

Syllabus

REQB®

Certified Professional for Requirements Engineering

Foundation Level



**Requirements
Engineering**
Qualifications Board

Version 1.3

2011

The copyright® to this edition of the syllabus in all languages is held by the
Global Association for Software Quality, gasq.

Overview of Changes

Version	Date	Comment
0.1	Apr. 17, 2006	First version of the syllabus; creation of a basic structure for the syllabus
0.2	Jul. 20, 2006	Expansion to Version 0.1
0.3	Sep. 4, 2006	Further expansion and revision of Version 0.1
0.4	Oct. 10, 2006	Revised Version 0.3
0.5	Dec. 15, 2006	Revised Version 0.4
0.6	Feb. 7, 2007	Completely revised Version 0.5
0.7	Apr. 10, 2007	Revised version for review
0.8	Jun. 15, 2007	Alpha version
0.9	Sep. 1, 2007	Beta version
1.0	Jan. 15, 2008	Released version 1.0
1.1	Mai 29, 2008	Updated version 1.1
1.2	Jul. 1., 2008	Updated version 1.2
1.3	Jul, 10, 2011	Updated version 1.3

Main Idea

The central theme for this syllabus was that the complexity of software and our dependency on software continues to increase. The result is a high level of dependency on the freedom from error in the software. The Requirements Engineering Qualifications Board (REQB) has therefore decided to create uniform international standards in the area of Requirements Engineering. Standards are like languages - it is only if you understand them that you can work effectively. In order to now create such a uniform language in this important area of requirements engineering, international experts got together in REQB and developed this syllabus.

Acknowledgements

Requirements Engineering Qualifications Board Working Party Foundation Level (Edition 2011): (Karolina Zmitrowicz (chair), Alain Betro, Dorothée Blocks, Jérôme Khoualed, Eric Riou Du Cosquer, Chris Hofstetter, Michał Figarski, Francine Lafontaine, Beata Karpińska, Folke Nilsson, Ingvar Nordström, Alain Ribault, Radosław Smilgin)

Table of contents

Table of contents.....	4
Introduction.....	9
1 Basics (K2).....	11
1.1 Requirement (K2).....	12
1.1.1 Definition and classification (K2).....	12
1.1.2 Problems with requirements (K1).....	14
1.1.3 Quality criteria for requirements (K2).....	14
1.1.4 Solution (K1).....	15
1.1.5 Commitment (K1).....	15
1.1.6 Legal responsibilities and faults (K1).....	16
1.1.7 Priority and criticality of requirements (K1).....	16
1.1.8 Validation and verification (K1).....	17
1.1.9 Requirements Engineering, Requirements Management and Requirements Development (K2).....	18
1.2 Standards and norms (K1).....	19
1.2.1 Standards (K1).....	19
1.2.2 Process norms (K1).....	20
1.2.3 The reasons of neglecting Requirements Engineering (K2).....	21
2 Process models and Requirements Engineering Process (K2).....	22
2.1 Process Models (K2).....	23
2.1.1 Process models (K2).....	23
2.1.2 General V Model (K2).....	24
2.1.3 Rational Unified Process (RUP©) (K2).....	24
2.1.4 Agile Approaches.....	25
2.1.5 Extreme Programming (K2).....	25
2.1.6 Scrum (K2).....	26
2.1.7 Maturity Model (K2).....	27
2.2 Requirements Engineering Process (K2).....	29

2.2.1	Definition of Requirements Engineering Process (K2).....	29
2.2.2	Influences of Requirements Engineering.....	29
3	Project and Risk Management (K2)	31
3.1	Project Management (K2).....	32
3.1.1	Necessity of Requirements Engineering in projects (K2).....	32
3.1.2	What errors can occur in Requirements Engineering? (K2)	33
3.2	Risk Management (K2).....	34
3.2.1	The necessity of Risk Management (K2).....	34
3.2.2	Risk (K2)	34
3.2.3	Risk Management (K2).....	35
3.2.4	Failure Mode and Effects Analysis (K2)	36
4	Responsibilities and Roles (K2)	38
4.1	Basic Roles (K1)	39
4.1.1	Basic roles (K2).....	39
4.1.2	Stakeholder (K2)	40
4.2	Tasks of Requirements Engineering (K2)	42
4.2.1	Tasks of Requirements Engineering (K2)	42
4.2.2	Knowledge of a Professional for Requirements Engineering (K1).....	42
5	Identification of Requirements (K2)	43
5.1	Customer (K1)	44
5.1.1	Customer (K1).....	44
5.1.2	Contract (K2).....	44
5.2	Project Visions and Goals (K2)	46
5.2.1	Vision (K2).....	46
5.3	Identifying Stakeholders (K2).....	48
5.3.1	Identifying Stakeholders (K2).....	48
5.3.2	The procedure of identification and evaluation of stakeholders (K2).....	48
5.4	Techniques for Identifying Requirements (K2)	49
5.4.1	Purpose of the identification of requirements (K2)	49
5.4.2	Techniques (K1)	49
5.5	Functional and Non-functional Requirements (K2)	55
5.5.1	Functional requirements (K2).....	55

5.5.2	Non-functional requirements (K2).....	55
5.6	Description of Requirements (K2).....	57
5.6.1	Description of Requirements (K2)	57
5.6.2	Procedure for Requirement Construction (K3).....	57
5.6.3	Requirement document (K2)	59
6	Specification of Requirements (K2)	60
6.1	Specification (K2)	61
6.1.1	Specification (K1)	61
6.1.2	Requirements Specifications (K2).....	61
6.1.3	User stories (K2).....	62
6.1.4	Solution Specifications (K2)	62
6.1.5	Important standards (K1).....	63
6.2	Procedure (K3)	64
6.2.1	Procedure of Solution Specification (K3).....	64
6.3	Formalization (K2)	65
6.3.1	Degrees of formalization (K2)	65
6.4	Quality of Requirements (K2).....	66
6.4.1	Background	66
6.4.2	Measures for quality improvement and quality assurance of requirements (K2)	66
7	Requirements Analysis (K2)	68
7.1	Requirements and Solutions (K1).....	69
7.1.1	Goal of the Requirements Analysis (K2)	69
7.1.2	Procedure of the Requirements Analysis (K2).....	69
7.1.3	Structural break between requirements and solutions (K2)	69
7.2	Methods and Techniques (K2)	70
7.2.1	Analysis methods and models	70
7.2.2	Types of models (K2)	71
7.2.3	Different perspectives of the system (K2)	71
7.2.4	Different models (K1)	72
7.3	Object-Oriented Analysis (K2).....	74
7.3.1	UML (K1)	74
7.3.2	SysML (K2).....	76

7.4	Cost Estimates (K2)	77
7.4.1	Types of estimates (K2).....	77
7.4.2	Influences on the development costs (K2)	77
7.4.3	Cost estimation approaches	78
7.5	Prioritization (K2)	82
7.5.1	Prioritization (K2).....	82
7.5.2	Procedure of prioritization (K2).....	82
7.5.3	Prioritization scale (K2).....	83
7.6	Agreeing on Requirements (K2)	84
7.6.1	Agreement (K2).....	84
7.6.2	Advantages of requirements' agreement (K1)	85
8	Tracking of Requirements (K2)	86
8.1	Tracing within the Project (K2).....	87
8.1.1	Evolution of requirements (K1)	87
8.1.2	Traceability (K2)	87
8.1.3	Types of traceability (K2)	88
8.2	Change Management (K2)	89
8.2.1	Changes of requirements (K1)	89
8.2.2	Change Management (K2)	89
8.2.3	Change Request (K2).....	90
8.2.4	Change Control Board (K1)	90
8.2.5	Life cycle of a requirement (K2).....	91
8.2.6	Distinction between Defect Management and Change Management (K2)	91
8.2.7	Impact of a change on the project (K2)	91
9	Quality Assurance (K2).....	93
9.1	Influencing Factors (K1)	94
9.1.1	Influences on the Requirements Engineering (K1).....	94
9.2	Verification of requirement in requirements' elicitation stage (K2)	95
9.3	Quality Assurance through Testability (K2).....	96
9.3.1	Requirements Engineering and testing (K2)	96
9.3.2	Acceptance Criteria (K2)	96
9.3.3	Methods for testing (K2).....	96

