target values.

Improvement actions can either refer to the process parameters described in Section 9.2, or to how the individual activities are performed in detail, for example, the methods used.

Another possibility for process improvement is to analyze the errors made in requirements engineering—for example, errors found during the specification inspection, or errors delivered with the software that can be traced back to requirements engineering. You then ask about their causes and the causes of the causes. This gives ideas for improvement actions.

Maturity models such as the CMMI (*capability maturity model integration*), based on ISO 15504 [Kneu2007], [CKS2011], [CMMI], or ITIL for software maintenance [Beim2012], [Ebel2014] offer more concrete help for process improvement in requirements engineering (but not only there).

According to [Eber2012], a maturity model is a "model that reflects the process capability in defined categories, thus allowing a reliable and repeatable process evaluation. A maturity model makes demands of processes and does not prescribe any processes itself. It is therefore not a process model. Used to evaluate process maturity and process improvement, for both a company's own process and those used by the supplier."

Maturity models describe activities or practices that must be performed to reach a certain level of maturity. All other methods of process improvement can also be used to improve the requirements engineering process, such as TQM (Total Quality Management) [HuMa2011] and Six Sigma [Tava2012], [BWJ2013].

In particular, TQM consists of principles for achieving quality and economic action. Important factors here are, for example, customer orientation, process orientation, quality orientation, joint responsibility of all employees, continuous improvement, and rational decisions. In contrast, Six Sigma is a framework for improvement, whereby measurements and statistical analyses are performed to create products that are free of errors in a process that is free of errors.

In particular, improving requirements engineering is supported by the collection of best practices from Sommerville and Sawyer [SoSa1997], who differentiate between three categories of best practices: *basic*, *intermediate*, and *advanced*. The first step when improving the requirements engineering process is to implement all basic practices, then the intermediate practices, and finally, those practices classified as advanced.

The basic techniques include, for example, the definition of a standard template for the requirements specification or a checklist for inspecting this specification.

The template for an action plan from Karl Wiegers [Wieg2005] supports the concrete planning of process improvement. The template comprises the following content:

- Name of the improvement project
- Date
- Goals (of the improvement, expressed as business goals)
- Indicators of success (i.e., achievement of goals)
- Organizational influence of the change
- Participants (employees, their roles and time budgets)
- Measurement and reporting process (when will the progress of actions within this plan be monitored, by whom, and how)
- Dependencies, risks, and constraints
- Estimated completion date of all actions within this plan
- Actions (3-10 per plan) with identifier, person responsible, target date, purpose, description, deliverables, and resource requirements

When improving the requirements engineering process, note that it cannot be optimized on its own, but only in cooperation with other project activities such as project management, development, and testing. Changes in the requirements engineering process will also affect those people's work.

Practical tip: Every process should be as simple as possible and only as complex as necessary. The larger the number of binding guidelines you make, the more you restrict creativity and flexibility. The constraints change, and therefore processes must also be constantly adjusted. If a process does not change, it becomes obsolete.

Improving the requirements engineering process

The requirements engineering process in our case study has been newly set up and encompasses a number of stakeholders who have not worked together before. These are risk factors that make observation of the requirements engineering process particularly important.

In Section 8.2.2.1, we discussed which key figures the status report should contain and why these key figures are so important. This data can be used to submit an updated forecast for the delivery deadline on a weekly basis.

If delays or other difficulties occur over the course of the requirements engineering process, the causes are investigated and actions taken.

However, a proactive approach is also to be taken, and risks associated with delivery reliability identified. Delivery reliability is very important for the project as a whole, and therefore this also applies for the requirements engineering process. Furthermore, various activities build on one another and are therefore dependent on one another. Together with the project manager, Peter Reber therefore performs a critical path analysis in the network diagram of the requirements engineering process to find out which activities are particularly critical for meeting the final deadline. These activities are then to be monitored particularly closely. (We do not describe the network diagram technique in more detail here because it is a project management method. However, you can find more information about it in DIN 69900 [DIN69900] and in any project management book.) The risk analyses that are to be performed after every change to the requirements and for every requirement change are seen as particularly critical. To ensure that these are not delayed unnecessarily, the availability of multiple IT security experts is ensured. These experts plan a workshop for every Monday afternoon. If the workshop is not necessary, it can be canceled. This ensures that resources are regularly available for the risk analyses.

The persons involved in the requirements engineering process are also asked to give their opinion. The usability expert sees a risk for the quality of the results if the interface design is created without participation by the users. The expert would like the opportunity to get feedback from the users, who are represented here by the Customer Advisory Board. Therefore, an additional activity "User tests of the interfaces" is scheduled. This activity is performed by the usability expert and the moderator together and they consult the Customer Advisory Board. Table 10 shows the extended RACI matrix.

Activity	Requirements Manager	Business Analyst	IT Security Expert	Usability Expert	Moderator	Customer Advisory Board
Analysis of business processes	A, I	R	I	C, I	I	
Risk analysis	A, I	C, I	R	C, I		
Interface design	A	C	C	R		
User tests of the interfaces	A			C	R	C
Ideas workshop	A, I	I	I	I	R	C
…						

Table 10: Example of a RACI matrix for requirements engineering

9.6 Content for the Requirements Management Plan

The requirements management plan documents the requirements engineering process in one of the notations described above. It also specifies whether the requirements are elicited upfront or iteratively, how changes to requirements are to be incorporated, and the responsibilities for the requirements engineering activities. The level of detail is defined by the requirements information model.

It should also be clear how the requirements engineering process is monitored (e.g., the report used). Actions for evaluating and improving processes should also be planned, for example *Lessons Learned* analyses after the end of the project.

9.7 Literature for Further Reading

[BWJ2013] Franz J. Brunner, Johann Wappis, Berndt Jung: Null-Fehler-Management: Umsetzung von Six Sigma, Carl Hanser Verlag GmbH & Co. KG; edition: 4, revised and extended edition, 2013 (available in German only).

[DIN69900] DIN 69900 Project management – Project network techniques; Descriptions and Concepts, 2009.

[HuMa2011] Thomas Hummel, Christian Malorny: Total Quality Management: Tipps für die Einführung, Carl Hanser Verlag GmbH & Co. KG; edition: 4, completely revised edition, 2011 (available in German only).

[Tava2012] Serkan Tavasli: Six Sigma Performance Measurement System: Prozesscontrolling als Instrumentarium der modernen Unternehmensführung, Deutscher Universitätsverlag, 2012 (available in German only).

10 Requirements Management in Agile Projects

10.1 Background

10.1.1 Basic Principles of Agile Development

In its pure form, a classic, phased, plan-driven process would run as follows: first, the overall project is planned, then the requirements are specified and accepted completely *upfront*, and then the requirements are implemented and tested.

However, this process does not work in all projects and requirements domains. In particular, it does not work if the requirements are not well-known enough due to a lack of knowledge or experience (e.g., for very innovative projects) or the requirements are constantly changing in a volatile project environment. Therefore, good requirements engineering usually takes place iteratively, using prototypes, for example, and not according to a pure waterfall model.

The agile development methods also recommend an iterative process, although a lightweight process with very short cycles, whereby within an iteration, only those documents that are absolutely necessary are created. Furthermore, agile processes welcome new requirements and changes to requirements at any time, as these can be considered in a subsequent iteration ("embrace change").

Of course, there are more than just these two extreme processes; there are all possible levels between upfront and iterative requirements engineering, between heavyweight and lightweight requirements engineering.

The Agile Manifesto is the common foundation of all agile approaches. From the software developers' point of view, the Agile Manifesto states [AgileManifesto]:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> *Individuals and interactions over processes and tools*
>
> *Working software over comprehensive documentation*
>
> *Customer collaboration over contract negotiation*
>
> *Responding to change over following a plan*
>
> That is, while there is value in the items on the right, we value the items on the left more.

In the agile approach therefore, collaboration, productivity, and the individual strengths of the team are more important than contracts and documentation (including requirements specifications).

This distinguishes agile methods from plan-driven approaches that require clear contractual elements (e.g., project scope, requirements specifications, release plans, or a defined change process).

The Agile Manifesto also defines thirteen principles [AgileManifesto]:

1. *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
2. *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
3. *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
4. *Business people and developers must work together daily throughout the project.*
5. *Build projects around motivated individuals.*
6. *Give them the environment and support they need, and trust them to get the job done.*
7. *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
8. *Working software is the primary measure of progress.*
9. *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
10. *Continuous attention to technical excellence and good design enhances agility.*
11. *Simplicity—the art of maximizing the amount of work not done—is essential.*
12. *The best architectures, requirements, and designs emerge from self-organizing teams.*
13. *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."*

The most important agile methods are:

- Scrum [ScBe2001], [SuSc2013], [ScSu2013]
- Extreme Programming XP [Beck2000]
- Kanban [Ande2010]
- Lean software development [PoPo2003]
- Crystal [Cock1997], [Cock2004], [Cock2006]
- Feature-driven development (FDD) [PaFe2002], [Nebu2014]

Practical tip: In actual fact, the alternatives for selecting a process model are not just "agile or waterfall". Not only are there numerous agile and numerous plan-driven process models, there are also various variants of every process model, such as hybrid processes with elements from both worlds. You can also tailor a process model from an existing model [Herr2014]. In light of these extensive selection options, it is worth analyzing the constraints and needs of your project precisely and selecting the process model carefully. We cannot look here in detail at how you do this. However, we will discuss the selection of the appropriate requirements management practices from the repertoire of agile methods. Requirements management practices from agile methods can also be applied in non-agile projects.

10.1.2 Scrum as the Representative of the Agile Methods

Scrum is currently the most widespread agile approach. For a complete description, see the Scrum Guide 2013 [ScSu2013]. Scrum is described by its process (driven by its events), its artifacts, and its roles. These are very typical for agile frameworks and can therefore be found in similar form in other agile methods.

10.1.2.1 Scrum Process

Scrum defines the work process as follows: a sprint (iteration) lasts up to four weeks. At the end of every sprint, a finished, usable, and potentially deliverable product (component, increment, etc.) must be completed.

The **sprint** contains the following events or meetings:

- *Sprint planning*: here, the entries in the *product backlog* (the list of all requirements currently elicited) that are to be processed in the next *sprint* are identified. A *sprint backlog* is created by filing the backlog items to be processed. In this *backlog*, tasks are often presented through *user stories* (see below), meaning that work is planned based on requirements.
- *Sprint*: defined period in which the team processes the items in the *sprint backlog*.
- *Daily scrum*: there is a *daily scrum* (also referred to as a *stand-up meeting*) every workday. This is a team meeting to exchange information about current work and difficulties and to plan the workday in detail.
- *Sprint review*: here, at the end of a sprint, the work results of the sprint that has just finished are discussed. The *product owner* accepts the sprint result.
- *Sprint retrospective*: after a *sprint* has finished, the *scrum team* (i.e., product owner, scrum master, and development team) discusses the collaboration. The aim is to find out how the work process in the team can be improved.

10.1.2.2 Scrum Artifacts

The ***product backlog***...

- Contains the *backlog items*, such as the *user stories* for the product to be developed, as well as technical and administrative tasks in the order of processing. The product owner should sort the backlog items in the product backlog such that the goals and missions can be achieved optimally.
- Does not have to be complete; it is maintained continuously.
- Describes high-priority *user stories* in more detail than the low-priority ones.

The ***sprint backlog***...

- Contains all *backlog items* to be realized in this *sprint*, along with the plan for the delivery of the product increment and for fulfilling the sprint goal.
- Makes all the work that the development team deems necessary to achieve the sprint goal visible.
- Is supplemented with additional work by the development team if this work is necessary to achieve the sprint goal.
- As good practice, the *backlog items* are broken down into *tasks* lasting typically one workday.

One tool that is widely used in practice is the *task board*. This is a pinboard for visualizing the *sprint backlog* and the degree of completion of the *backlog items*. The *tasks* of the current *sprint* move from left to right according to the processing status. Each column represents a status: *To Do*, *In Process*, *To Verify*, and *Done*. Each line groups the tasks that belong to one *backlog item* (e.g., *user story*). The *task board* thus presents a view of the requirements (*user stories*) and their status.

Another tool often used in practice to present project progress is the *burndown chart*. This is a graphical presentation, to be recorded every day, of the remaining effort to be performed for each sprint. In an ideal situation, the curve falls continuously (hence *burndown*) and at the end of the *sprint*, the remaining effort is zero. Here, the status and degree of completion of the current *sprint* presented on the *task board* are visualized quantitatively and graphically.

The *increment* is the completed, executable, and potentially deliverable product at the end of the *sprint*. According to the Agile Manifesto, this is the most important artifact.

The *impediment backlog* is a list of all impediments to the project. The *scrum master*, together with the team, is responsible for eliminating these impediments.

From the point of view of requirements management, the *user story* is the central artifact. A *user story* describes a requirement on an index card with a defined sentence construction. A user story usually takes the following form:

```
As <ROLE>
I want <FUNCTIONALITY>,
so that <BENEFIT>.
```

For example:

As a customer,

I want to transfer money from my account to another account
to pay my bills as soon as possible.

The specification of the benefit is optional and can be omitted. Specifying the benefit can make the added value of the functionality for the role explicit, which in turn improves the understanding of the user story.

The *user story* can also contain (e.g., at the corners) information such as the cost estimate (e.g., in *story points*), the benefit for the user (on a points scale), and the technical risk. This information is used to prioritize the *user stories*.

Acceptance criteria and test cases are also specified more precisely for every *user story*. These can be documented briefly on the rear of the index card in the following form:

```
On condition that <PRECONDITION>,
if <TRIGGER>,
then <RESULT>.
```

For example:

On condition that there is more than €100 in my account,
if I activate a transfer of €100,
then there will be €100 less displayed in my account, and €100 more than before in the target account.

Noting requirements (*user story*) and test cases on the same card provides the traceability between both with little effort.

10.1.2.3 Scrum Roles

Scrum differentiates between only three roles in the *scrum team*: *product owner*, *scrum master*, and *development team*. The *development team* organizes itself—not only the programming, but also requirements engineering, requirements management, and project management. With regard to requirements management, the tasks are divided up as follows:

- The *product owner* makes all content-based decisions: which *backlog items* (e.g., *user stories*) there are, what they cover, how they are formulated, how they are to be tested, and in particular, what priorities they have.
- The *scrum master* is responsible for the understanding and the execution of scrum. The scrum master does this by ensuring that the scrum team complies with the theory, practices, and rules of scrum; that is, the scrum master coaches the scrum team.
- The *development team* implements the requirements. The team members inform each other about the processing status in the *daily scrum*.

10.2 Requirements Management as Part of Agile Product Development

In agile development requirements are very important. Requirements in the form of *user stories* are particularly popular: the user stories are the basis for planning the iterations and work, and for monitoring the progress on the *task board* and in the *burndown chart*. However, it is typical for requirements to be specified as lightweight as possible and "just good enough". In agile methods, team members usually work closely together and communicate with each other on a daily basis. The *user story* is therefore merely a note about what was discussed. It therefore does not need to be complete or clear for third parties.

From the point of view of requirements management, the agile methods are presented as follows:

- The agile requirements landscape is generally simple: *user stories* are used with particular frequency, as well as additional acceptance tests for the specification of the requirements. If necessary, *user stories* that belong together can be grouped in *epics* (see below).
- Each requirement or *user story* has just a few attributes—for example, a cost and benefits evaluation or the risk—which are included on the *user story* card. The *product owner* evaluates the benefits, and the *development team* evaluates the technical risks and costs.
- There are just a few views of the requirements: the *product backlog*, the *sprint backlog*, and the *task board*.
- The effort for the backlog items is often estimated using *planning poker*, which we described in Section 4.5.5. The criteria used to prioritize the backlog items are the few available attributes, in particular the costs and benefits. The cost/benefit ratio thus determines the priority. Or conversely, only those attributes that are useful for the prioritization are managed.
- The order of implementation for the *user stories* is determined as follows: according to the value for the *product owner*, according to technical dependencies (functions that are the basis for others must be implemented first), according to the technical risk (*user stories* that have technical risks with regard to implementation are implemented as early as possible to allow the risks to be evaluated better at an early stage), and according to sprint topic.
- There is no version management in this sense. Completed or obsolete *user stories* usually end up in the wastepaper bin.
- The change process is simple: new requirements or changes to requirements are written to the *product backlog* and considered in the next sprint planning. If a *user story* that has not yet been realized is replaced, it ends up in the wastepaper bin. No approval process and no committee are necessary.

  The *product owner* bears the full responsibility for this. The decision about the actual implementation of new ideas is taken during the sprint planning.

- Variant management is not planned in agile development. Furthermore, traceability is at best implicit, for example, through the assignment of *user stories* to iterations or *user stories* to *epics*. The traceability between the user story and the code takes the form of check-in comments in the version management system. *User stories* and test cases must also be linked traceably with one another—for example, by being physically on the same *story card*.
- Reporting is easy and is based on requirements: the *task board* shows the processing status of every task and every *user story* for the current day. The *burndown chart* shows, quantitatively, the amount of work still to be done and whether it is likely that all planned *user stories* will have been realized by the end of the sprint. Furthermore, no reports are necessary if the team coordinates with one another daily and communicates constantly with one another.
- In the agile methods, there is no explicitly defined requirements engineering process. The *product owner* can either represent all stakeholders of the system and formulate their requirements, or is responsible for eliciting these requirements. This requirements elicitation process is not part of *scrum* and is therefore not defined here.
- The *impediment backlog* and the sprint retrospective are used to improve the work process and thereby also requirements engineering and requirements management.
- The tools to be used are simple. Originally, only index cards and a pinboard were used. However, the more that agile teams do not work in the same location, the more that simple software tools are being used. These tools implement the *backlogs*, *task board*, and *burndown chart* electronically. This allows employees distributed globally to access these artifacts at any time.

In agile development, therefore, some of the requirements management elements that we recommend are missing. Naturally, these cannot simply be omitted without any risk! From the perspective of requirements management, this lightweight, iterative agility requires the following:

- The *product owner* knows the requirements or can elicit them. If applicable, the *product owner* can consult multiple additional people. However, these persons must be constantly available and must work actively in the project.
- The *development team* has enough domain knowledge to be able to understand the requirements correctly despite their lightweight description as a *user story*. If applicable, the lightweight *user stories* can also be supplemented and made more specific with more heavyweight forms of presentation.
- The team organizes itself and takes responsibility for its own work.

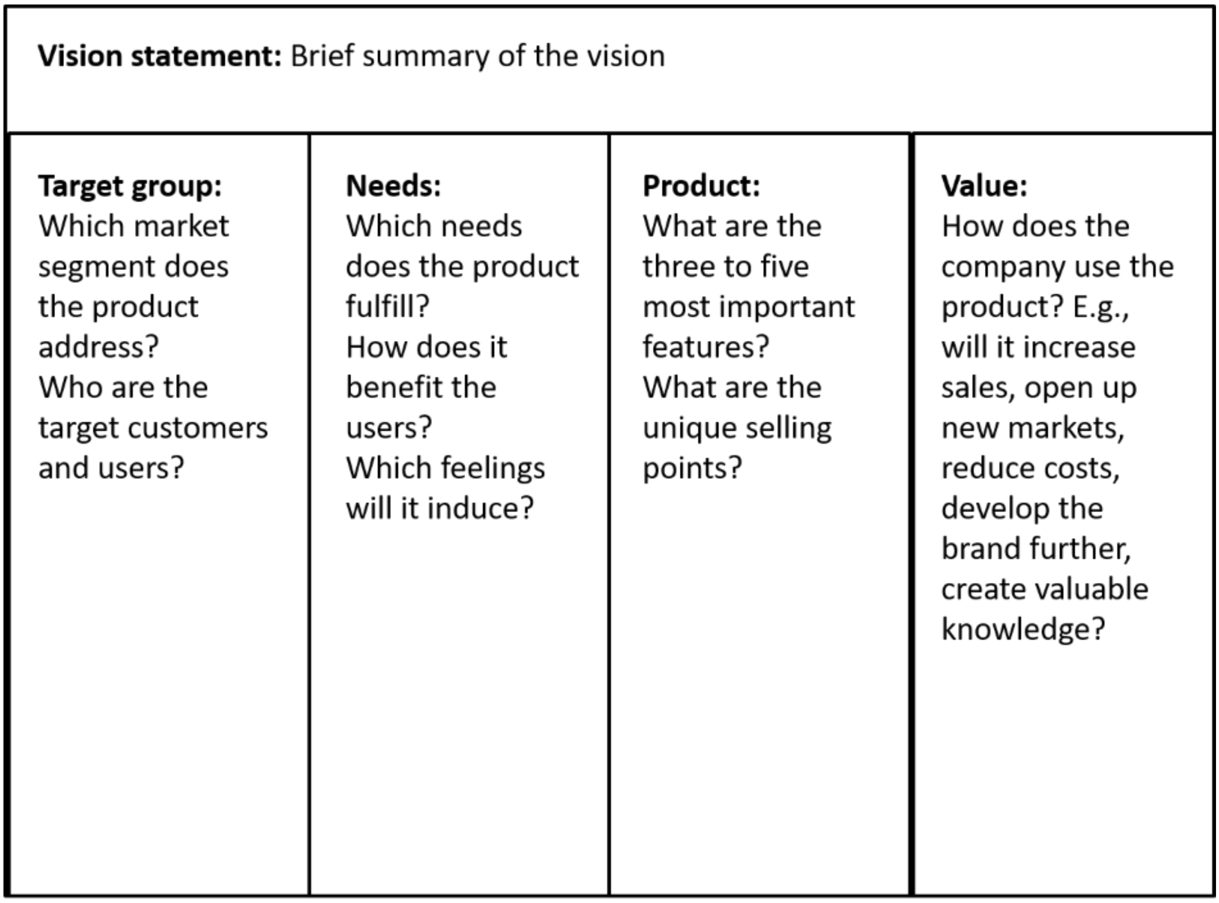
In practice, it is actually the case that sometimes, the lightweight requirements specification in the form of *user stories* and acceptance tests (see Section 10.1.2.2) is not enough. However, the agile principles also do not prohibit individual or all requirements being specified in more detail, in a more heavyweight form, or in a form other than *user stories*.

Methods from classic requirements engineering and classic requirements management can be used additionally in agile projects if the team feels this makes sense or company guidelines prescribe this. The *use case*, which is also established as a tool in classic requirements specification, can be used as an artifact in an agile environment as well (e.g., [Cock2001] or [JSB2011]).

A mixture between a classic and an agile project is also feasible, for example, the creation of a requirements specification upfront and then iterative agile development and testing.

The agile methods have already expanded with additional forms of the requirements specification that originate from classic requirements engineering or have been adapted from there in a lightweight form. In the following, we briefly describe the *vision board, minimal viable product (MVP)* and *minimal marketable product (MMP*), *epics,* and *story maps*.

The *vision board* (also referred to as the *product canvas*) [Pich2014] describes the vision and a very brief, lightweight form of a *business case* for a product or project. It has only five fields, as shown in Figure 46.



Vision statement: Brief summary of the vision			
Target group: Which market segment does the product address? Who are the target customers and users?	**Needs:** Which needs does the product fulfill? How does it benefit the users? Which feelings will it induce?	**Product:** What are the three to five most important features? What are the unique selling points?	**Value:** How does the company use the product? E.g., will it increase sales, open up new markets, reduce costs, develop the brand further, create valuable knowledge?

Source: http://scaledagileframework.com/epics/

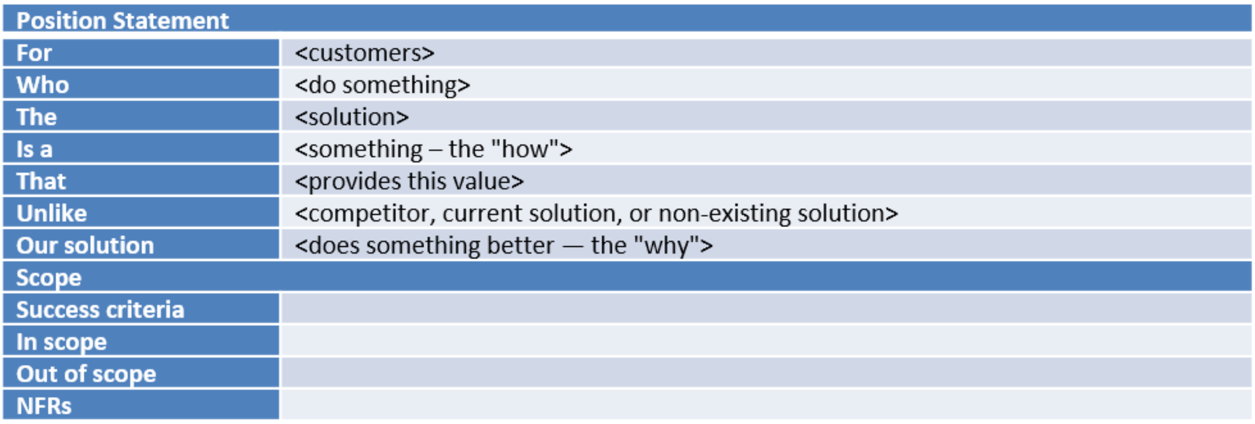
Figure 46: Vision board according to Roman Pichler [Pich2014] (own presentation)

Minimal viable product (MVP): This is the smallest product that can already be used to get feedback from stakeholders. It contains just enough features for users to be able to evaluate its usefulness [Ries2011].

Minimal marketable product (MMP): This is the smallest product that can be sold on the market. It contains just enough features to allow a user to use it usefully. It is therefore the smallest product that can be sold.

Epics are descriptions of requirements at a higher level of detail than user stories. They therefore usually group multiple user stories.

An *epic* could bear the name "Account management", for example, and comprise multiple user stories such as "View account balance", "Open account", and "Close account". *Epics* can be discussed and presented as an *epic value statement* (see Figure 47) to show who they are useful for and to what extent. This discussion of the benefits is then used to prioritize the *epics* and the associated *user stories* and therefore the iteration planning.



Position Statement	
For	<customers>
Who	<do something>
The	<solution>
Is a	<something – the "how">
That	<provides this value>
Unlike	<competitor, current solution, or non-existing solution>
Our solution	<does something better — the "why">
Scope	
Success criteria	
In scope	
Out of scope	
NFRs	

Figure 47: Epic value statement template [Leff2011]

Story maps [Patt2008] are a presentation of the overview of the connection between requirements and business processes. Story maps are used to determine the "walking skeleton", that is, the minimum implementation of a functioning business process. One possible presentation is to present the activities of the business process horizontally and to assign the associated requirements (e.g., user stories) to the respective activities.

The scaling of agile approaches to large and distributed teams is in its infancy, some frameworks are currently being developed. Some approaches can be found in [Ecks2004], [Ecks2010], [Leff2011], and [KoBe2013].

10.3 Mapping Requirements Management Activities to Scrum Activities

Scrum sees itself as a "framework within which people can address complex adaptive problems, allowing them to productively and creatively deliver products of the highest possible value" [SuSc2013]. However, scrum specifies only general work processes. In the following table, the requirements management activities are assigned to the scrum activities or artifacts. Furthermore, the executing role is specified in scrum. Not all requirements management activities are covered by scrum. (That is, some of the activities are not covered by the scrum guide. Beside the scrum guide there is a not insignificant amount of literature describing more or less successful additions to scrum. Here we refer exclusively to the scrum guide.) Whether and how the corresponding requirements management activity is then executed in a scrum project is up to the scrum team.

RM Activity	Scrum Activity or Artifact	Scrum Role
Assignment of attributes	User stories in the backlog: description, order, estimate, status, and value. Optional: grouping	PO, DT
Evaluation and prioritization	Estimation of the benefits and costs through planning poker	DT
	Arrangement of the user stories in the product backlog	PO
	Selection of user stories for a sprint	PO and DT
	Prioritization within a sprint	DT
Traceability	There is an implicit traceability of user stories to the corresponding acceptance test cases and, with suitable attribute assignment, back to the sources of the user stories. In addition, traceability is possible within the product backlog (dependencies) and from user stories to the source code. Scrum says nothing about connecting user stories within the product backlog. Traceability via epics (grouped user stories) would be conceivable. Traceability is documented only if necessary.	None
Versioning	Versioning of user stories is unnecessary. The current version of a user story is always relevant.	None or PO
Changes	Changes can be proposed at any time. New requirements lead to new user stories, changes to requirements lead to a user story being changed or replaced by a new one.	PO
Variant management	Agile methods do not explicitly support variant management. However, it is possible to use standard methods of variant management.	PO
Reporting	Reports are mainly verbal. The artifacts used to track the completion status can also serve as reports: Daily standup Sprint review Sprint retrospective Product backlog Sprint backlog Burndown chart	DT
Process management	Sprint retrospective and impediment backlog	SM, DT

Table 11: Mapping of requirements management activities to scrum

10.4 Literature for Further Reading

[AgileManifesto] Manifesto for Agile Software Development. Available at http://agilemanifesto.org/ (status: November 11, 2014).