9.3.4	Requirements and test process (K2).....	97
9.4	Metrics (K2).....	98
9.4.1	Metric (K1).....	98
9.4.2	Metrics for requirements (K1).....	98
9.4.3	Measurement of the requirements quality (K2).....	99
10	Tools (K2).....	100
10.1	Advantages of Tools (K2).....	101
10.1.1	Uses of Tools in Requirements Engineering (K2).....	101
10.1.2	The advantages of using tools (K2).....	101
10.2	Categories of Tools (K2).....	102
10.2.1	Categories of Tools (K2).....	102
11	Literature.....	104
12	Index.....	107

Introduction

Purpose of the Syllabus

This syllabus defines the Foundation Level of the training program to become a REQB Certified Professional for Requirements Engineering (short form CPRE). REQB developed this syllabus in cooperation with the Global Association for Software Quality. The subject of scope of REQB is all types of IT-related products where a product can include HW and services as well as software and their commitment requirements, business requirements with related documentation.

The syllabus is to serve as a foundation for training providers who are seeking accreditation as teachers. All areas of this syllabus must correspondingly be incorporated in the training documents. The syllabus should, however, also serve the student as preparation material for certification. All areas listed here are thus relevant for the examination, which can be taken either after completion of an accredited courses or independently in open examination.

The syllabus provides also a recommended instruction time for each chapter.

Examination

The examination to become a Certified Professional for Requirements Engineering Foundation Level is based on this syllabus. All sections of this syllabus can thereby be tested. The examination questions are not necessarily divided into the individual sections. A question may refer to several sections.

The format of the examination is Multiple Choice.

Examinations can be taken after having attended accredited courses or in open examination (without a previous course). You will find detailed information regarding examination times on the website of gasq (www.gasq.org) or on REQB's website (www.reqb.org).

Accreditation

Providers of a REQB Certified Professional for Requirements Engineering Foundation Level Course must be accredited by the Global Association for Software Quality. Their experts review the training provider's documentation for accuracy. An accredited course is regarded as conforming to the syllabus. At the end of such a course, an officially Certified Professional for Requirements Engineering examination (CPRE exam) may be carried out by an independent certification institute (according to ISO 17024 rules).

Accredited Training Providers can be identified by the official REQB Accredited Training Provider logo:



Internationality

This syllabus was developed in cooperation between several international experts. The content of this syllabus can therefore be seen as an international standard. The syllabus thereby makes it possible to train and examine internationally at the same level.

K Levels

The learning objectives of this syllabus have been divided into different cognitive levels of knowledge (K-levels). This makes it possible for the candidate to recognize the "knowledge level" of each point.

There are 3 K-levels in the current syllabus:

- K1 - remember, recognize, recall
- K2 - understand, explain, give reasons, compare, classify, summarize
- K3 - apply in a specific context

1 Basics (K2)	40 minutes
----------------------	-------------------

Learning Objectives for Foundation Level of Requirements Engineering

The objectives identify what you will be able to do following the completion of each module.

1.1 Requirement (K2)

- LO-1.1.1 Recall the definition of a requirement (K1)
- LO-1.1.2 Explain the meaning and purpose of requirements (K2)
- LO-1.1.3 Explain how requirements can be classified (K2)
- LO-1.1.4 Describe the different types of requirements (K2)
- LO-1.1.5 Explain what problems there are concerning requirements (K2)
- LO-1.1.6 Describe what concepts are important in connection with requirements (K2)
- LO-1.1.7 Explain the difference between RM (Requirements Management) and RE Requirements Engineering (K2)

1.2 Standards and norms (K1)

- LO-1.2.1 Recall important norms and standards for Requirements Engineering (K1)
- LO-1.2.2 Explain why Requirements Engineering is important (K2)

1.1 Requirement (K2)	20 minutes
-----------------------------	-------------------

Terms

Commitment, Criticality, Functional Requirement, Non-functional Requirement, Requirement, Requirements Engineering, Requirements Management, Process Requirements, Product Requirements, Priority, Solution, Validation, Verification

1.1.1 Definition and classification (K2)

Definition of what is meant by the term “A requirement” (K1):

Requirement [IEEE 610.12]:

- A condition or capability needed by a user to solve a problem or achieve an objective.
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
- A documented representation of a condition or capability as in (1) or (2).

The meaning and purpose of requirements (K2):

- Foundation for assessment, planning, execution and monitoring of the project activity
- Customer expectations
- Component of agreements, orders, project plans etc.
- Setting of system boundaries, scope of delivery, contractual services

Classification of requirements [Ebert05] (K2)

Requirements consist of:

- Process requirements
- Product requirements

Process requirements are those related to the development and delivery processes. Examples include: costs, marketing, processing time, sales and distribution, organization, documentation.

Product requirements consist of functional and non-functional product requirements. Both can be regarded from the point of view of the user (external) or customer and from the point of view of the development team (internal). It is important to remember that user and customer can be different.

Functional requirements describe the function (behaviour) of the system.

Non-functional requirements describe the quality attributes of the system. They are often called “quality attributes”.

The difference between functional and non-functional requirements can be expressed by the following statements:

- Functional requirements describe *what* the system does.
- Non-functional requirements describe *how* the system does.

Examples:

Functional product requirements from the user's and customer's point of view: user interface, applications, services.

Functional product requirements from the development team's point of view: architecture, power supply, load distribution.

Non-functional product requirements from the point of view of the user and customer's: reliability, performance, usability

Non-functional product requirements from the point of view of the development team: testability, serviceability, tools

Basic types of requirements (K1):

- Customer requirements
 - Customer's desires, needs and expectations (high level business requirements)
 - Limitations on business
- Solution/system requirements
 - Specification of customer's needs (detailed specification of high level business requirements)
- Product/component requirements
 - Functions and characteristics of the solution
 - Basis for detailed analysis and design (example: system use cases)

1.1.2 Problems with requirements (K1)

Most common problems with requirements are:

- Unclear objectives
- Communication problems
- Language barriers
- Knowledge barriers
- Vague formulation
- Too formal formulations
- Volatility of the requirements
- Bad quality of the requirements (inc. ambiguous, overly specified, unclear, impossible, contradictory requirements)
- Gold plating (too much description on a requirement what causes the real requirement is covered by unnecessary description)
- Insufficient user involvement
- Overlooked user classes (resulting from missing stakeholders)
- Inaccurate planning
- Minimal specification

1.1.3 Quality criteria for requirements (K2)

Quality criteria for requirements [Wiegers05] are the following (K2):

1. Each requirement must be:
 - Correct – the requirement must accurately describe the functionality to be provided. The point of reference to evaluate the correctness is the source of the requirement (for example, customers or a higher-level system requirement).
 - Feasible – the requirement must be possible to implement within the known capabilities and/or limitations of the system and the environment.
 - Necessary – the requirement should document what the customer (or other stakeholders) really need and what is required in order to accomplish an external requirement or interface or a specified standard.