[Beck2000] Kent Beck: Extreme programming explained. Addison-Wesley, Upper Saddle River, 2000.

[JSB2011] I. Jacobson, I. Spence, K. Bittner: Use Cases 2.0. Ivar Jacobson International, 2011.

[KoBe2013] H.-P. Korn and J.P. Berchez (eds.): Agiles IT-Management in großen Unternehmen. Symposion, 2013 (available in German only).

[Leff2011] D. Leffingwell: Agile Software Requirements, Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley Professional, 2011.

[PoPo2003] Mary Poppendieck, Tom Poppendieck: Lean Software Development. Addison Wesley, 2003.

[Ries2011] Eric Ries: The Lean Startup: How Constant Innovation Creates Radically Successful Businesses. Penguin, 2011.

[ScBe2001] Ken Schwaber, Mike Beedle: Agile Software Development with SCRUM. Prentice Hall, 2001.

[ScSu2013] Ken Schwaber, Jeff Sutherland: The Scrum Guide — The Definitive Guide to Scrum: The Rules of the Game, July 2013, http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf (status: 09/29/2014).

11 Tool-Based Requirements Management

The market for requirements engineering and requirements management tools currently includes a number of different tool providers with different license models. Everything from freeware to company licenses is represented. These tools differ above all in their core focus (documentation, collaboration, traceability, agility).

Not all of the tools available on the market can be seen as true requirements management tools, even though they can certainly be helpful for requirements management (e.g., modeling tools or version control systems (VCS)).

Via the following link, you can access an extensive list of more than 100 requirements management tools, including a classification of their core focus: http://makingofsoftware.com/resources/list-of-rm-tools. [HJD2011], describes how the requirements management tool DOORS® can be used to manage requirements.

Practical tip: You will not always find special requirements management tools being used in a company. Standard office applications and web-based platforms are often used for exchanging documents and for collaboration. Even under these conditions, a good requirements management can be implemented with some organizational rules and the required discipline. Always remember that selecting just any requirements management tool is generally not a constructive solution if you have not yet decided how you want to implement your requirements engineering process (see the requirements management plan). A short aid to selecting tools based on the requirements management aspects discussed in the handbook can be found in Annex B.

11.1 Role of Tools in Requirements Management

The use of tools is intended to make it easier for the requirements manager to document and manage requirements. Due to their special functionalities, requirements management tools enable a holistic view of requirements, in that, amongst other things, relationships between different requirements (see Chapter 6, Traceability) as well as the lifecycle of individual requirements (see Chapter 5, Version and Change Management) can be represented.

A requirements management tool is a software application whose main objective is to support activities in requirements management.

Many different applications are traditionally used in software and system development. However, many of them cover only some aspects of requirements engineering and/or requirements management. The distinction between these tools and dedicated tools for requirements management is therefore not always clear-cut.

Tools for requirements management are based on specific assumptions, which means that these tools can concentrate, for example, on specific process models, work environments, or application domains:

- Specific process models, such as agile or plan-driven development
- Specific work environments, such as local or distributed collaboration
- Specific application domains, such as the automotive industry or the armaments industry

[SoSa1997] describes the following five features as core functions of a requirements management tool:

- **Editor for requirements**, including their attributes, to enable the recording and attribute assignment for uniquely identifiable requirements artifacts; the editor can be a pure text-based editor or an editor that supports textual and model-based descriptions
- **Import of requirements** from existing documents into the tool (e.g., based on the ReqIF format, see Section 11.3) and e**xport of managed requirements** to other formats (e.g., in document-based specifications)
- **Tracing of requirements**, beginning with support for maintenance of traceability relationships up to the use of maintained traceability relationships—for example, as part of an impact analysis
- **Versioning** of requirements and the creation of requirements configurations and baselines
- The creation of **user-defined views** of requirements, including their attributes

If we compare these features with those from [PoRu2011] (Section 9.3), we find the following additional features which are important when selecting a requirements management tool:

- Distributed processing of requirements artifacts, including access control
- Creation of role-specific views for different user groups
- Creation of reports or evaluations of the managed artifacts

Regardless of the features that a tool offers, when selecting and introducing a tool, note that any requirements management tool selected must fit with the procedures and processes established in the company.

11.2 Basic Procedure for Tool Selection

Selecting the right tool is not easy. There are a lot of tools, and the tool that best meets your situation depends on your own requirements engineering process. It is the process that determines the requirements for the tool.

Requirements management tools are usually selected for more than just one single project. It is often the case that tools are selected for multiple projects—for example, for all projects in a department or a company. This generally makes the tool selection complex, which means that the introduction of a requirements management tool is often driven by a separate project, see also [RuSo2009].

The recommendation is to implement the tool evaluation and selection through a separate project. [RuSo2009] uses a two-phased selection procedure in which there is an initial, rough selection for a first potential tool list (long list), and in a second step, an advanced selection to reduce the tool list to the favorites (short list). Based on the short list, a selection decision is taken and the tool is then introduced into the company through a project, and potentially tailored to the company-specific requirements (customizing). Furthermore, to increase acceptance, initial use in a pilot project is recommended. If the pilot project reveals that the selected tool does not provide the desired support, the tool selection must be repeated. If no tool meets the requirements exactly, the process can be adjusted instead of a tool.

Tool selection according to [RuSo2009]:

- Launch a tool selection **project**.
- Define **rough selection criteria** by formulating basic requirements.
- **Perform the rough selection** (long list) to identify the first potential systems.
- **Refine the catalog of criteria** on the basis of new and refined requirements for the tool.
- **Conduct a fine selection** (short list), up to a favored software candidate.
- Optional: If no tool meets requirements precisely, the software application must be **adapted** (customized).
- In order to strengthen the acceptance in the company and to eliminate possible last concerns, a pilot project is then launched.

Tip: Annex B contains some useful criteria for tool selection based on the requirements management plan.

11.3 Data Exchange between Requirements Management Tools

The import and export of requirements, attributes, meta-information, links, and associated views is necessary, for example, to support collaboration with other departments, partners, and suppliers who use tools from other providers. Such functions are also required if migrations from one tool to another are planned.

In most requirements management tools, requirements and their relationships to one another are placed in manufacturer-specific (proprietary) structures. This means that a simple exchange between two requirements management tools from different manufacturers is generally not possible without a lot of effort (even if the requirements information model is identical).

The Object Management Group (see [OMG2013]) has defined the industry standard Requirements Interchange Format (ReqIF). This allows requirements artifacts and meta-information to be exchanged between tools from different manufacturers. It is used primarily at the interface between the customer and the supplier. In addition to the exchange format, a procedure is also defined.

The initiative comes from the automotive industry, which features close collaboration between suppliers and the automotive manufacturers, whereby precisely defined versions of requirements artifacts have to be exchanged.

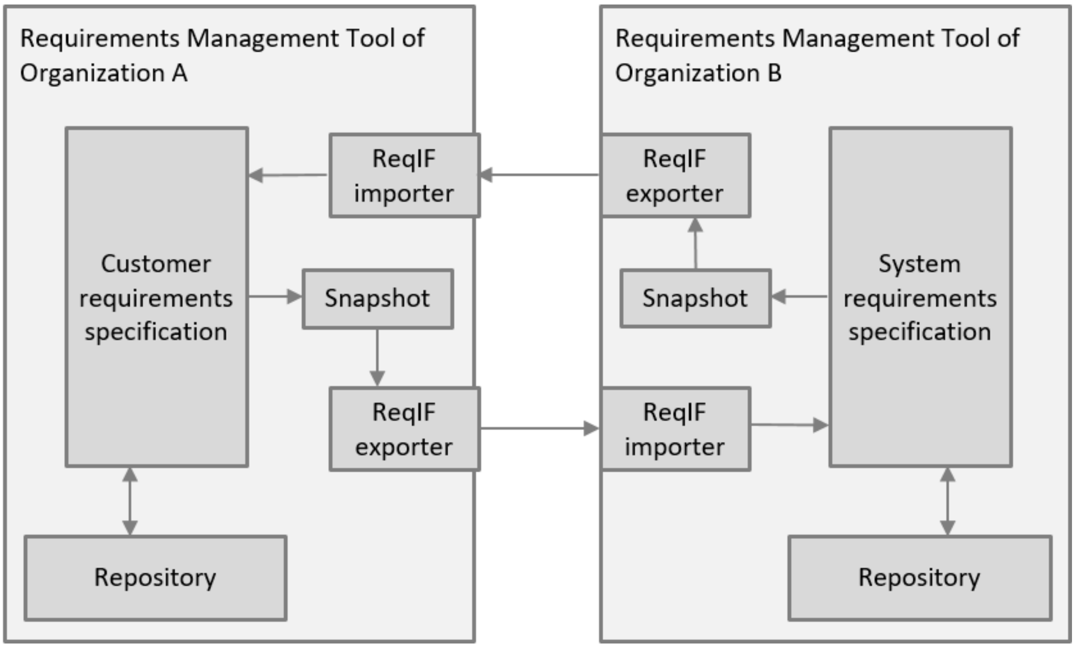

Requirements Management Tool of Organization A
Customer requirements specification
ReqIF importer
Snapshot
ReqIF exporter
Repository
Requirements Management Tool of Organization B
ReqIF exporter
Snapshot
System requirements specification
ReqIF importer
Repository

Figure 48: Example of an exchange of requirements artifacts between two organizations (simplified according to [OMG2013])

ReqIF is an open, non-proprietary format. It is stored in XML documents. ReqIF thus enables the exchange of requirements between different tools and partners. However, the prerequisite for this is a standardized, aligned data model for the exchange (e.g., a requirements information model).

ReqIF therefore offers the following advantages for data exchange here:

- The partners do not have to work with the same tool, which means that the suppliers do not need to have a separate requirements engineering tool for each customer.
- With ReqIF, collaboration between companies can be improved by applying requirements management methods across companies.
- Requirements can be transferred within an organization, even across tool boundaries.
- With ReqIF, requirements, with all attributes and meta-information, can be exchanged without loss, unlike document exports in Word, PDF, etc.

Figure 48 shows the process of an exchange of requirements artifacts between organizations. The specification of the requirements ("Customer requirements specification" in organization A, and "System requirements specification" in organization B) is versioned using a repository.

The tools have interfaces for exporting and importing the requirements. Using snapshots from the specification, content is transferred between the tools. The specification of how the exact data exchange is to take place within the scope of the project is documented in the requirements management plan.

11.4 Content for the Requirements Management Plan

From a tool perspective, in the requirements management plan (see also Annex A), you define how you want to use requirements management tools in your specific context. You can base the description on the previously described chapters. Document what you want to support or map, in which form, and using which tool, for example:

- Chapter 2 (Requirements Information Model): DOORS® is to be used to describe all of the textual requirements in the requirements information model and the levels of detail defined there. The model-based requirements (class diagrams, BPMN diagrams) are to be described in Visual Paradigm.
- Chapter 3: DOORS® is to be used to create and maintain the attributes defined for the textual requirements defined in Chapter 2. All user-defined views are to be defined in DOORS® and assigned using role-based access rights.

In addition to the requirements information model, the techniques to be used for prioritization, the version and change management, the implementation of traceability, the selected procedure for variant management, the actual requirements engineering process, and the reporting, the requirements management plan also describes which of these techniques or activities are to be supported by a tool and by which tool.

Furthermore, the requirements management plan should also describe the parties between which requirements have to be exchanged, and how this exchange is to take place using defined imports and exports (see Section 11.3).

11.5 Literature for Further Reading

[OMG2013] OMG: Requirements Interchange Format (ReqIF). Object Management Group, Version 1.1., 2013, http://www.omg.org/spec/ReqIF/1.1/PDF/ (status: November 11, 2014).

[Pohl2010] K. Pohl: Requirements Engineering – Foundations, Principles, and Techniques. Springer, 2010.

[RuSo2009] C. Rupp & die SOPHISTen: Requirements-Engineering und –Management. Hanser, 5th edition, updated and extended, 2009 (available in German only).

List of Abbreviations

AC	actual cost
AHP	analytical hierarchy process
BAC	budget at completion
CAB	Change Advisory Board
CCB	Change Control Board
CMMI	capability maturity model integration
CPRE	Certified Professional for Requirements Engineering
CR	change request
DIN	Deutsches Institut für Normung (German Institute for Standardization)
EV	earned value
FDD	feature-driven development
GQM	goal, question, metric
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IREB	International Requirements Engineering Board
ISO	International Organization for Standardization
IT	information technology
ITIL	IT Infrastructure Library
CPI	continuous process improvement
MMP	minimal marketable product
MVP	minimal viable product
PDCA	plan, do, check, act
PV	planned value
RE	requirements engineering
RACI	responsible, accountable, consulted, informed
RIM	requirements information model
RM	requirements management
RMP	requirements management plan
TBD	to be determined
TBR	to be resolved
UML	Unified Modeling Language
XP	Extreme Programming

Bibliography

[AgileManifesto] Manifesto for Agile Software Development. Available at http://agilemanifesto.org/ (status: November 11, 2014).

[Ande2010] David J. Anderson: Kanban. Successful Evolutionary Change for Your Technology Business. Blue Hole Press, Sequim, Washington 2010.

[BaWe1984] Victor R. Basili, David M. Weiss: A methodology for collecting valid software engineering data. Software Engineering, IEEE Transactions on Software Engineering (1984): 728–738.

[Basi1992] V.R. Basili: Software Modeling and Measurement: The Goal Question Metric Paradigm. Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96), University of Maryland, College Park, MD, September 1992.

[BBHK2014] Braun, P.; Broy, M.; Houdek, F.; Kirchmayr, M.; Müller, M.; Penzenstadler, B.; Pohl, K.; Weyer, T.: Guiding requirements engineering for software-intensive embedded systems in the automotive industry. Computer Science - R&D 29(1): (2014).

[BCR] Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach: The Goal Question Metric Approach. Tutorial, University of Maryland, http://www.cs.umd.edu/~mvz/handouts/gqm.pdf (status: November 11, 2014).

[Beck2000] Kent Beck: Extreme programming explained. Addison-Wesley, Upper Saddle River, 2000.

[Beim2012] Martin Beims: IT-Service Management mit ITIL®: ITIL® Edition 2011, ISO 20000:2011 und PRINCE2® in der Praxis, Carl Hanser Verlag GmbH & Co. KG, 3rd updated edition, 2012 (available in German only).

[BSB2008] Christoph Bommer, Markus Spindler, Volkert Barr: Softwarewartung – Grundlagen, Management und Wartungstechniken. Dpunkt.verlag, 2008 (available in German only).

[Bout2011] E. Boutkova: Experience with Variability Management in Requirement Specifications. In: D.E. Almeida, T. Kishi, C. Schwanninger, I. John, and K. Schmid (eds): Software Product Lines – 15th International Conference (SPLC), Munich, 2013.

[BoHo2011] E. Boutkova, F. Houdek: Semi-automatic identification of features in requirement specifications. In: Proceedings of the 19th International Requirements Engineering Conference, Trento, Italy, September 2011.

[BWJ2013] Franz J. Brunner, Johann Wappis, Berndt Jung: Null-Fehler-Management: Umsetzung von Six Sigma, Carl Hanser Verlag GmbH & Co. KG; edition: 4, revised and extended edition, 2013 (available in German only).

[BLP2004] S. Bühne, K. Lauenroth, K. Pohl: Why is it not Sufficient to Model Requirements Variability with Feature Models. In: Aoyama, M.; Houdek, F.; Shigematsu, T. (eds) Proceedings of Workshop: Automotive Requirements Engineering (AURE04). IEEE Computer Society Press, Los Alamitos 2004.

[CMMI] http://www.cmmi.de/ (status: November 11, 2014).

[CNAT2011] J.M. Carillo de Gea, J. Nicolás, J.L.F. Alemán, A. Toval, C. Ebert, A. Vizcaíno: Requirements Engineering Tools. In: IEEE Software, July/August 2011.

[CNAT2012] J.M. Carillo de Gea, J. Nicolás, J.L.F. Alemán, A. Toval, C. Ebert, A. Vizcaíno: Requirements Engineering Tools: Capabilities, survey, and assessment. In: Information and Software Technology, Volume 54, Issue 10, October 2012.

[CKS2011] Mary Beth Chrissis, Mike Konrad, Sandy Shrum: CMMI for Development: Guidelines for Process Integration and Product Improvement, Addison Wesley, 2011.

[Cock1997] A. Cockburn: Surviving Object-Oriented Projects. Addison-Wesley, 1997.

[Cock2001] A. Cockburn: Writing Effective Use Cases. Addison-Wesley, 2001 (available in German only).

[Cock2004] A. Cockburn: Crystal Clear, A Human-Powered Methodology for Small Teams. Addison-Wesley, 2004.

[Cock2006] A. Cockburn: Agile Software Development. Addison-Wesley, 2006.

[ClNo2007] P. Clements, L. Northrop: Software Product Lines: Practices and Patterns. Addison Wesley, Boston, 6th Edition, 2007.

[CHW1998] J. Coplien, D. Hoffmann, D. Weiss: Commonality and Variability in Software Engineering. In: IEEE Software, Volume 15, Issue 6, 1998.

[CzEi2000] K. Czarnecki, U.W. Eisenecker: Generative Programming: Methods, Tools, and Applications. Addison Wesley, 2000.

[CHQW14] Thorsten Cziharz, Peter Hruschka, Stefan Queins, Thorsten Weyer: Handbook of Requirements Modeling According to the IREB Standard, Education and Training for IREB Certified Professional for Requirements Engineering Advanced Level "Requirements Modeling" IREB, Version 1.0-3, March 1, 2014.

[Davi2003] A. Davis: The Art of Requirements Triage. IEEE Computer, Volume 36, Issue 3, 2003.

[Davi2005] Alan M. Davis: Just Enough Requirements Management – Where Software Development Meets Marketing. Dorset House Publishing, 2005.

[DeMa1982] Tom DeMarco: Controlling Software Projects: Management, Measurement, and Estimation. Prentice Hall/Yourdon Press, 1982.

[DeMa2009] Tom DeMarco: Software Engineering: An Idea Whose Time Has Come and Gone? IEEE Software, July/August 2009.

[Demi1982] W.E. Deming: Out of the Crisis. Massachusetts Institute of Technology, Cambridge 1982.

[DIN 61508] IEC DIN EN 61508-2 Functional safety of electrical/electronic/programmable electronic safety-related systems. VDE Verlag, 2002.

[DIN69900] DIN 69900 Project management – Project network techniques; Descriptions and Concepts, 2009.

[Ebel2014] N. Ebel: ITIL®(R) 2011 Edition: Grundlagen und Know-how für das IT Service Management und die ITIL®(R)-Foundation-Prüfung, dpunkt.verlag GmbH, 1st edition, 2014 (available in German only).

[Eber2012] C. Ebert: Systematisches Requirements Engineering. Dpunkt, 4th edition, 2012 (available in German only).

[Ecks2004] J. Eckstein: Agile Software Development in the Large. Dorset House Publishing, 2004.

[Ecks2010] J. Eckstein: Agile Software Development with Distributed Teams. Dorset House Publishing, 2010.

[Gabl2014a] Springer Gabler Verlag (ed.), Gabler Wirtschaftslexikon, keyword: "Produktfamilie", online: http://wirtschaftslexikon.gabler.de/Archiv/135585/produktfamilie-v5.html (status: November 11, 2014).

[Gabl2014b] Springer Gabler Verlag (ed.), Gabler Wirtschaftslexikon, keyword: "Produktlinie", online: http://wirtschaftslexikon.gabler.de/Archiv/56903/produktlinie-v6.html (Stand: November 11, 2014).

[GoFi1994] O.C.Z. Gotel, A.C.W Finkelstein: An Analysis of the Requirements Traceability Problem. Proceedings of IEEE International Conference on Requirements Engineering, 1994.

[Glin2014] Martin Glinz: A Glossary of Requirements Engineering Terminology. Version 1.6 May 2014.

[Herr2014] A. Herrmann: Leichte Dellen - Wenn agil nicht geht: Feature Driven Development. iX 9/2014, pp. 110–113 (available in German only).

[HJD2011] E. Hull, K. Jackson, J. Dick: Requirements Engineering. Springer, 3rd edition, 2011.

[HuMa2011] Thomas Hummel, Christian Malorny: Total Quality Management: Tipps für die Einführung. Carl Hanser Verlag GmbH & Co. KG; edition: 4, completely revised edition, 2011 (available in German only).

[IEEE830] IEEE: IEEE 830-1998 Recommended Practice for Software Requirements Specifications, 1998.

[IEEE1233] IEEE: IEEE Standard 1233 Guide for Developing of System Requirements Specifications, 1998.

[IREB2015] IREB: Syllabus IREB Certified Professional for Requirements Engineering – Foundation Level, Version 2.2, 2015.

[ISO9000] ISO: ISO 9000-1 Quality systems – Model for Quality Assurance in Design, Development, Production, Installation and Servicing. International Organization for Standardization (ISO), 1994.

[ISO9241] ISO: DIN EN ISO 9241 Ergonomics of human-system interaction.

[ISO12207] ISO: ISO/IEC 12207: 1995, Information Technology – Software life cycle processes. International Organization for Standardization (ISO), 1995.

[ISO29148] ISO: ISO/IEC/IEEE 29148:2018: Systems and software engineering – Life cycle processes – Requirements engineering, 2018.

[ISO14102] ISO/ IEC 14102:1995 Information Technology – Evaluation and Selection of CASE Tools, 1995.

[ISO15288] ISO/IEC 15288:2008 Systems and software engineering — System life cycle processes, 2008.

[ISO24766] ISO: ISO/IEC TR 24766:2009: Information technology – Systems and software engineering – Guide for requirements engineering tool capabilities. International Organization for Standardization (ISO), 2009.

[ISO25010] ISO: ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, 2011.

[ISO29110] ISO: ISO 29110 Lifecycle process standard for Very Small and Medium Entities (VSME), 2011.

[Oran2013] Foundations of IT Service Management with ITIL®2011. 2nd Edition ITILyaBrady, 2013 (Kindle Edition).

[JSB2011] I. Jacobson, I. Spence, K. Bittner: Use Cases 2.0. Ivar Jacobson International, 2011.

[KCHN1990] C. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson: Feature-Oriented Domain Analysis (FODA) – Feasibility Study. Software Engineering Institute, 1990.

[KKLK1998] K. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering, No. 5, 1998.

[KLD2002] K. Kang, J. Lee, P. Donohoe: Feature-Oriented Product Line Engineering. IEEE Software 19(4): 2002.

[Kano1984] N. Kano: Attractive Quality and Must-be Quality. Journal of the Japanese Society for Quality Control, H. 4, 1984.

[KaRy1997] J. Karlsson, K. Ryan: A Cost-Value Approach for Prioritizing Requirements. IEEE Software 14, No. 5, 1997.

[Kerz2003] H. Kerzner: Project Management. A Systems Approach to Planning, Scheduling, and Controlling. John Wiley & Sons, 2017.

[Kneu2007] Ralf Kneuper: CMMI: Improving Software and System Development Processes Using Capability Maturity Model Integration. Rocky Nook, 1st edition, 2009.

[KoBe2013] H.-P. Korn and J.P. Berchez (eds.): Agiles IT-Management in großen Unternehmen. Symposion Publishing, 2013 (available in German only).

[Küpp2005] H.-U. Küpper: Controlling: Konzeption, Aufgaben, Instrumente. Schäffer-Poeschel, 4th edition, 2005 (available in German only).

[KuSt2001] K. Kurbel, E. Stickel: Informationsmanagement. Oldenbourg Wissenschaftsverlag, 2001 (available in German only).

[Leff2011] D. Leffingwell: Agile Software Requirements, Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley Professional, 2011.

[LeWi2000] D. Leffingwell, D. Widrig: Managing Software Requirements – A Unified Approach. Reading, Addison-Wesley, 2000.

[LoKa1995] P. Loucopoulos, V. Karakostas: System Requirements Engineering. McGraw-Hill, 1995.

[MGP2009] P. Mäder, O. Gotel, I. Philippow: Getting Back to Basics: Promoting the Use of a Traceability Information Model in Practice. Proceedings of 5th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE2009), Vancouver, Canada, May 2009.

[MJZC2013] P. Mäder, P.L. Jones, Y. Zhang, J. Cleland-Huang: Strategic Traceability for Safety-Critical Projects. IEEE Software, Volume 30, Issue 3, May/June 2013.

[Mois2002] F. Moisiadis: The fundamentals of prioritizing requirements. Systems Engineering, Test & Evaluation Conference, Sydney, October 2002.

[Nebu2014] Nebulon Pty. Ltd: Feature Driven Development.
http://www.featuredrivendevelopment.com/ (status: 09/29/2014).

[Nuse2001] B. Nuseibeh: Weaving the Software Development Process between Requirements and Architecture. Proceedings of ICSE2001 Workshop STRAW-01, Toronto, May 2001.

[OMG2013] OMG: Requirements Interchange Format (ReqIF). Object Management Group, Version 1.1., 2013, http://www.omg.org/spec/ReqIF/1.1/PDF/ (status: November 11, 2014).

[PaFe2002] Stephen R. Palmer, John M. Felsing: A Practical Guide to the Feature-Driven Development. Prentice Hall International, 2002.

[Patt2008] Jeff Patton: The new user story backlog is a map, 10/08/2008
https://www.jpattonassociates.com/the-new-backlog/ (status: September 29, 2014).

[PHAB2012] Pohl, K., Hönninger, H., Achatz, R., Broy, M. (Eds.): Model-Based Engineering of Embedded Systems - The SPES 2020 Methodology, Springer 2012.

[Pich2014] Roman Pichler: The Product Vision Board,
http://www.romanpichler.com/tools/vision-board/ (status: 09/28/2014).

[PMI2013] PMI: Project Management Book of Knowledge (PMBOK). Project Management Institute, 5th Ed., 2013.

[Pohl1996] K. Pohl: Process-Centered Requirements Engineering. John Wiley Research Science Press, 1996.

[Pohl2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, 2010.

[PBL2005] K. Pohl, G. Böckle, F. van der Linden: Software Product Line Engineering – Foundations, Principles, and Techniques. Springer, 2005.

[PoRu2015] K. Pohl and Chris Rupp: Requirements Engineering Fundamentals - A Study Guide for the Certified Professional for Requirements Engineering Exam - Foundation Level - IREB compliant. Rocky Nook, 2015.

[PoPo2003] Mary Poppendieck, Tom Poppendieck: Lean Software Development. Addison Wesley, 2003.

[PriEsT] PriEsT, http://sourceforge.net/projects/priority/ (status: September 25, 2014).

[RBSP2002] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow: Extending Feature Diagrams with UML Multiplicities. in Proc. World Conf. Integrated Design and Process Technology (IDPT), 2002.

[Ries2011] Eric Ries: The Lean Startup: How Constant Innovation Creates Radically Successful Businesses. Penguin, 2011.

[RoRo2014] S. Robertson, J. Robertson: Mastering the Requirements Process – Getting Requirements Right. Addison-Wesley, 3rd Edition, 2014.

[RuSo2009] C. Rupp & die SOPHISTen: Requirements-Engineering und –Management. Hanser, 5th edition, updated and extended, 2009 (available in German only).

[Saat1990] Thomas L. Saaty: Multicriteria decision making – the analytic hierarchy process. Planning, priority setting, resource allocation. 2nd edition, RWS Publishing, Pittsburgh 1990.

[Schi2001] B. Schienmann: Kontinuierliches Anforderungsmanagement. Prozesse – Techniken – Werkzeuge. Addison-Wesley, 2001 (available in German only).

[SHT2006] P.-Y. Schobbens, P. Heymans, J.C. Trigaux: Feature Diagrams: A Survey and a Formal Semantics. Proceedings of the 14th International Requirements Engineering Conference (RE'06), September 2006.

[ScBe2001] Ken Schwaber, Mike Beedle: Agile Software Development with SCRUM. Prentice Hall, 2001.

[ScSu2013] Ken Schwaber, Jeff Sutherland: The Scrum Guide, July 2013 http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf (status: 09/29/2014).

[SEI1999] Carnegie Mellon SEI: The Capability Maturity Model, Guidelines for Improving the Software Process. Addison Wesley, 1999.

[SEI2010] Carnegie Mellon SEI: CMMI for Services, Version 1.3, Improving processes for providing better services. 2010.