- Prioritized – the requirement should have a priority assigned indicating how essential it is for a particular product release.
 - Unambiguous – the requirement should be interpreted only in one way. Different readers of a requirement should have the same interpretation and understanding of a requirement.
 - Verifiable – the requirement should be possible to verify if it is implemented correctly.
 - Singular – does not contain multiple requirements; implies a sufficient granularity to specify a single requirement.
 - Design independent – describes “What”, not “How”
2. The requirements specification must be:
- Complete – no requirements or necessary information should be missing in the requirements specification. Completeness is also expressed as a desired characteristic of an individual requirement and the level of detail.
 - Consistent – the requirement cannot conflict with other software requirements or with higher level (system or business) requirements.
 - Modifiable – the specification must allow introducing changes in the requirements. A history of changes made to each requirement should be maintained.
 - Traceable – each requirement should be possible to link to its source (for example, a higher-level system requirement, a use case, or a customer’s statement) and related implementation artefacts (for example, design elements, source code, and test cases).

1.1.4 Solution (K1)

Solution (K1)

A solution is the implementation of the requirements.

Solution may be a software system, process’s improvement etc.

1.1.5 Commitment (K1)

Commitment (K1)

Commitment is the degree of obligation of meeting the requirement.

The commitment is defined through key words allocated to high level requirements:

- The system should...

Key words can include: “must”, “will”, “should”, “would”, “could”.

Key words “must”, “should”, “would”, “could” relate to business and user requirements before agreement. After agreeing and baselining the requirements key words should determine the degree of obligation stricter:

- The system will...

The level of obligation of a requirement can be expressed using MoSCoW notation (Must have, Should have, Could have, Would have).

Once the solution provider and the customer reach an agreement, commitment to the requirements is obtained from project participants.

Requirements may evolve throughout the project. As requirements evolve, obtaining commitment to requirements from project participants ensures that project participants commit to the current and approved requirements and the resulting changes in project plans, activities, and work products.

1.1.6 Legal responsibilities and faults (K1)

Legal responsibilities (K1)

There are legal responsibilities related to the quality of the software. Legal responsibilities are often related to a specific requirement (i.e. environmental requirement for a nuclear plant, an airplane) that must be satisfied in the delivered product. The responsibilities may also relate to defects in the product.

Legal responsibilities should be defined in the contract between the vendor and the customer. Some industries may also be required to meet contractual or legal requirements, or industry-specific standards.

Fault (defect) (K1)

A fault is a flaw in a component or system that can cause the component or system to fail to perform its required function, e.g. an incorrect statement or data definition.

A defect, if encountered during execution, may cause a failure of the component or system [ISTQB].

1.1.7 Priority and criticality of requirements (K1)

Priority of requirements (K1)

Priority is an evaluation of the importance/urgency of a requirement.

According to [SWEBOOK], in general, the higher the priority, the more essential the requirement is for meeting the overall goals of the software. Often classified on a fixed-point scale such as

mandatory, highly desirable, desirable, or optional, the priority often has to be balanced against the cost of development and implementation.

Examples of requirement's priorities:

- High
- Medium
- Low

Criticality of requirements (K1)

Evaluation of the risk of a requirement by evaluating the damage in case of non-fulfillment of a requirement.

The criticality is expressed in levels; the higher a level is the more severe the consequences are in case of a functional failure.

1.1.8 Validation and verification (K1)

Validation is a process of confirmation that the specification of a phase or the entire system fulfills the customer's requirements.

Validation is usually conducted with the support of a customer and aims to confirm that the requirements or the requirements specification described what the customer needs.

According to CMMI, validation activities demonstrate that a product or product component fulfills its intended use when placed in its intended environment. So, you know that "you built the right thing". Customers often identify product descriptions or requirements in a weak manner and validation helps to understand what is needed (by using tools as scenarios, use cases, prototyping, etc.) Verification is a comparison of an intermediate product with its specifications. It is thereby determined if the software was developed correctly and if the specifications that were determined during the previous phase were fulfilled.

According to CMMI, verification provides checkpoints at which selected deliverables or intermediate products are verified to confirm that they meet their requirements. Verification activities focus on incremental confirmation of the implementation of requirements; they enable early and ongoing confirmation of building the product right.

Most common techniques for verification and validation are reviews, audits, checklists, and testing.

The difference between validation and verification can be expressed by the following:

- Verification – did we create the product correctly?
- Validation – did we create the correct product?

1.1.9 Requirements Engineering, Requirements Management and Requirements Development (K2)

Delineation between Requirements Management, Development and Engineering (K2)

Requirements Engineering (RE) is a sub-discipline of Software Engineering, focused on determining and managing the requirements of hardware and software systems. Requirements Engineering encompasses Requirements Management and Requirements Development.

Requirements Engineering discipline involves the following sub-processes: requirements elicitation, analysis and negotiation (including requirements prioritization), specification, system modeling, requirements validation.

These sub-processes can overlap, for example system modeling can be a part of both analysis and specification, even in some cases elicitation.

Requirements Management constitutes a working framework for Requirements Engineering and interfaces with other processes like project management, configuration management and quality management.

The purpose of Requirements Management is to manage requirements of the project's products, and components, to ensure alignment between those requirements, the project's plans and work products throughout a project's products lifecycle (development cycle and maintenance cycle)

Requirements Management (RM) includes processes for the overall management of requirements. It is a continuous process of documenting, analyzing, tracing, prioritizing, communicating, agreeing on requirements and managing requirements' changes.

Requirements Development is a collection of activities, tasks, techniques and tools to identify, analyze and validate requirements. Includes the process of transforming needs into requirements.

The purpose of Requirements Development is to elicit, analyze, establish and validate customer, product, and product component requirements.

1.2 Standards and norms (K1)	20 minutes
-------------------------------------	-------------------

In order to pass the examination, it is not necessary to know the contents of all the norms. It is, however, important (K1) to know which norms are of importance for Requirements Engineering.

1.2.1 Standards (K1)

ISO 9000:

Requirements of a Quality Management System:

- Defined concepts and basics of a QMS
- Domain or industry neutral

ISO 9126 (replaced by ISO/IEC 25000):

Defines a quality model with six categories: functionality, reliability, usability, efficiency, maintainability, portability

IEEE 610:

Standard Glossary of Software Engineering Terminology

IEEE 830:

Recommended Practice for Software Requirements Specifications

IEEE 1233:

Guide for Developing of System Requirements Specifications

IEEE 1362:

Guide for Information Technology – System Definition

SWEBOK - The Guide to the Software Engineering Body of Knowledge (known as ISO Technical Report 19759):

SWEBOK describes generally accepted knowledge about software engineering. Its 10 knowledge areas summarize basic concepts and include a reference list pointing to the detailed information.

1.2.2 Process norms (K1)

ISO 12207:

Standard for Software Life Cycle Process

ISO 15288:

System Life Cycle Process

ISO 15504:

Software Process Improvement and Capability Determination (SPICE)

1.2.3 The reasons of neglecting Requirements Engineering (K2)

Requirements Engineering is of vital importance. And yet it is neglected time and again.

The reasons of neglecting Requirements Engineering may be the following (K2):

- High time pressure
- An exclusive orientation toward fast results
- An exclusive fixation on costs
- Considering only functional requirements
- Misinterpretations (many things are seen as given) and lack of understanding of the importance of Requirements Engineering for the success of a project

Possible consequences of neglecting Requirements Engineering (K2):

- Requirements become imprecise
- Requirements are ambiguous
- Requirements are contradictory
- Requirements that often change during the software development
- Requirements that do not fulfill the criteria
- Requirements that can be interpreted differently
- Missing requirements

2 Process models and Requirements Engineering Process (K2)	60 minutes
---	-------------------

Learning Objectives for Foundation Level of Requirements Engineering

The objectives identify what you will be able to do following the completion of each module.