[SEI2011] SEI: CMMI® for Development, Version 1.3 CMU/SEI-2010-TR-033. Available at https://resources.sei.cmu.edu/asset_files/TechnicalReport/2010_005_001_15287.pdf (status: August 28, 2015).

[SMK2013] S. Siraj, L. Mikhailov, J.A. Keane: PriEsT: an interactive decision support tool to estimate priorities from pairwise comparison judgments. International Transactions in Operational Research, 2013.

[SoSa1997] I. Sommerville, P. Sawyer: Requirements Engineering: A Good Practice Guide. John Wiley & Sons, 1997.

[SpLi2007] A. Spillner, T. Linz: Basiswissen Softwaretest – Aus- und Weiterbildung zum Certified Tester. Dpunkt.verlag, 3rd edition, 2007 (available in German only).

[SuSc2013] J. Sutherland, K. Schwaber: Scrum Guide, July 2013, available at www.scrum.org.

[Syra2014] Ingo Geppert, Torsten Lodderstedt: Projektanforderungsmanagement - Eine pragmatische Lösung für effiziente Toolunterstützung. Projektmanagement, edition 4, 2010 http://www.syracom.de/uploads/media/Projektanforderungsmanagement.pdf (status: 11/06/2014) (available in German only).

[Tava2012] Serkan Tavasli: Six Sigma Performance Measurement System: Prozesscontrolling als Instrumentarium der modernen Unternehmensführung. Deutscher Universitätsverlag, 2012 (available in German only).

[USCo2002] US Congress: Sarbanes-Oxley Act. Washington, USA, 107th Congress of the United States of America, 23.01.2002.

[VanL2009] A. van Lamsweerde: Requirements Engineering – from System Goals to UML Models to Software Specifications. John Wiley and Sons, 2009.

[VDI2001] VDI: VDI guideline 2519 sheet 1 - The procedure for the creation of tender and performance specifications, 2001.

[Wann2013a] Roland Wanner: Earned Value Management: The Most Important Methods and Tools for an Effective Project Control. CreateSpace Independent Publishing Platform, 2013.

[Wann2013b] Roland Wanner: Earned Value Management: So machen Sie Ihr Projektcontrolling noch effektiver Taschenbuch. CreateSpace Independent Publishing Platform, 3rd edition, 2013 (available in German only).

[Wieg2005] Karl Wiegers: Software Requirements. Microsoft Press Deutschland, 1st edition, 2005.

[WiBe2013] K. Wiegers, J. Beatty: Software Requirements. 3rd Edition. Microsoft Press, 2013.

[Youn2014] R. Young: The Requirements Engineering Handbook, Artech House, Boston, 2004.

[Zieg1998] K. Ziegbein: Controlling. Kiehl Friedrich Verlag, 6th edition, 1998.

Index

Annex A: Template for a Requirements Management Plan

As mentioned in the introduction to our handbook for IREB Certified Professional for Requirements Engineering Advanced Level "Requirements Modeling", the requirements manager must plan the requirements engineering process at the beginning of a project. The requirements manager documents the results of these considerations in a **requirements management plan (RMP)**. Over the course of this book, we have discussed the decisions that have to be taken and, using a case study, created excerpts from an example requirements management plan. As a summary, this annex now presents a template for a requirements management plan, the respective chapter headings, and a short description for each chapter. As the requirements manager, you can create your requirements management plan according to this schema.

<Your name>

Requirements Management Plan for <your project>

Based on: IREB Certified Professional for Requirements Engineering Advanced Level “Requirements Management”

Table of Contents

1 The Requirements Engineering and Requirements Management Process

In this chapter, define the process that you want to use to elicit, document, validate, negotiate, and manage requirements. To do so, define the following parameters:

- Timing of the elicitation (upfront or iterative)
- Level of detail of the documentation, that is, the highest level of detail used for the specification (with the two extremes, heavyweight versus lightweight specification)
- Incorporation of changes, in particular: change request versus product backlog
- Allocation of responsibility

Document the process as an activity diagram, a RACI matrix, or Gantt diagram, for example.

Furthermore, document how the requirements engineering process will be monitored (e.g., the report and key figures used). Actions for process improvement can also be planned here, for example Lessons Learned analyses after the end of the project.

! For more details about the content, see Chapter 9 of the handbook.

2 Requirements Engineering and Requirements Management Tools

Define which tool or tools are to be used in your project to support the requirements engineering process. Alternatively, the tools can be documented in a process model.

! For more details about tools and in particular, tool selection, see Chapter 11 of the handbook.

3 Requirements Information Model

In this chapter, define your requirements landscape and describe how you want to document your requirements. This includes, for example:

- Which types of requirements do you want to consider?
- How do you want to document these requirements?
- To what level of detail will you describe the requirements and which levels of detail should be considered?
- Which level of formality must your requirements reach?

This information can be documented in the form of a requirements information model (RIM), for example.

! For more details about the content, see Chapter 2 of the handbook.

4 Attribute Schema

In this chapter, document the attributes that your requirements should have, using, for example, a table or information model.

- Name of the attribute
- Meaning
- Person responsible
- Permissible values
- Default value
- Mandatory field
- Dependencies between attributes

The attributes can differ depending on the requirement type or level of detail. The attribute schema also includes attributes that are to be used for prioritizing the requirements.

! For more details about the content, see Chapter 3 of the handbook.

5 Prioritization

At the beginning of the project, document the criteria that you want to use for prioritization. Prioritization is generally necessary to allow you to work to a specific schedule or budget, or in case of doubt, even both. In this chapter, define the criteria to be used to prioritize your requirements, when they are to be prioritized, by whom, and the prioritization method to be used.

! For more details about the content, see Chapter 4 of the handbook.

6 Traceability

In this chapter, document the traceability strategy: the traceability goal, usage strategy, recording strategy, and the project-specific traceability model.

! For more details about the content, see Chapter 6 of the handbook.

7 Views and Reports

Based on the attributes for requirements described in Chapter 4, in this chapter, you can define the required views and their content. Here, you also describe who (which stakeholders) needs each respective view when and for what reason (goal). The content of the view is created, for example, by filtering and sorting the requirements according to attributes.

Here, you also define which (requirements-based) reports are to be created and when they are to be created. For each report, the report recipient and the goal of the report are documented, for example in tabular form. The derivation of report content from goals can be represented graphically as a goal, question, metric tree. You also define how this content can be determined or calculated from which attributes, and how the content is presented (e.g., the specific graphical form of presentation). The specification can also be documented in the form of a report template or view.

! For more details about the content, see Chapters 3 and 8 of the handbook.

8 Versioning

Here, document how you want to version requirements and documents in your project. Define the statuses that a requirement may take, how the status transitions are to take place, and who is permitted to change the status of requirements artifacts.

In addition, define the basis for creating a requirements baseline and what the creation of such a baseline means for the subsequent requirements management process—for example, following a requirements baseline, changes are accepted only via a change management process. In the requirements management plan, define how you want to handle changes in the project, how changes are to be documented, whether there is a change committee, who makes up this change committee, etc.

! For more details about the content, see Chapter 5 of the handbook.

9 Change Process

In this chapter, describe your change management process and the associated documents. Here, describe who can request changes to requirements, and how changes are requested, evaluated, and decided—that is, by whom, when, and according to which criteria. The change process can also include a template for a change request which defines the information that has to be determined and documented for a change request.

! For more details about the content, see Chapter 5 of the handbook.

10 Variant Management

In this chapter, define whether and how you want to document variability—that is, variation points, variants, and their dependencies—in your requirements. You can do this for example in text form, as an orthogonal model, or as a feature model.

! For more details about the content, see Chapter 7 of the handbook.

Annex B (Tool Selection)

In this annex, we describe some criteria for tool selection. These criteria originate on the one hand from literature, and on the other hand, from the requirements management plan created step by step in this handbook and the question of which of the activities, processes, and techniques described in the requirements management plan should/could be supported by a management tool. For this purpose, we have created a criteria catalog and applied it using three completely different tools. These tools each represent one tool category. The evaluations are based on the status at 2015 and become obsolete, of course, as soon as the tools are developed further. Naturally, we do not recommend Microsoft Word ® and Microsoft Excel ® for requirements management because there are many requirements management activities that they do not support well.

We hope you enjoy studying these criteria and trying them out.

1 The Challenges of Introducing and Using Tools

Just like the introduction of any application, the introduction of a (new) requirements management tool is a topic which, in addition to technical and methodological aspects, must above all consider human aspects, as it is humans that will have to subsequently work with the tool. In the long term, only an accepted tool that provides the user with a direct or indirect benefit will be accepted and used correctly. When a requirements management tool is introduced, social, cognitive, organizational, and corporate aspects must be taken into account.

To illustrate this, we want to describe examples of some of the challenges of introducing a tool. We have assigned these examples to the views introduced in [PoRu2011a].

Provider view

- **Market position**: For a company, the selection of the tool is generally a long-term decision because the introduction of the tool often also necessitates organizational changes. When a tool is selected, this creates a tie to the provider of the tool. If this provider or the tool soon ceases to exist, good advice literally becomes expensive.
- **Trends and updates**: It may be desirable for the provider to support future trends and offer these in the tool. However, following trends may also be undesirable if the provider changes their strategy completely and decides to pursue, for example, only agile procedures.

User view

- **User acceptance**: The tool must be available for many user groups to be able to reflect the development cycle (marketing, development, finance department, etc.). User acceptance is therefore extremely important to ensure that the tool is subsequently used correctly. Not only must your work processes be supported, this support must also be provided in an efficient and ergonomic way. In particular, the tool must also be user-friendly.

Economic view

- **The license model** of the software must fit the usage profile and the cost structure of the company.
- **The operability** of the software must be compatible on the one hand with the cost structure, and on the other hand with the existing IT infrastructure of the company.

Product view

- **The capabilities** of the software must satisfy the requirements of the tool. On the one hand, this means that criteria of the requirements management plan (assignment of attributes, view creation, traceability, reporting, etc.) relevant for you must be fulfilled. On the other hand, it also means that the tool must deliver the required user support in all areas. It is generally a misconception to think that the primary issue is that a functionality must be supported, regardless of how this is done, and the user will accept whatever is offered. Let us take the example of traceability: if the tool supports the creation of traceability relationships between artifacts in principle, but the creation of these relationships is not user-oriented and does not save the user any time, these relationships will probably not be set or will only be set unsatisfactorily. Given the number of (possible) functional and non-functional requirements of the requirements management tool, it is difficult to impossible to find the perfect tool. Therefore, you have to prioritize your selection criteria cleverly and sacrifice only what is really unimportant.

Project view

- **Adaptability**: No two projects are the same. In the same way that the requirements engineering process and the requirements documents have to be adapted to the size and other properties of the project, the tool must also support this adaptation. If it does not, it can only be used in some projects and not in others.

Process view

- **The tool follows the process**: Before the tool is selected, the process must be clear and established in the company, based, for example, on an existing requirements management plan for the company. The tool helps only to support an existing process. If there is no defined process, it is difficult to select a tool and this restricts the possible subsequent requirements management procedures.
- **Methodological knowledge**: On its own, the use of a tool does not ensure that only correct data or requirements are recorded. What it does do is support the recording and maintenance of data. Therefore, when tools are used, it is essential that all persons involved in the development process are sufficiently trained in documenting requirements correctly.
- It must be possible to use the tool to map **project-specific data models (requirements information models)**. Prior standardization of the data models may be necessary to ensure that tools can be exchanged.

Technical view

- **Data exchangeability**: In collaboration with other departments, partners, suppliers, etc., data exchangeability is a particular challenge because the exchange between heterogeneous requirements information models and different tools must be ensured. It must also be possible to export and import data for migration to a new tool or tool release.

2 Criteria for Selecting a Requirements Management Tool

Just like the introduction of any software in a company, the selection of a requirements management tool raises the question of the requirements. What should the tool support? How should the integration take place? Who should operate the software? Are there existing systems that have to be replaced? Is a migration necessary? And so on.

Here, literature offers a number of helpful reference points, checklists, and questions that can support you in introducing a requirements management tool ([PoRu2011a], [CNAT2011], [CNAT2012], [ISO24766], [Eber2012]).

[PoRu2011a] proposes a view-based process for considering the requirements from all relevant stakeholders for the tool (analog to introducing software).

- *Provider view*: Amongst other things, the provider view considers the market position and the service options (in the sense of training, user support, company-specific adjustments, etc.) that the tool provider offers. This view is necessary because the introduction of tool support generally means a longer-term tie.
- *User view*: The user view considers the requirements that result from the view of the different system users. These include, for example, requirements for role concepts, multiple user capability, etc.
- *Economic view*: The economic view considers the entire costs as a full costing required for the introduction and operation of the tool, and for the running costs for licenses and support, etc.
- *Product view*: The product view considers the functionality that the tool to be introduced requires in order to support requirements management. This includes, for example, requirements for attribute assignment, creation of views, and traceability.
- *Project view*: The project view considers the extent to which the tool can support future projects—for example, with regard to planning, reporting, etc.
- *Process view*: The process view considers requirements for the tool in terms of the methodological support, for example, through suitable workflows. However, caution must be taken here to ensure that it is not the tool that specifies the methodology.
- *Technical view*: The technical view considers requirements for operability, portability, scalability, integration of, for example, test tools in an existing tool landscape, as well as data exchange and data migration.

These views help you to define the requirements for your tool. Literature also offers numerous checklists for tool selection, see [CNAT2011], [CNAT2012], [ISO24766], [Eber2012].

In this section, we explicitly address the aspects of requirements management that must be supported for your company or project. In selecting these aspects, we therefore concentrate primarily on the points that should be considered in your requirements management plan.

In the previous chapters of this book, you have learned how to create a requirements management plan.

You have either defined a specific plan for a project or an abstract plan for requirements management in your company. To ensure that your requirements management plan is supported in the best possible way, looking back over the previous chapters and your requirements management plan, consider therefore, which criteria for tool selection are particularly important to you and consider, for example, the following questions in your evaluation:

- Does the tool support the implementation of your requirements information model?
    - Are the different types of requirements supported?
    - Are different requirements artifacts supported?
    - Are different forms of presentation supported?
    - Are different levels of detail supported?
    - Can the requirements documented in the tool be exported in a structured and readable form (e.g., as a requirements specification)?
- Does the tool support the creation of the required attributes and views?
    - Are different attributes supported for each requirement type?
    - Is the definition of value ranges for attributes supported?
    - Can multiple attributes be selected?
    - Can attribute value transitions be defined?
    - Is the user supported with automatic values (e.g., date of creation, creator) when entering information?
    - Can default values be defined for attributes?
    - Is there a differentiation between optional and mandatory attributes?
    - Are dependencies between attributes supported?
    - Can ad-hoc views be created?
    - Can views created be saved?
    - Can views be restricted using role concepts?
- Does the tool support the prioritization of requirements artifacts?
    - Are ad-hoc prioritization methods supported?
    - Are analytical prioritization methods supported?
    - Can a history be maintained for prioritization decisions?
- Does the tool support version control for requirements?
    - Are new versions of artifacts created automatically?
    - Can different versions be compared with one another?
    - Can the change reason be documented and traced?
    - Do changes to attributes lead to new versions of the artifact?
    - Can individual attributes be removed from the versioning?
    - Is it possible to roll back to old requirements versions?