2.1 Process models (K2)

- LO-2.1.1 Describe the different process models (K2)
- LO-2.1.2 Explain how the different process models differ (K2)

2.2 Requirements Engineering Process (K2)

- LO-2.2.1 Describe the characteristics of the Requirements Engineering process (K2)
- LO-2.2.2 Explain the phases of the Requirements Engineering process (K2)

2.1 Process Models (K2)	30 minutes
--------------------------------	-------------------

Terms

Extreme Programming, Process models, Product life cycle, Rational Unified Process, V Model

2.1.1 Process models (K2)

Process models are method-independent process description of development processes.

Roles, activities, phases and documents are thereby taken into account.

A software process model gives a standard format for planning, organizing and running a software project.

Product life cycle (PLC) (K2)

Defines various phases of the product development.

Basic phases are:

1. Planning
2. Development
3. Maintenance
4. End of life

The planning phase includes: vision, strategy, business plan, and cost benefit analysis.

The development phase includes: specification, draft, and implementation. The development phase is often divided into the following four phases:

- Analysis
- Design
- Implementation
- Testing

2.1.2 General V Model (K2)

Steps of development:

- Definition of requirement, determination of requirement (high-level requirements specifications)
- Functional system draft, systems analyses (functional specifications)
- Technical system draft, architecture draft (software design)
- Component specification
- Implementation

This model has a V-shape and each level has an associated testing level:

Requirements definition and analysis	→	User Acceptance Testing
Functional system design	→	System Testing
Technical design	→	Integration Testing
Component (module) design	→	Unit Testing
Implementation		

For training companies: graphic portrayal of the General V model; in depth description of the General V Model

2.1.3 Rational Unified Process (RUP©) (K2)

Rational Unified Process is a process model by IBM Rational ©

It is an iterative model (i.e. the process is repeated until all scenarios, risks and changes have been addressed). It has 9 disciplines including a requirements discipline (6 engineering disciplines plus 3 supporting disciplines). Each discipline is covered by 4 consecutive project life-cycle phases: inception, elaboration, construction, transition.

For training companies: deepening of the RUP© with graphic presentation; deepened study of the requirements discipline.

2.1.4 Agile Approaches

Requirements Management

In Agile environments, requirements are communicated and tracked through mechanisms such as product backlogs, story cards, and screen mock-ups. Commitments to requirements are either made collectively by the team or an empowered team leader. Work assignments are regularly (e.g., daily, weekly) adjusted based on progress made and as an improved understanding of the requirements and solution emerge. Traceability and consistency across requirements and work products is addressed through the mechanisms already mentioned as well as during start-of-iteration or end-of-iteration activities such as “retrospectives” and “demo days.”

Requirements Development

In Agile environments, customer needs and ideas are iteratively elicited, elaborated, analyzed, and validated. Requirements are documented in forms such as user stories, scenarios, use cases, product backlogs, and the results of iterations (working code in the case of software). Which requirements will be addressed in a given iteration is driven by an assessment of risk and by the priorities associated with what is left on the product backlog. What details of requirements (and other artifacts) should be documented is driven by the need for coordination (among team members, teams, and later iterations) and the risk of losing what was learned. When the customer is on the team, there can still be a need for separate customer and product documentation to allow multiple solutions to be explored. As the solution emerges, responsibilities for derived requirements are allocated to the appropriate teams.

2.1.5 Extreme Programming (K2)

Extreme Programming is a software development methodology by Kent Beck et al responsiveness to changing customer requirements. The project management (e.g. Scrum, paragraph 2.1.6.) defines how and when the customer requirements changes are implemented in the software solution (e.g. in which Sprint). It advocates frequent software releases to the customer in short development cycles (timeboxing), which aims to improve productivity and provide checkpoints where new requirements can be adopted.

Some characteristics of extreme programming include:

- Pair programming
- Extensive code review
- Unit testing of the code
- Avoiding programming of features until they are actually needed

- Flat management structure (no complex hierarchy within the team. It can be best observed in Scrum teams – the Scrum team is “self-managed”, there is no such roles as: leads, PMs etc.)
- Simplicity and clarity in code
- Expecting changes in the customer's requirements as time passes and the problem is better understood
- Frequent communication with the customer and among programmers
- Complete renouncement of requirement determination (no separate requirements “phase” before development starts; requirements development, refinement and discovery are a part of actual software development (programming))

2.1.6 Scrum (K2)

Scrum is an Agile framework. Scrum originally was formalized for software development projects. Scrum contains sets of practices and predefined roles. The main roles in Scrum are:

- Scrum Master (responsible for maintaining the processes)
- Product Owner (represents the stakeholders and the business)
- Team (a cross-functional group performing the actual analysis, design, implementation, testing, etc.)

One of the major characteristics of Scrum is dividing the development into “Sprint” (typically a two to four week period). During each Sprint, the team creates so called potentially shippable product increment.

Scrum allows managing requirements via “backlogs”. There are two types of backlogs:

- Product Backlog – a high-level list maintained during the entire project. It aggregates requirements in a form of broad descriptions of all potential features, prioritized by business value. The Product Backlog is the property of the Product Owner.
- Sprint Backlog – the list of work to be addressed by the team during the next Sprint. Features are broken down into tasks, which, as a best practice, should normally be between four and sixteen hours of work. The Sprint Backlog is the property of the team.

The major characteristics of Scrum approach are:

- At the end of every Sprint, functional software should be delivered – in practice, teams start working on the requirement analysis and continue during the actual Sprint. While decomposing tasks, the requirement is clarified.

- Team members are expected to interact, and regular customer involvement is a key concept in Agile development. Customer feedback can be given at demonstration sessions of the new software version towards the end of a Sprint, or through user acceptance tests.
- Requirements evolve due to the customer feedback – seeing working software helps the customers clarify their requirements.
- Requirements are not fully specified before the start of development, but the features selected for a Sprint are specified at the beginning of the Sprint.
- Features are expected to be re-prioritized as the project advances.

Scrum's principal impact on Requirements Engineering is that requirement specifications are not completed and validated before the beginning of development the project.

The Product Owner and team agree on features from the Product Backlog to be included in the upcoming Sprint based on business priority and required work effort. User requirements are formulated by the Product Owner as “user stories” that contain information on the “who, what, why” and not the “how” of a requirement. At the beginning of the Sprint, the selected features are broken down into the tasks of the Sprint backlog, and then developed.

Involvement of the Product Owners, for example by presenting them implemented functions of the software, helps clarify requirements for the team and the Product Owners themselves.

For training companies give examples for user stories, and the corresponding product and sprint back log items.

For training companies: also explain at least two further agile models including Crystal.

2.1.7 Maturity Model (K2)

Maturity levels serves for the identification and improvement of the process maturity (process assessment and process improvement).

ISO/IEC 15504 (SPICE – Software Process Improvement and Capability Determination) (K1)

ISO/IEC 15504 (SPICE – Software Process Improvement and Capability Determination), is a set of technical standards for the software development process and related business management functions.

ISO/IEC 15504 can be used as a mean to process improvement and/or capability determination (for example, evaluation of the vendor’s process capability).

SPICE defines processes divided into the five process categories:

- Customer-supplier
- Engineering

- Supporting
- Management
- Organization

For each of the processes above, ISO/IEC 15504 defines a capability level:

0. Incomplete process
1. Performed process
2. Managed process
3. Established process
4. Predictable process
5. Optimizing process

Capability Maturity Model Integrated (CMMI)

Defines five maturity levels for development, services and acquisition:

1. Initial (chaotic, ad hoc, individual heroics) – the starting point for use of a new process.
2. Managed – the process is managed in accordance with agreed metrics.
3. Defined – the process is defined/confirmed as a standard business process, and decomposed to levels 0, 1 and 2.
4. Quantitatively managed
5. Optimizing – process management includes deliberate process optimization/improvement.