  - Can requirements configurations be created?
  - Is it possible to roll back to old requirements configurations?
  - Is a comparison of requirements configurations possible?
  - Can requirements baselines be created?
  - Is it possible to roll back to old requirements baselines?
  - Is a comparison of requirements baselines possible?
- Does the tool support change management?
  - Can a change management process be defined?
  - Are change request templates offered or supported?
  - Can change requests be created and processed based on roles?
  - Is the processing and evaluation of change requests supported?
  - Can the change requests be subsequently placed in a relationship to the requirements to be changed through linking?
- Does the tool support the traceability strategy of the requirements management plan?
  - Is traceability between artifacts supported?
  - Can different relationship types be created?
  - Can relationship types to artifacts be restricted to prevent all relationship types being used in an uncontrolled way?
  - Is linking to predecessor and successor artifacts (goals and test cases) possible (keyword: tool integration)?
  - Is a role-based maintenance of traceability relationships supported or can any user create, change, or remove all relationships?
  - Is traceability between textual and model-based artifacts supported (where applicable, on a cross-tool basis)?
  - How can traceability relationships be presented (matrix, table, graph, etc.)?
  - Are impact analyses possible for changes, presenting the predecessor and successor artifacts to the user?
  - Over how many levels is an impact analysis possible?
  - Can evaluations of traceability relationships be created (e.g., number of relationships between test cases and requirements to the number of test cases and requirements)?

- Does the tool support the documentation of variability?
    - Is the explicit documentation of variability supported?
    - Is the implicit documentation of variability supported?
    - Are relationships between variation points and variants supported?
    - Is feature modeling supported?
    - Are orthogonal traceability models supported?
    - Is the derivation of specific products from the defined variability supported?
    - Is it possible to search for variants and variation points?
- Does the tool support reporting as part of requirements management?
    - Are there templates for defining reports?
    - Can own reports be created?
    - Is automated creation of reports (e.g., at certain points in time) supported?
    - Can reports be exported, for example as a PDF file?
    - Can reports be sent automatically?
    - Can reports be printed?
- Does the tool support the definition of requirements engineering processes?
    - Can workflows be defined for the defined requirements engineering activities (e.g., documentation, check, acceptance)?
    - Is the definition of roles, responsibilities, and (user) rights supported?
    - Can company-wide process models, which are adapted in individual projects, be mapped?
    - Is parallel and role-based work supported?
    - Are open item lists (and tasks) supported to document unclear points and tasks and assign them to specific persons?
    - Can decisions be documented (e.g., decision logs)?
    - Can requirements engineering processes be checked (target/actual comparison for process conformity)?
- Does the tool support agile methods?
    - Are storyboards and Kanban boards supported?
    - Are burndown charts supported?
    - Are product backlogs and sprint backlogs supported?
    - Are retrospectives supported?

Think about which of these points are relevant for you and weight the points for your project for tool introduction. Based on the requirements management plan, you can create a structured question list for your tool selection.

Using this list, evaluate the selected tools in terms of the requirements management functions to be supported. The important thing is that the tool satisfies your requirements engineering processes.

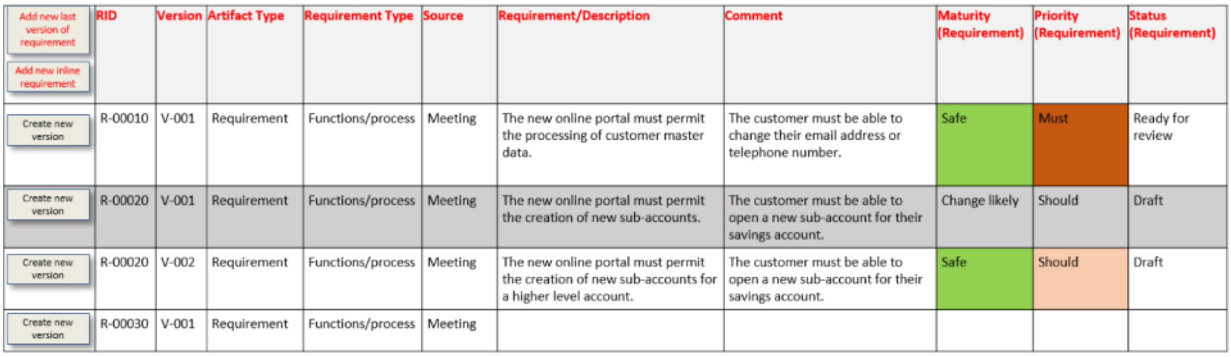
3 Analyzing Selected Tools Using the Requirements Management Plan Evaluation Criteria

In this chapter, based on the criteria introduced in the previous section, we will perform an example tool evaluation. To do this, we have intentionally dedicated our example to three very different classes of tools.

- Standard office applications
- Systems engineering tools
- Requirements management tools

It is not our intention to look closely at the actual result, and particularly not to create an independent evaluation of requirements management tools. Instead, we want to give you an insight into how these criteria can be applied and used for tool selection.

In the class of standard office applications, we will look at the most widespread tool, which is used worldwide to record and subsequently manage probably more than half of all requirements. even though these applications by far do not support the required properties of requirements management tools mentioned at the beginning [PoRu2011a] (Section 9.3). Nevertheless, this type of documentation and management, with some methodological and organizational guidelines for assigning attributes, versioning, and traceability, is better than no documentation. The massive advantage of these applications lies in the fact that they are widespread, that is, the initial existence, the existing user acceptance, as well as the advantage that almost everyone knows how to use these applications and the files can be exchanged easily between parties involved via email. Word, for example, offers the advantage that specifications can be structured exactly as required. With Excel, just a little knowledge also allows you to create attributes and views. Compared to Word, Excel has the decisive advantage that you can filter, sort, evaluate, etc. based on the defined attributes. To create a versioning, you can also use smaller macros to make life easier so that you can create new versions of a requirements artifact at the push of a button. Figure 49 presents an example of an Excel-based requirements list with different attributes. Here, the line highlighted gray reflects an old version of a requirement which is presented below as a current and revised version.

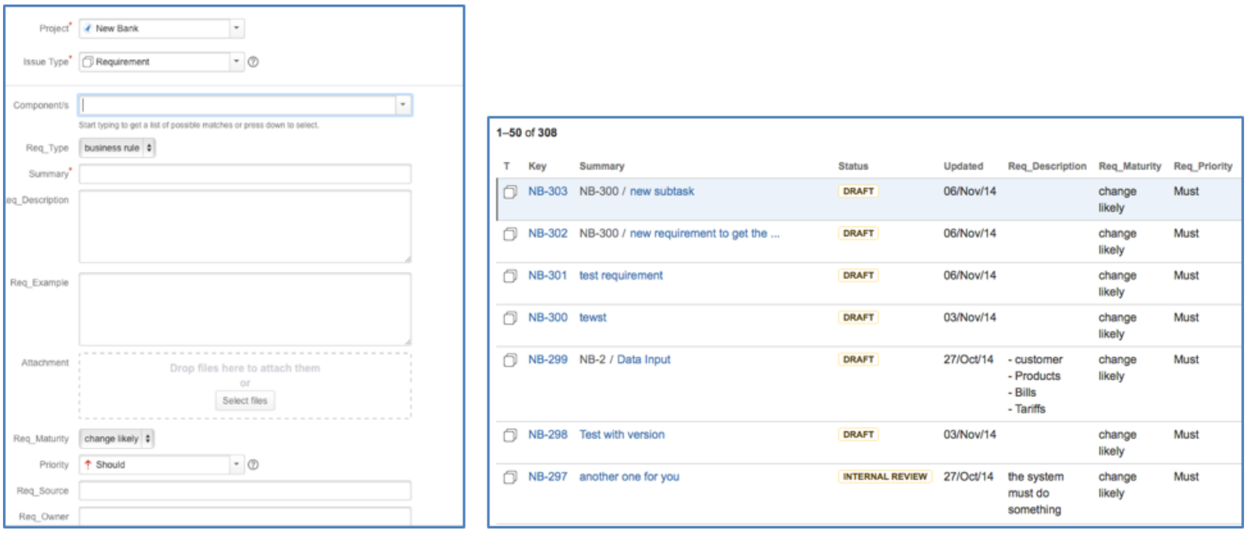


Add new last version of requirement Add new inline requirement	RID	Version	Artifact Type	Requirement Type	Source	Requirement/Description	Comment	Maturity (Requirement)	Priority (Requirement)	Status (Requirement)
Create new version	R-00010	V-001	Requirement	Functions/process	Meeting	The new online portal must permit the processing of customer master data.	The customer must be able to change their email address or telephone number.	Safe	Must	Ready for review
Create new version	R-00020	V-001	Requirement	Functions/process	Meeting	The new online portal must permit the creation of new sub-accounts.	The customer must be able to open a new sub-account for their savings account.	Change likely	Should	Draft
Create new version	R-00020	V-002	Requirement	Functions/process	Meeting	The new online portal must permit the creation of new sub-accounts for a higher level account.	The customer must be able to open a new sub-account for their savings account.	Safe	Should	Draft
Create new version	R-00030	V-001	Requirement	Functions/process	Meeting					

Figure 49: Screenshot of an Excel-based requirements list

However, creating structures and hierarchies is more difficult in table-based applications such as Excel than it is in Word. In Table 12: Analysis of selected tools: results table, we look at the capabilities of Word and Excel for requirements management in more detail.

The second class of tools considered here is also not a classic requirements management tool, but rather an application originally used for tracking issues. Using this class, the main thing we want to point out is that before you introduce a requirements management tool in your company, you should look left and right and check which other tools are already being used and could potentially even be used for requirements management. Through its adaptability, the issue tracking system Jira from Atlassian (www.atlassian.com) offers many more options than just tracking issues. In Jira, you can create different requirements artifacts with separate attributes, define workflows for status transitions, for example, link requirements artifacts to one another, and so on. In addition to the standard scope of functions, Jira offers plug-ins to support agile project methods such as scrum, for example. Furthermore, Confluence offers a web-based application for organizing documents, meeting minutes, decisions, or for creating reports and describing the overall project context. Confluence can be seamlessly integrated into Jira. The use of Jira and Confluence for requirements management is described in [Syra2014], for example. Figure 50 shows two screenshots in Jira: on the left, a template for creating requirements with the defined attributes, and on the right, an example view of requirements. Figure 51 shows an example of how links can be presented in Jira (Issue Links). In the example, the requirement is connected to two further requirements via the relationship type "blocks", and to two other requirements via the relationship type "relates to". These hyperlinks allow bidirectional navigation between the artifacts.

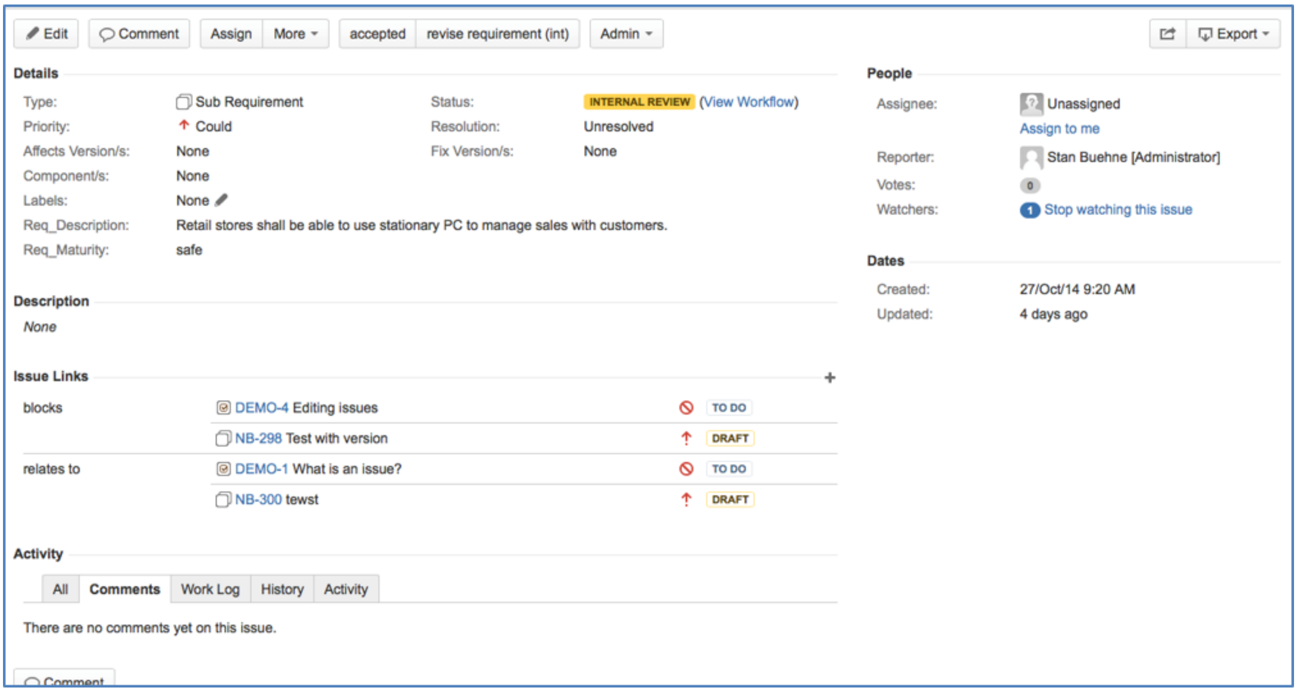


Project*	New Bank
Issue Type*	Requirement
Component/s	
	Start typing to get a list of possible matches or press down to select.
Req_Type	business rule
Summary*	
eq_Description	
Req_Example	
Attachment	Drop files here to attach them or Select files
Req_Maturity	change likely
Priority	Should
Req_Source	
Req_Owner	

1–50 of 308

T	Key	Summary	Status	Updated	Req_Description	Req_Maturity	Req_Priority
	NB-303	NB-300 / new subtask	DRAFT	06/Nov/14		change likely	Must
	NB-302	NB-300 / new requirement to get the ...	DRAFT	06/Nov/14		change likely	Must
	NB-301	test requirement	DRAFT	06/Nov/14		change likely	Must
	NB-300	tewst	DRAFT	03/Nov/14		change likely	Must
	NB-299	NB-2 / Data Input	DRAFT	27/Oct/14	- customer - Products - Bills - Tariffs	change likely	Must
	NB-298	Test with version	DRAFT	03/Nov/14		change likely	Must
	NB-297	another one for you	INTERNAL REVIEW	27/Oct/14	the system must do something	change likely	Must

Figure 50: Example template for recording requirements (left) and a status view of requirements (right)



Details

Type:	Sub Requirement
Priority:	Could
Affects Version/s:	None
Component/s:	None
Labels:	None
Req_Description:	Retail stores shall be able to use stationary PC to manage sales with customers.
Req_Maturity:	safe
Status:	INTERNAL REVIEW (View Workflow)
Resolution:	Unresolved
Fix Version/s:	None

Description

None

Issue Links

blocks	DEMO-4 Editing issues		TO DO
	NB-298 Test with version		DRAFT
relates to	DEMO-1 What is an issue?		TO DO
	NB-300 tewst		DRAFT

Activity

There are no comments yet on this issue.