For training companies: deepening ISO 15504/SPICE and CMMI; with a description of the typical requirements for Requirements Engineering

2.2 Requirements Engineering Process (K2)	30 minutes
--	-------------------

Terms

Customer-oriented process, Perspective, Requirements Engineering

2.2.1 Definition of Requirements Engineering Process (K2)

Requirements Engineering is a discipline, which includes processes needed for the identification, structuring and managing requirements. Requirements Engineering contains the following sub-processes:

- Identification of requirements
- Analysis of requirements
- Specification of requirements
- Agreement on requirements
- Changes of requirements
- Validation and Quality assurance

2.2.2 Influences of Requirements Engineering

Some factors may have a negative influence on Requirements Engineering:

- On the internal side (inside the software vendor's organization):
 - Lack of knowledge of the user's domain
 - Ineffective Requirements Engineering approach/methodology
 - Insufficient personnel experience and skill
- On the external side (outside the software vendor's organization):
 - Lack of communication
 - Unclear and/or changing business objectives resulting in unstable requirements
 - No knowledge about the software development process
 - No involvement of users and/or business stakeholders

There are various stakeholders with different points of view on the Requirements Engineering process. In general, the process can be regarded from the point of view of the customer and from the point of view of the supplier (vendor).

Example:

From the customer point of view the most important aspect in the Requirements Engineering can be: user interface, applications, and services. From the supplier point of view other aspects to be considered are: architecture, load distribution etc.

Methods of the Requirements Engineering process with the customer at the center:

- Customer-oriented analysis and design
- Prototyping approach
- Using demonstrations as a mean to validate the increments of the system

3 Project and Risk Management (K2)	60 minutes
---	-------------------

Learning Objectives for Foundation Level of Requirements Engineering

The objectives identify what you will be able to do following the completion of each module.

3.1 Project Management (K2)

LO-3.1.1 Explain why Requirements Engineering important in projects (K2)

LO-3.1.2 Recall the errors that can occur in Requirements Engineering (K1)

3.2 Risk Management (K3)

LO-3.2.1 Recognize the risks related to Requirements Engineering (K1)

3.1 Project Management (K2)**30 minutes****Terms**

Project conception, Contract negotiations, Project definition, Project execution

3.1.1 Necessity of Requirements Engineering in projects (K2)

Some of main reasons why projects fail are related to requirements. Neglecting Requirements Engineering can cause that requirements are not precise or contradictory, or they do not fulfill the criteria and do not satisfy stakeholders' needs. Therefore careful and structured Requirements Engineering is a necessary part of any project.

Requirements Engineering should contribute to the following areas (K1):

- Project conception
 - Identification of customer's needs and expectations regarding the solution of the problem
 - Establishing high-level requirements
- Contract negotiations
 - Evaluation of customer's requirements
 - Determining the initial scope and required resources for the project
 - Determining costs of development (implementation of requirements)
 - Agreeing on priorities of requirements
- Project definition
 - Definition of roles, tasks, activities and additional processes (example: Change Management)
 - Detailed business design of the solution
 - Contributing to architecture design
 - Contributing to test process deliverables
- Project execution
 - Providing a base for requirements development and requirements verification and validation (testing)

- Forcing reviewing plans and adjusting them to the current scope of the solution in case of changes in requirements

3.1.2 What errors can occur in Requirements Engineering? (K2)

Most common errors are the following:

- Unclear requirements
- Changing requirements (if changes result from the unclear objectives of the project and lack of knowledge of the customer's business domain; changes of requirements are not perceived as an error in Agile and iterative approaches)
- Unstable product and design basis for sub-orders
- Unclear responsibilities (on both the customer's and the vendor's side)
- Gap between customer expectations and project contents
- Insufficient customer involvement
- Project definition with milestones that cannot be achieved
- Imprecise expense estimate
- Imprecise estimate of impact of requirement's changes on the other areas of the product under development
- Lack of traceability

3.2 Risk Management (K2)

30 minutes

Terms

Failure Mode and Effect Analysis, Product Risk, Project Risk, Risk, Risk Management, Risk Management Plan

3.2.1 The necessity of Risk Management (K2)

Efficient Risk Management as a key to lowering project and product risks. Identification, proper analysis and planning adequate reactions on risks minimize the chances of a risk occurring and the consequences if a risk occurs.

3.2.2 Risk (K2)

Risk (K1)

Risk is defined as the effect of uncertainty on objectives, whether positive or negative [ISO 31000].

The other definition describes risk as the chance of an event, hazard, threat or situation occurring and resulting in undesirable consequences or a potential problem. The level of risk is determined by the likelihood of an adverse event happening and the impact (the harm resulting from that event) [ISTQB].

Types of risk (K2)

There are two types of risk:

- Product risk
- Project risk

Project risks (K2)

Project risks are the risks that surround the project's capability to deliver its objectives, such as:

- Organizational factors:
 - Skill, training and staff shortages
 - Personnel issues
 - Political issues, such as:

- Problems with stakeholders communicating their needs and expectations
- Failure to follow up on information found in reviews (example: not improving requirements documenting practices)
 - Improper attitude toward or expectations of Requirements Engineering
- Technical issues:
 - Problems in defining the right requirements
 - The extent to which requirements cannot be met given existing constraints
 - Development or test environment not ready on time
 - Low quality of the design, code, configuration data, test data and tests
- Supplier issues:
 - Failure of a third party (e.g., components not delivered on time)
 - Contractual issues

Product risks (K2)

Potential failure areas (adverse future events or hazards) in the software or system are known as product risks, as they are a risk to the quality of the product. These include:

- Higher risk of failure of delivered software (software or system unable to perform a required function within specified limits)
- Low quality software documentation (incomplete, inconsistent, difficult to maintain)
- The potential that the software/hardware could cause harm to an individual or company
- Poor software characteristics (e.g., functionality, reliability, usability or performance)
- Poor data integrity and quality (e.g., data migration issues, data conversion problems, data transport problems, violation of data standards)
- Software that does not perform its intended functions and does not satisfy the needs of stakeholders
- Business risk based on bad quality

3.2.3 Risk Management (K2)

Risk Management is the process of identification, assessment and prioritization, planning reaction on risks and resolving and monitoring of risks. It allows identifying potential factors that may have a negative effect on the execution of a project and prepare proper action to deal with the risk if it appears.

Different risks may come from different groups of stakeholders: for example, the development team will see different risks than business stakeholders or end-users.

Risk Management consists of the following activities [ISTQB]:

- Risk identification

- Risk analysis
- Risk mitigation

Potential risk treatments

Techniques to manage the risk can be divided into four major categories:

- Avoidance
- Reduction
- Sharing
- Retention

Risk Management Plan

Risk Management Plan should be created before and after (periodically updated) creation of the Project Plan. The Risk Management Plan should provide effective security controls for managing the risks and contain a schedule for control implementation and responsible persons for those actions.

Risk Management Plan includes:

- List of risks
- Probability of occurrence and/or priority
- Severity of impact for each of risks (including cost if applicable)
- Mitigation strategies for each of risks (including the person/group responsible for taking actions)
- Risk assessment matrix

3.2.4 Failure Mode and Effects Analysis (K2)

Common technique for Risk Management (identification, analysis and planning reaction) is Failure Mode and Effects Analysis (FMEA).

FMEA allows prioritizing of potential failures according to the severity of consequences, frequency of the occurrence and how easily they can be detected. FMEA also documents current knowledge and actions about the risks of failures for use in continuous improvement. In most cases FMEA is used during the design stage of the project and its main purpose is to avoid future failures. In later phases it can be used for process control. FMEA should begin in the earliest conceptual stages of the project and continue throughout the whole life cycle. Ideally, FMEA should be scheduled as soon as the preliminary information is available.

The results of an FMEA are actions preventing or reducing the severity or likelihood of failures.

FMEA implementation steps

FMEA is developed in three main steps:

- Step 1: Severity (identification of the severity of potential fault)
- Step 2: Occurrence (identification of how often a potential fault can occur)
- Step 3: Detection (identification of techniques for fault's detection)

4 Responsibilities and Roles (K2)	50 minutes
--	-------------------

Learning Objectives for Foundation Level of Requirements Engineering

The objectives identify what you will be able to do following the completion of each module.

4.1 Basic Roles (K1)

LO-4.1.1 Recall the basic roles that are in Requirements Engineering (K1)

LO-4.1.2 Describe the purpose and role of a stakeholder (K2)

4.2 Tasks of Requirements Engineering (K2)

LO-4.2.1 Describe the tasks of Requirements Engineering (K2)

LO-4.2.2 Recall the characteristics of a Professional for Requirements Engineering (K1)

4.1 Basic Roles (K1)	20 minutes
-----------------------------	-------------------

Terms

Client, Customer, Contractor, Stakeholder

4.1.1 Basic roles (K2)

Roles that affect or are affected by Requirements Engineering

Client (customer)

A person, group or organization requesting the solution.

Contractor (supplier, vendor)

A person, group or organization providing the solution.

The client formulates his needs and provides initial business needs and expectations. Usually it is provided together with the offer/service request. It is the vendor responsibility to explore these needs and extract requirements on that basis.

The contractor delivers solutions based on the client's needs.

Roles within Requirements Engineering

Requirements Manager

A Requirements Manager is a person responsible for documenting, analyzing, tracing, prioritizing and agreeing on requirements and then controlling change and communicating to relevant stakeholders.

Requirements Developer

A Requirements Developer is a technical oriented person mainly involved in the Elicitation, Analysis and prioritizing of requirements.

4.1.2 Stakeholder (K2)

Stakeholder is a group or individual that is affected by or is in some way accountable for the outcome of an undertaking. Project stakeholders are individuals and organizations that are actively involved in the project, or whose interests may be affected as a result of project execution or project completion.

Stakeholders can be either natural persons, legal entities or abstract persons.

Stakeholders often have conflicts of interest among each other. It often results in contradictory requirements. The problem of conflicting requirements needs to be addressed during the Requirements Analysis phase.

There may be many categories of stakeholders including:

- Customers
- End-users
- Managers
- People involved in the organizational processes
- Engineers responsible for system development and maintenance
- Customers of the organization who will use the system
- External bodies (regulators)
- Domain experts

Typical stakeholders are:

- Customer
- End-user
- Project manager
- Product manager
- System analyst
- Business analyst
- Business representatives
- Marketing and sales staff
- Software developer
- Quality assurance staff
- Technical experts (architects, database engineers)
- Change manager

- Project core team
- Management team

Identification of all stakeholders is necessary to take adequate consideration of all points of view on the planned solution.

For training companies: description of typical stakeholder (e.g. Managing Director, Project Manager, client)

4.2 Tasks of Requirements Engineering (K2)

30 minutes

4.2.1 Tasks of Requirements Engineering (K2)

The main task of Requirements Engineering is the following:

- Analysis of business processes performed within an organization
- Identification and analysis of requirements
- Structuring and modeling of requirements
- Quality assurance of requirements and specifications
- Creation of the requirements specification
- Risk analysis (in the context of requirements)
- Change management of requirements
- Agreeing on requirements with stakeholders

4.2.2 Knowledge of a Professional for Requirements Engineering (K1)

Apart from the technical skills, the Professional for Requirements Engineering should possess the following soft skills:

- Skill of moderation
- Self-confident manner
- Ability to convince
- Language skills
- Ability to communicate
- Precision
- Analytical, clear thinking
- Ability to act in a structured way
- Methodological competence
- Stress resistance

5 Identification of Requirements (K2)	150 minutes
--	--------------------

Learning Objectives for Foundation Level of Requirements Engineering

The objectives identify what you will be able to do following the completion of each module.

5.1 Customer (K1)

LO-5.1.1 Recall the contents of a contract (K1)

LO-5.1.2 Identify what should be considered when evaluating requirements (K2)

5.2 Project Visions and Goals (K2)

LO-5.2.1 Explain the characteristics of a typical project vision (K2)

5.3 Identifying Stakeholders (K2)

LO-5.3.1 Explain how can stakeholders be identified (K2)

LO-5.3.2 Identify stakeholders for a specific project (K3)

5.4 Techniques for Identifying Requirements (K2)

LO-5.4.1 Identify the goals of the identification of requirements (K2)

LO-5.4.2 Apply various techniques for identifying requirements (K3)

5.5 Functional and Non-functional Requirements (K2)

LO-5.5.1 Describe the characteristics of functional and non-functional requirements (K2)

LO-5.5.2 Compare the differences between functional and non-functional requirements (K2)

5.6 Description of Requirements (K2)

LO-5.6.1 Describe the contents of a standard requirements document (K2)

LO-5.6.2 Describe the characteristics of a good requirement (K2)

LO-5.6.3 Construct a requirement (K3)

5.1 Customer (K1)**20 minutes****Terms**

Client, Customer, Contract

5.1.1 Customer (K1)

Customer is the organization or person purchasing the software and is one of the key stakeholders of the project. Customer's needs must be satisfied.

The customer must always be involved. The goal is to understand the customer and to develop a mutual understanding of one another. The contractor (vendor) should thereby always put himself in the customer's position.

When evaluating the requirements for project planning purposes (what is one of the topics to be covered by the project), different points of view must be taken into consideration as a specific requirement may have different priority and severity for different stakeholders.

5.1.2 Contract (K2)

The agreement (contract) should formally specify and describe what the customer wants. It must be ensured that the interest of the customer takes center stage (i.e. the vendor is not forcing a solution he prefers but analyzes customer's needs and recommends a solution that allows to satisfy these needs in best possible way).

The agreement:

- Must be compliant with available resources to implement solution
- Is based on: estimations, deadlines, prices and project plans

Requirements Engineering provides input information for such estimates.

The contract should include:

- Short description of the planned solution
- The list of prioritized high level requirements
- The acceptance criteria for each requirement
- The list of products (documentation, code, working software)

- Deadlines for development and delivery of the product
- Other needs and expectations like preferred technology to be used, resource requirements etc.

5.2 Project Visions and Goals (K2)

20 minutes

Terms

Goal, Vision

5.2.1 Vision (K2)

Development of project visions is the first step of Requirements Engineering.

Vision should:

- Define the customers, markets and competitors
- Define the objectives to be achieved
- Allow to achieve common understanding of all stakeholders

It is crucial to have a clear definition of project visions.

For training companies: presentation of typical project visions

Important questions regarding project visions (K2):

- What will the project change?
- Why is the project necessary?
- What happens once the project has been terminated?
- Who will profit from the project?
- Which costs are we willing to bear?
- What risks are we willing to assume?

For each project, the vision must be set anew.

Influences on the project's vision (K1)

The following factors can influence the vision:

- Customers
 - Customer's objectives
 - Customer's preferences
- Strategy
 - Strategy of an organization
 - Market positioning
 - Directions to be followed
- Competition
 - Competition data
 - Market development
- Products
 - Innovation level
 - Target group
- Technologies
 - New tools
 - New standards
- Available resources
 - Time, collaborators, capacities

5.3 Identifying Stakeholders (K2)

20 minutes

Terms

Stakeholder

5.3.1 Identifying Stakeholders (K2)

All stakeholders on the customer and supplier side must be identified.