People

Assignee:	Unassigned Assign to me
Reporter:	Stan Buehne [Administrator]
Votes:	0
Watchers:	1 Stop watching this issue

Dates

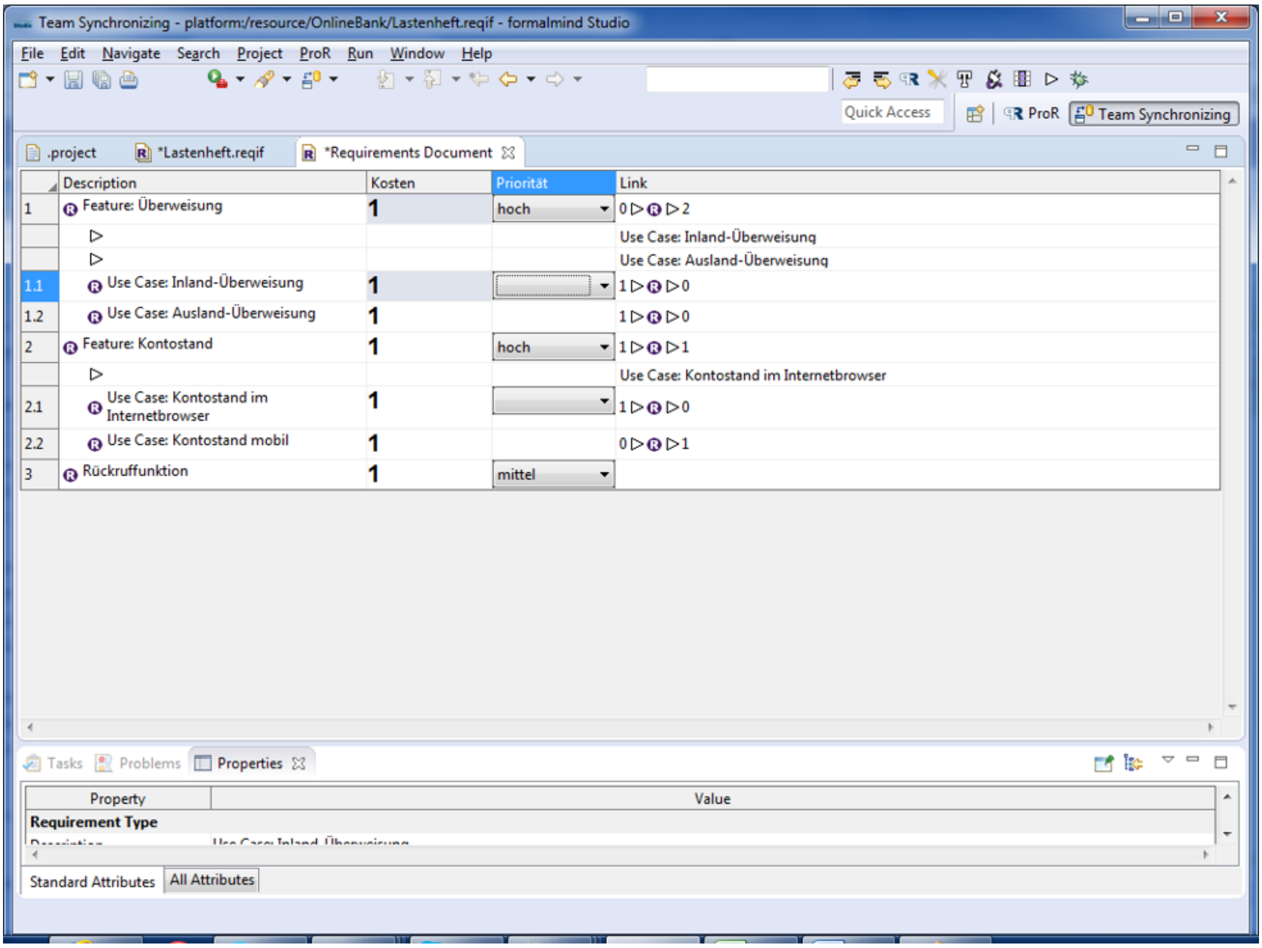
Created:	27/Oct/14 9:20 AM
Updated:	4 days ago

Figure 51: Example of traceability relationships for a requirement in Jira

Our third class of requirements management tools is dedicated to ProR/RMF, an open source tool for requirements engineering with Eclipse. The software is available to download free of charge, along with documentation on the tool, on the Internet at: http://eclipse.org/rmf, www.pror.org, or the download page here: http://eclipse.org/rmf/download.php. ProR normally requires the installation of Eclipse. There was a stand-alone version that did not need Eclipse. However, this is no longer supported by the RMF project (RMF stands for Requirements Modeling Framework). A stand-alone ProR variant that is still available to download free of charge, however, is formalmind Studio from Formal Mind GmbH: http://www.formalmind.com/studio. formalmind Studio contains enhancements called ProR Essentials that make work more efficient. Documentation for this tool can be found here: http://wiki.eclipse.org/RMF, and the RMF Guide here: http://download.eclipse.org/rmf/documentation/rmf-latex/main.html.

The tool is subject to constant further development. To find out how to implement your own ideas, see:
https://wiki.eclipse.org/RMF/Contributor_Guide/Presentations.

Figure 52 shows a screenshot of a hierarchical requirements list in formalmind Studio. The attributes themselves can be defined, as well as the types of links that establish traceability between requirements. As you can see in Table 12: Analysis of selected tools: results table, formalmind Studio currently supports primarily basic requirements management functionality. However, there are a number of Eclipse plug-ins that can be used to integrate the tool and enhance its functions.


Team Synchronizing - platform:/resource/OnlineBank/Lastenheft.reqif - formalmind Studio
File Edit Navigate Search Project ProR Run Window Help
Quick Access
ProR
Team Synchronizing
.project
*Lastenheft.reqif
*Requirements Document
Tasks
Problems
Properties

	Description	Kosten	Priorität	Link
1	Feature: Überweisung	1	hoch	0 ▷ Ⓡ ▷ 2
	▷			Use Case: Inland-Überweisung
	▷			Use Case: Ausland-Überweisung
1.1	Use Case: Inland-Überweisung	1		1 ▷ Ⓡ ▷ 0
1.2	Use Case: Ausland-Überweisung	1		1 ▷ Ⓡ ▷ 0
2	Feature: Kontostand	1	hoch	1 ▷ Ⓡ ▷ 1
	▷			Use Case: Kontostand im Internetbrowser
2.1	Use Case: Kontostand im Internetbrowser	1		1 ▷ Ⓡ ▷ 0
2.2	Use Case: Kontostand mobil	1		0 ▷ Ⓡ ▷ 1
3	Rückruffunktion	1	mittel	

Property	Value
Requirement Type	
Description	Use Case: Inland-Überweisung

Standard Attributes | All Attributes

Figure 52: Screenshot from formalmind Studio: Requirements hierarchy with two attributes (costs and priority), as well as links between the features and use cases.

Table 12: Analysis of selected tools: results table gives an overview of the evaluation. We have used only three values for the evaluation: "Yes" means that the tool supports the criterion completely; "Partially" means that the tool supports the criterion with some limitations; and "No" means that the tool does not support this criterion. The evaluation here is not intended to be a tool recommendation, but rather a criteria template for evaluating tools that has been applied for three non-typical requirements management tools by way of example.

	Excel/Word		JIRA/Confluence		ProR/formalmind Studio	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Does the tool support the implementation of your requirements information model?						
Are the different types of requirements supported?	Yes	No limitation, provided they can be presented textually	Yes	Via attributes	Yes	No limitation, provided they can be presented textually
Are different requirements artifacts supported?	Yes	No limitation, provided they can be presented textually	Yes	Via new issue types and own attributes and views	Yes	
Are different forms of presentation supported?	Partially	Text and templates	Partially	Text and templates	Partially	Text only, but elements of diagrams can be referenced, e.g., on integration with Rodin
Are different levels of detail supported?	Partially	Via document structures or attributes	Partially	Via sub-requirements and linking	Yes	Through hierarchical linking, which can be changed flexibly using drag & drop
Can the requirements documented in the tool be exported in a structured and readable form (e.g., as a requirements specification)?	Yes	Already recorded as a document	Partially	Standard export is purely table-based	Partially	As HTML file and as reqIF file
Does the tool support the creation of the required attributes and views?						
Are different attributes supported for each requirement type?	Yes		Partially	Dependent on the realization	Yes	You can define any number of requirement types, each with different attributes. These can be mixed within an artifact.
Is the definition of value ranges for attributes supported?	Yes	Value ranges can only be mapped usefully in Excel	Yes		Yes	Value ranges for numbers as well as value lists

	Excel/Word		JIRA/Confluence		ProR/formalmind Studio	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Can multiple attributes be selected?	No	Maximally as a value list	Yes	In different ways (checkbox, label, list selection)	Yes	For attributes for which value lists are defined
Can attribute value transitions be defined?	No	In Excel, could be defined as a maximum via a macro	Partially	For attributes such as the status, explicit statuses and transitions can be defined	No	
Is the user supported with automatic values (e.g., date of creation, creator) when entering information?	Partially	In Word and Excel, entries could be pre-labeled	Yes		Partially	Default values are supported
Can default values be defined for attributes?	Partially	In Word and Excel, entries could be pre-labeled	Yes		Yes	
Is there a differentiation between optional and mandatory attributes?	Partially	Possible only through special marking	Yes		No	
Are dependencies between attributes supported?	No		No		No	Possible using the Eclipse Validation Framework
Can ad-hoc views be created?	No		Yes		Partially	Yes, but always only one: attributes can be displayed or hidden, filtering by attributes is possible, sorting is not possible
Can views created be saved?	No		Yes		Yes	But always only one
Can views be restricted using role concepts?	No		Yes		No	

	Excel/Word		JIRA/Confluence		ProR/formalmind Studio	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Does the tool support the prioritization of requirements artifacts?						
Are ad-hoc prioritization methods supported?	No		No		No	
Are analytical prioritization methods supported?	No		No		No	
Can a history be maintained for prioritization decisions?	No		No		No	
Does the tool support version control for requirements?						
Are new versions of artifacts created automatically?	No	The only option is through tracking changes	Yes		No	Possible via an Eclipse plug-in for version control
Can different versions be compared with one another?	Partially	To a limited extent, via the track changes function in Word; no option in Excel	Yes		No	Possible via an Eclipse plug-in for version control
Can the change reason be documented and traced?	Partially	A reason can be recorded in pure text form at artifact and/or document level	Yes	Changes can be documented for example via comments or separate issue types	Partially	Would be possible in a separate attribute
Do changes to attributes lead to new versions of the artifact?	No	Manual versioning only	Yes		No	Possible via an Eclipse plug-in for version control
Can individual attributes be removed from the versioning?	No	Manual versioning only	No		No	Possible via an Eclipse plug-in for version control
Is it possible to roll back to old requirements versions?	Partially	To a limited extent, via the track changes function in Word; no option in Excel	No	Previous versions can only be presented, but not rolled back	No	Possible via an Eclipse plug-in for version control

	Excel/Word		JIRA/Confluence		ProR/formalmind Studio	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Can requirements configurations be created?	No	As a maximum, a requirements configuration can be created as a document version	No		No	Possible via an Eclipse plug-in for version control
Is it possible to roll back to old requirements configurations?	No		No		No	Possible via an Eclipse plug-in for version control
Is a comparison of requirements configurations possible?	No		No		No	Possible via an Eclipse plug-in for version control and EMF Compare
Can requirements baselines be created?	No	As a maximum, a baseline can be created as a document version	No		No	Possible via an Eclipse plug-in for version control
Is it possible to roll back to old requirements baselines?	No		No		No	Possible via an Eclipse plug-in for version control
Is a comparison of requirements baselines possible?	No		No		No	Possible via an Eclipse plug-in for version control and EMF Compare
Does the tool support change management?						
Can a change management process be defined?	No		Partially	If changes are created as a separate issue type, a workflow could be defined for this	No	
Are change request templates offered or supported?	No		Partially	If changes are created as a separate issue type, separate attributes for a change request are possible	No	

	Excel/Word		JIRA/Confluence		ProR/formalmind Studio	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Can change requests be created and processed based on roles?	No		Yes		No	
Is the processing and evaluation of change requests supported?	No		Partially	The processing can be reflected via a workflow; the evaluation itself must be manual	Partially	Change requests can be documented as requirements
Can the change requests be subsequently placed in a relationship to the requirements to be changed through linking?	No		Yes		Partially	Yes, if the change requests are managed as requirements
Does the tool support the traceability strategy of the requirements management plan?						
Is traceability between artifacts supported?	Partially	Only via the manual maintenance of textual references, hyperlinks, or matrices	Yes	Via linking of issue types	No	
Can different relationship types be created?	Partially	Possible in pure text form	Yes		Yes	
Can relationship types to artifacts be restricted to prevent all relationship types being used in an uncontrolled way?	No		No		No	
Is linking to predecessor and successor artifacts (goals and test cases) possible (keyword: tool integration)?	Partially	Only via manual textual references	Yes	If all artifacts are described in Jira	Yes	If all artifacts are stored in formalmind Studio or through integration with Rodin