Each stakeholder or each group of stakeholders may provide new requirements and influence the design of the planned solution. If not all stakeholders are identified there is a risk that some important requirements or limitations remains unknown and are not considered in the design. Missing stakeholders may result in complex changes requested for the software in late stage of the project or after releasing the system into production environment.

Some of stakeholders may create interest groups (for example, all business stakeholders). Interest groups should be brought together as it allows managing their requirements more effectively.

5.3.2 The procedure of identification and evaluation of stakeholders (K2)

The procedure of identification and evaluation of stakeholders involves the following activities:

- Identification of stakeholders (analysis of business processes, determining process' and product' owners, analyzing organizational structure and market)
- Grouping stakeholders into groups (if possible)
- Determining relationships
- Identification of potential conflicts
- Analysis of conflicts, their sources and identification of win-win opportunities
- Identification of risk-minimizing stakeholders to involve them more in project activities
- Identification of stakeholders' perspectives

For training companies: explanation of the identification and evaluation of stakeholders

5.4 Techniques for Identifying Requirements (K2)	40 minutes
---	-------------------

Terms

Apprenticing, Brainstorming, Customer's Representative, Field Observation, Interview, Questionnaire, Reuse, Self-recording, Workshop

5.4.1 Purpose of the identification of requirements (K2)

Main purposes of the Identifying Requirements are the following:

- Identifying all desired functions, characteristics, limitations and expectations
- Orientating the requirements toward the project vision
- Detailing high-levels requirements and describing functions and services clearly
- Excluding functions and features that the customer does not want

5.4.2 Techniques (K1)

Most common techniques for Identifying Requirements are:

- Questionnaires
- Interviews
- Self-recording
- Representatives of the customer on site
- Identification on the basis of existing documents
- Reuse (Reusing the specification of a certain project)
- Brainstorming
- Field observation
- Apprenticing
- Workshops

5.4.2.1 Questionnaires

A questionnaire can consist of open-ended or closed-ended questions. An open-ended question requires from the respondent to formulate his own answer. In case of a closed-ended question the respondent is asked to choose an answer from a number of possible options. These options should be mutually exclusive.

Advantages:

- Minor costs
- A larger audience can be targeted

Disadvantages:

- Not applicable for collecting implicit knowledge
- Low return rate with no responders' motivation
- Questionnaires can often be directive, which prevents identification of real user needs

5.4.2.2 Interview

Interview is a conversational technique where the interviewer is asking the responder to obtain information on specified topic. This technique is very interactive and allows modifying the order of asking previously prepared questions depending on the responder's answers and the situation.

Good interviews are more difficult to conduct than one would think due to normal conversation behaviour (e.g. ending phrases for the conversation partner), which can lead to interpretations being introduced into the data. The interviewer should ask open-ended questions to obtain information and ask close-ended question just to confirm status (e.g. confirm already identified requirements).

Advantages:

- The progression can be adapted according to specific responder

Disadvantages:

- Time-consuming
- Insufficient reproduction of results (difficulty of getting the same answers when repeating an interview)

5.4.2.3 Self-recording

In this technique the stakeholder (for example, an end-user) documents his activities performed to complete specific task.

In addition to document the activities "as-is" the user describes also changes, desires and needs.

Techniques associated with this approach are: demonstrations or document reviews.

Advantages:

- Low time and effort for the Requirements Engineer on the software vendor side

Disadvantages:

- Neglecting “automated” activities (like printing and receiving the printed copy)
- Highly dependent from the motivation and the experience of the user

5.4.2.4 Representatives of the customer on site

This approach is one of the most effective Requirements Identification (and validation) method as it allows the representative systematically monitor the progress, verify correctness of the design and provide feedback and additional information whenever necessary.

Having representatives of the customer on site is one of the main rules in Agile methods.

Advantages:

- Quick feedback
- Provides user-oriented requirements that are easily accepted

Disadvantages:

- High costs for the customer
- Adaptation costs

5.4.2.5 Requirements’ identification on the basis of existing documents

This technique can be used in case there is some already existing documentation that can help in identifying requirements within an organization. Such documentation can be:

- Process models and maps
- Process descriptions
- Organization charts
- Product specification (in the meaning of the outcome of specific process)
- Procedures (i.e. work procedures)
- Standards and instructions

The requirements identified are base for further Requirements Analysis and needs to be detailed and extended with other, related and linked requirements.

Advantages:

- No functionality is neglected

Disadvantages:

- Costly
- Not applicable when there is no or only basic documents within an organization
- Not applicable when the documentation is not maintained correctly (not kept up to date)

5.4.2.6 Reuse (Reusing the specification of a certain project)

Reusing the specification of a certain project can be done when an organization already completed one or more projects similar to the current project. Requirements specification prepared for previous project(s) can be used in another project to shorten the duration of Requirements Analysis and documentation and therefore – allows starting implementation earlier.

In most of the cases only some parts of existing specifications can be used in new project. The documentation to be reused should be always checked against the compliance with the current needs and requirements and properly adjusted.

Advantages:

- Cost savings

Disadvantages:

- High costs of the first project
- Reusing requirements can demand extensive and costly change management if not properly handled in previous projects

5.4.2.7 Brainstorming

Brainstorming is a commonly used technique to obtain requirements related to not well known or new areas of an organization's activity or planned system functionality. It allows collecting many ideas from various stakeholders in short time and with low cost. During the brainstorming session the participant are submitting ideas and concepts regarding given problem.

Advantages:

- Low costs
- A chance of collecting many valuable ideas in short time

Disadvantages:

- Difficult with non-motivated participants
- Difficult to apply in distributed teams

5.4.2.8 Field observation

Field observation allows watching the users' activities and processes being conducted and identifying system requirements on that basis. Conducting on-site observation is watching the users working and documenting the process, tasks and results. In some cases observation is extended with interviewing users about their jobs and the ways they realize their tasks.

Advantages:

- Possibility to observe users at work and identify actual requirements
- Useful when the stakeholders have problems with expressing their needs

Disadvantages:

- Exceptional cases may be omitted
- Not applicable in some situations (for example, from safety and legal reasons)

5.4.2.9 Apprenticing

The purpose of apprenticing is to collect requirements from a customer, especially in case when the processes and activities performed by customer's staff are not easy to describe using other techniques, like interviews, or the customer has problems with articulating the requirements about the planned software.

Apprenticing is a process of learning from the customer his job. The customer, who knows the best how to do specific job, teaches the Requirement Engineer – like a master and a student.

Advantages:

- Helps overcome the difficulty that customer's employees may have in thinking about things abstractly and expressing their tasks in words

Disadvantages:

- High costs and time consuming
- Not applicable in dangerous environments

5.4.2.10 Workshops

Workshop is a kind of meeting focused on specific (previously defined and announced to the participants) topic, usually involving stakeholders representing different areas or/and domains for a short, intensive period.

Workshops may have various goals:

- Identifying requirements (i.e. in order to establish a solution's scope)
- Discovering hidden requirements (i.e. requirements not directly stated or even not realized by the stakeholders but needed to accomplish some of their needs or higher level requirements)

- Developing (detailing) requirements in a newly identified area
- Prioritizing requirements
- Reaching consensus of requirements when it comes to requirements agreeing (sign off)
- Reviewing results of specified process or activity (i.e. reviewing Functional Requirements Specification) and resolving issues that might have appeared

Advantages:

- Involves people who have different points of view on a given problem
- Allows to determine and describe requirements coming from various perspectives
- Allows to quickly discover and resolve potential conflicts between stakeholders' requirements

Disadvantages:

- Difficult in case of geographically distributed teams
- Availability of all people required to attend the workshop
- Consensus is not necessarily easily reach during a workshop, and discussion can stall on (minor) problematic issues, thus making the process lengthy and de-motivating for participants

5.5 Functional and Non-functional Requirements (K2)

20 minutes

Terms

Functional Requirements, Non-functional Requirements

In order to pass the examination, students should be able to provide examples of functional and non-functional requirements and detail the individual quality characteristics.