	Excel/Word		JIRA/Confluence		ProR/formalmind Studio	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Is a role-based maintenance of traceability relationships supported or can any user create, change, or remove all relationships?	No				No	All users can do the same thing
Is traceability between textual and model-based artifacts supported (where applicable, on a cross-tool basis)?	Partially	Via textual references, URLs, and embedded objects	Partially	Via textual references, URLs, and attachments	Yes	There is an integration with the modeling tool Rodin
How can traceability relationships be presented (matrix, table, graph, etc.)?		In every form via manual effort		Via hyperlinks		Via hyperlinks
Are impact analyses possible for changes, presenting the predecessor and successor artifacts to the user?	No		Partially	Via hyperlinks	Partially	Via hyperlinks
Over how many levels is an impact analysis possible?		Only the directly linked artifact is ever visible		Only the directly linked artifact is ever visible		Only the directly linked artifact is ever visible
Can evaluations of traceability relationships be created (e.g., number of relationships between test cases and requirements)?	No		No		No	
Does the tool support the documentation of variability?						
Is the explicit documentation of variability supported?	No		No		No	

	Excel/Word		JIRA/Confluence		ProR/formalmind Studio	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Is the implicit documentation of variability supported?	Partially	Implicit documentation of variability can be supported via templates	Partially	Implicit documentation of variability can be supported via attributes	No	
Are relationships between variation points and variants supported?	No		No		No	
Is feature modeling supported?	No		No		No	
Are orthogonal traceability models supported?	No		No		No	
Is the derivation of specific products from the defined variability supported?	No		No		No	
Is it possible to search for variants and variation points?	No		Partially	If mapped by separate attributes	No	
Does the tool support reporting as part of requirements management?						
Are there templates for defining reports?	No	As a maximum, separate Word templates	No		No	
Can own reports be created?	Yes	Own reports can be created in Word and Excel	No		No	Views only
Is automated creation of reports (e.g., at certain points in time) supported?	No		No		No	

	Excel/Word		JIRA/Confluence		ProR/formalmind Studio	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Can reports be exported, for example as a PDF file?	Yes	From Office 2010, a document can be saved as a PDF	No		Yes	Only in HTML
Can reports be sent automatically?	No		No		No	
Can reports be printed?	Yes		No		Yes	In HTML format via the browser
Does the tool support the definition of requirements engineering processes?						
Can workflows be defined for the defined requirements engineering activities (e.g., documentation, check, acceptance)?	No		Yes		No	
Is the definition of roles, responsibilities, and (user) rights supported?	No		Yes		No	
Can company-wide process models, which are adapted in individual projects, be mapped?	Partially	Specification templates can be created via document templates	No		No	
Is parallel and role-based work supported?	No		Yes		No	
Are open item lists (and tasks) supported to document unclear points and tasks and assign them to specific persons?	No	Not with direct assignment; in principle, open item lists can of course be maintained	Partially	Via the creation of tasks, which are placed in a relationship to requirements	No	

	Excel/Word		**JIRA/Confluence**		**ProR/formalmind Studio**	
Criterion	**Eval.**	**Justification**	**Eval.**	**Justification**	**Eval.**	**Justification**
Can decisions be documented (e.g., decision logs)?	No		Partially	Via attachments to a requirement or linking to Confluence	No	
Can requirements engineering processes be checked (target/actual comparison for process conformity)?	No		No		No	
Does the tool support agile methods?						
Are storyboards and Kanban boards supported?	No		Yes	Via the Jira "Agile" plug-in, both Kanban and storyboards	No	
Are burndown charts supported?	No		Yes	Via the Jira "Agile" plug-in	No	
Are product backlogs and sprint backlogs supported?	No		Yes	Via the Jira "Agile" plug-in	No	
Are retrospectives supported?	No		Yes	Via the Jira "Agile" plug-in	No	

Table 12: Analysis of selected tools: results table

At this point, note again that it was not our intention to perform a comprehensive evaluation of tools or these selected tools; we have used these tools only to demonstrate our requirements management-based selection criteria in the application. Therefore, we apologize if we have estimated certain functions of one of the tools incorrectly.

Annex C (Earned Value Analysis)

The earned value analysis (also referred to as the value benefit analysis) [Kerz2003], [Wann2013a], [Wann2013b] follows the status of a project based on the progress of the results and the cost consumption. The progress (degree of completion or *earned value*) is compared with the planned progress (*planned value*) for this point in time and also with the budget consumed to date (*actual cost*).

With this method, you can detect deviations in the actual project progress compared to the schedule or the budget at an early stage. If the project is not on plan, then either measures must be taken to enable the project to be completed on plan, or the earned value analysis can be used to calculate the delivery delay and the cost overspend in advance.

The earned value analysis requires the following four key figures of the project:

- *Budget (budget at completion BAC)*: The budget available for the entire project. In the earned value analysis, the assumption is that this budget corresponds to the total costs and the *planned value* (value of benefit) of the project at the end of the project. If this is not the case, use the planned total costs for your calculation. Three factors are added to this total volume of the project, and the values for these three factors have to be determined at the respective time of reporting.
- *Planned degree of completion (planned value, PV)*: Here, you specify in % the proportion of the project that should be completed at the current point in time. The figure is calculated as the quotient of the planned work volume and the total volume of the project.
- *Degree of completion (earned value, EV)*: Here, you specify in % the proportion of the project that is actually completed at the current point in time.
- *Costs to date (actual cost, AC)*: Here, you specify the costs that have already arisen. The cost index is the quotient of the costs to date and the total budget of the project.

These key figures tell you what part of the result has already been completed and what proportion of the budget has been used to do so. You can therefore calculate whether the project is on schedule and whether work is being performed efficiently—that is, whether the result created matches the budget consumed. If the cost index matches the degree of completion, x% of the resources has been used to complete x% of the results and the project is within budget. If the degree of completion matches the planned degree of completion, the project is on schedule. Based on the size of any deviations from the plan, you can create forecasts of how late the project will probably be completed and what the cost overspend will be.

To check the *delivery reliability*, calculate as follows:

Compare the *earned value* with the *planned value* for the same point in time, that is, compare the actual degree of completion with the planned degree of completion. If both are the same, you are on plan. If the actual degree of completion is higher than the planned degree of completion, that is favorable as you will then probably finish earlier than planned. However, the opposite is the most common case: the actual degree of completion is lower than the planned degree of completion. The project is therefore behind schedule.

The delay gives rise to a late delivery date for the planned results. If the delivery date remains fixed in place, then the delay means that the project scope is restricted and part of the delivery scope may be delivered after the delivery date.

To check that *costs are within budget*, calculate as follows:

First, calculate the cost index. This is the quotient of the costs to date and the total budget of the project in %. Then compare the actual degree of completion with the cost index. The creation of 26% of the project result should consume a maximum of 26% of the total budget. However, if the degree of completion is smaller than the cost index, then the budget will probably be exceeded.

There are various approaches for *adapting the time and cost plans to reality*, that is, for anticipating the delay and budget overspend:

1. ***Keep the original plan***: The assumption is that the deviation that has already occurred has no effect and the delay or budget overspend can be recuperated. The end date and budget therefore remain the same. However, this optimistic hope is rarely fulfilled even with the most sophisticated of justifications.
2. ***Add the delay or increase in costs***: The delay or cost overspend that has already arisen is added to the planned values if you assume that the rest of the project will progress according to plan. This assumption should also be justified. Here, you calculate as follows:
   - **Date forecast**: If the actual degree of completion (e.g., 26%) has been achieved seven days later than planned, the delay is seven days. The overall project will be finished with a delay of seven days. Seven days are added to the end date.
   - **Cost forecast**: If, for example, 30% of the budget (= cost index) has been used to achieve the actual degree of completion (e.g., 26%), at the end, the project will probably consume 104% of the planned total budget. The probable total costs are calculated by multiplying the total budget (BAC) by 1.04.
3. ***Linear forecast***: This pessimistic assumption is usually the most realistic. The assumption is that if the first part of the project is already a certain percentage more expensive than estimated, there is a systematic error in the estimation and the remaining work will also be correspondingly more expensive.
   - **Date forecast**: If the actual degree of completion (e.g., 26%) has been achieved seven days later than planned, that is 33 days instead of 26 days, for example. The rule of three is then applied: if a 26% degree of completion corresponds to a duration of 33 days, how long will 100% take? The calculation is as follows: 33 days x 100%/26% = 127 days. A duration of 100 days was originally planned. The delay is therefore 27 days.

The same result is obtained from an easier calculation: 26% degree of completion corresponds to a delay of 7 days, 100% degree of completion is therefore 7 days x 100%/26% = 27 days delay. These 27 days are added to the original delivery date to calculate the probable delivery date. The result is almost an entire month.

- **Cost forecast**: If, for example, 30% of the budget (= cost index) has been used to achieve the current degree of completion (e.g., 26%), the rule of three calculation is as follows: if 26% degree of completion uses 30% of the budget, 100% degree of completion is calculated as 30% of the budget x 100%/26% = 115.4%. This means that at the end, the project costs will probably be 115.4% of the previously planned total budget. Therefore, in the worst case, the small overspend to date will add up to a much larger amount at the end of the project.

The best way to perform an earned value analysis for the requirements engineering and requirements management activities is by using the requirements management tool, because the requirements management tool primarily contains the information about the requirements and their status. However, if the requirements management tool manages the entire development process (in the sense of requirements-based project management), an earned value analysis can be performed for the entire project with the data from the requirements management tool. For this purpose, the status attribute of the requirements must map the entire lifecycle, for example.

To support the earned value analysis with a requirements management tool, the tool must support the management of the following content:

- For each requirement, its entire lifecycle is mapped and managed in a status attribute—that is, from the elicitation, through the agreement on a specific release, through implementation, testing, and delivery. A degree of completion is assigned to each status value, as shown in Table 13, for example.
- For each requirement, its planned implementation effort is defined in an attribute "Effort". These efforts are then used as weighting factors for the determination of the actual degree of completion of the overall project. On its own, the number of requirements completed does not correspond to the degree of completion as each requirement has a different implementation effort.
- For an earned value report, the percentage degree of completion is calculated respectively for each requirement, as shown in Table 13, for example. The degree of completion is then determined as a weighted average of the degrees of completion of all requirements, whereby the requirements are averaged according to their planned budget. Requirements with a high effort therefore have a heavier weighting. If this is not possible, as an alternative, only the completed requirements can be calculated as completed (100%), and all others are calculated as incomplete (0%).

  The calculation and all other analyses remain the same. The formula for the degree of completion remains the same, and the same applies for all analyses. However, when defining the planned value for the degree of completion, you must also consider how the actual degree of completion will subsequently be determined.

If only the completed requirements are evaluated as "finished", at the beginning of the project there will initially be no measurable progress. In contrast, at the end of the project, a lot of requirements will be finished within a short time. However, this type of plan progression does not allow deviations from the plan to be detected so early. We therefore recommend a proportionate consideration of requirements in progress.

- To determine the actual budget consumed at any point in time, the costs incurred to date must be managed in an attribute for each requirement. However, this type of requirements-based cost recording in a requirements management tool could, depending on the company, lead to a duplicate recording and could appear pedantic to the developers. Therefore, the budget consumed can also be determined from another tool (time recording, project management tool, Controlling) at any point in time.

Additional information that is not requirements-based but is used for the earned value report is the planned degree of the completion of the project at a point in time and the total budget for the project (*budget at completion*), which must also be managed in a tool if the earned value report is calculated.

The following table (Table 13) shows the degrees of completion of requirements, dependent on the status, according to various authors. [RuSo2009] evidently refers only to the elicitation, documentation, and agreement process for the requirement. Therefore, the degree of completion of 100% corresponds to the status "Approved", while in [Eber2012], this status corresponds to 0% because here, the project progression is considered after the requirement approval. The percentages in the left column are therefore suitable for an earned value analysis of requirements engineering, while the status in the middle column is suitable for the earned value analysis of the implementation. We want to cover the entire project cycle, and therefore we propose a further scale on the far right.

To support the earned value analysis, for each requirement, the attributes must be included and maintained in the attribute schema, as presented in Table 13. "Planned costs", that is, the costs estimated for each requirement, are entered once. For each point in time to be considered, the time-specific values "Planned value", "Actual costs", and "Status" must be maintained. The degree of completion of the requirement is calculated automatically from the respective status that a degree of completion is assigned to (see Table 14). The value achieved for a requirement is calculated from the degree of completion multiplied by the effort. The values for the overall project are therefore calculated as follows:

- **Budget at completion** = total of planned costs for all requirements
- **Actual costs** = total of the actual costs for all requirements
- **Status**: determined manually as the result of the earned value analysis and its forecasts, but also the evaluation of the project manager with regard to whether delays that have occurred can be compensated for
- **Completion**: the degree of completion of the project is the weighted average value of the degrees of completion of the requirements, weighted by effort. In this example: (€4000 x 50% + €2000 x 10% + €2000 x 100% + …)/€30000. The result is a percentage.
- **Value achieved** = total of the values achieved for all requirements

Requirement Status	[Rupp & Sophist 2004]	[Eber2012]	[RuSo2009]	Our Proposal
	Degree of completion in relation to the elicitation, documentation, and approval process	Degree of completion in relation to implementation	Degree of completion in relation to the overall project	Degree of completion in relation to the overall project
Created	0 %		20 %	10 %
Signed off	30 %		30 %	
Acceptance criteria complete	60 %			
Consistent with object model	75 %			
Consistent with prototype	90 %			
Verified/checked	95 %		40 %	20 %
Approved/agreed/released	100 %	0 %	50 %	25 %
Drafted			60 %	
In implementation		10 %		50 %
Implemented		50 %	80 %	70 %
Tested/completed		100 %	100 %	100 %

Table 13: The degrees of completion of requirements, dependent on the status

Date: Today	Effort/Planned Costs	Actual Costs	Status	Completion	Value Reached
Requirement 1	4,000 €	1,800 €	In implementation	50 %	2,000 €
Requirement 2	2,000 €	150 €	Created	10 %	200 €
Requirement 3	2,000 €	2,100 €	Completed	100 %	2,000 €
…					
Overall project	**30,000 €**	**9,500 €**	**Yellow**	**30 %**	**9,000 €**

Table 14: Assignment of attributes for the requirements, to support the earned value analysis