5.5.1 Functional requirements (K2)

Functional requirements specify what the system does. They specify the functions of the system perceived by the end user. Functional requirements describe also triggers of the process (user action, input/output data causing start of the business process).

Functional requirements should be characterized by the following quality characteristics [ISO/IEC 25000] (K1):

- Suitability
- Accuracy
- Interoperability
- Functionality
- Compliance
- Security

5.5.2 Non-functional requirements (K2)

Non-functional requirements specify how the system does; they describe the quality attributes of the whole system or its specific component or function. They limit the solution i.e. by requiring specific efficiency parameters.

Non-functional requirements are difficult to describe therefore they are often vaguely expressed or not documented at all. This makes them difficult to test. Particular attention should thus be paid to non-functional requirements at all stages of the RE process.

Due to the problems with expressing non-functional requirements, they may be difficult to test. Non-functional requirements should be therefore expressed clearly and be measurable.

Non-functional requirements are [ISO/IEC 25000] (K1):

- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Non-functional requirements specify criteria that can be used to judge the operation of a system therefore they have a great impact on the customer's satisfaction in using the software. Functional requirements have to provide functions; non-functional requirements determine how easy and effective the functions can be used.

5.6 Description of Requirements (K2)

30 minutes

5.6.1 Description of Requirements (K2)

Requirements must be clearly and accurately specified. They should be measurable in order to ensure they are testable and their implementation can be properly checked. It is important to remember that common language has some limitations and disadvantages. This may cause the description of the requirements to be unclear and ambiguous. Therefore proper standards and templates should be used wherever possible. Standards provide common understanding and the best practices of the specification where templates limit the language that can be used.

In addition to standard and templates, vocabularies are an important tool in order to facilitate communication between different stakeholders and to introduce a little control of the natural language's ambiguity.

The description of a requirement must satisfy different criteria (e.g. clear, accurate, unambiguous, measurable, without unnecessary words etc.)

5.6.2 Procedure for Requirement Construction (K3)

Construction of a requirement is performed in the following steps:

1. Determination of the affected process
 - Focus is held on functionality
 - Determining inputs and outputs
2. Classification of the system activity
 - Identifying independent system activity
 - Identifying interactions with the user
 - Identifying interface requirements
3. Determination of the legal commitment
 - Clarification of the legal commitment by key words (should, must etc.)
4. Refining the description of the process
 - Detailed description of objects and integration points
5. Logical and time constraints

- Establishing boundary conditions

Example: a requirement related to generating an invoice with the data taken from external system.

Step of the procedure	Output – a requirement's description
1. Determination of the process: <ul style="list-style-type: none"> • The system process of generating an invoice 	The system generates an invoice
2. Classification of the system activity <ul style="list-style-type: none"> • Generating an invoice after the user's command • Interface with an external system is set 	After the users submits an invoice request the system generates an invoice using data taken from the external system
3. Determination of the legal commitment <ul style="list-style-type: none"> • The system has to generate an invoice with correct data from the external system 	After the users submits an invoice request the system has to generate an invoice using data taken from the external system
4. Refining the description of the process <ul style="list-style-type: none"> • Establishing external financial system's name 	After the users submits an invoice request the system has to generate an invoice using data taken from the SAP Finance system
5. Logical and time constraints <ul style="list-style-type: none"> • Determining time constraint – maximum duration of the operation 30 seconds 	After the users submits an invoice request the system has to generate an invoice using data taken from the SAP Finance system within 30 seconds

For training companies: description of the individual steps in the construction of a requirement

5.6.3 Requirement document (K2)

Standard content of the requirement document is [IEEE 830]:

Table of Contents

1. Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms, and abbreviations
- 1.4 References
- 1.5 Overview

2. Overall description

- 2.1 Product perspective
- 2.2 Product functions
- 2.3 User characteristics
- 2.4 Constraints
- 2.5 Assumptions and dependencies

3. Specific requirements

- 3.1 External interfaces
- 3.2 Functions
- 3.3 Performance requirements
- 3.4 Logical database requirements
- 3.5 Design constraints
 - 3.5.1 Standards compliance
- 3.6 Software system attributes
 - 3.6.1 Reliability
 - 3.6.2 Availability
 - 3.6.3 Security
 - 3.6.4 Maintainability
 - 3.6.5 Portability

Appendixes

Index

6 Specification of Requirements (K2)**100 minutes***Learning Objectives for Foundation Level of Requirements Engineering*

The objectives identify what you will be able to do following the completion of each module.

6.1 Specification (K2)

- LO-6.1.1 Describe the purpose and contents of a requirements specification (K2)
- LO-6.1.2 Describe the characteristics of a requirement specification (K2)
- LO-6.1.3 Describe the purpose and contents of a solution specification (K2)
- LO-6.1.4 Describe the characteristics of a solution specification (K2)
- LO-6.1.5 Recall the standards that are important for requirement specifications and solution specifications (K1)
- LO-6.1.6 Explain what a User Story is (K2)

6.2 Procedure (K3)

- LO-6.2.1 Apply a typical procedure for specification of requirements (K3)

6.3 Formalization (K2)

- LO-6.3.1 Explain the different degrees of formalization that exist for the specification of requirements (K2)
- LO-6.3.2 Apply specific level of formalization in given scenario (K3)

6.4 Quality of Requirements (K2)

- LO-6.4.1 Recall the consequences of errors in the requirements (K1)
- LO-6.4.2 Describe possibilities for avoiding requirement errors (K2)

6.1 Specification (K2)	30 minutes
-------------------------------	-------------------

Terms

IEEE 830, Requirements Specification, Solution Specification

6.1.1 Specification (K1)

A specification is an explicit set of requirements to be satisfied by a material, product, or service.

The specification serves to track and manage requirements. In the specification, requirements are specified in a structured way and are modeled separately (requirements are modeled "independently", as high-level requirements are broken down to a level which each individual requirement constitutes an "independent" entity that can be further developed and tracked). Specification is a formal agreement on requirements to be implemented in the planned software system (or in other form of solution).

The term "specification" may be used in the context of requirements and solution.

6.1.2 Requirements Specifications (K2)

Requirements Specification describes the problem area (an area of interest e.g. a solution of a business problem, a new feature etc.) and contains at least the following information:

- Customer requirements
- Limitations
- Acceptance criteria

The creation of customer's requirements specification should be the customer's task. However in some cases, the vendor can support preparing the requirements specifications.

Creation of other types of specifications of: system requirements, SW requirements, safety requirements, security requirements, environmental requirements, legal requirements etc are tasks for other roles.

6.1.3 User stories (K2)

User stories are used with Agile software development methodologies. User stories are a quick way of handling customer/user requirements. The intention of the user story is to be able to respond faster and with less overhead to rapidly changing real-world requirements.

User story describes functionality that will be valuable. User stories are composed of three aspects:

- A written description of the story used for planning and as reminder
- Conversations about the story that serve to flesh out the details of the story
- Tests that convey and document details and that be used to determine when a story is complete [Mike Cohn: User Stories applied. 2009]

6.1.4 Solution Specifications (K2)

The solution specifications are also called functional specifications, system requirement specifications or software requirements specifications and describe the solution area.

A functional specification is a document that clearly describes the technical requirements for solution. Functional specification is the basis for further system development therefore it must provide precise information about the all functional aspects of the software to be implemented. Based on it, architects and developers are able to efficiently design the technical aspects of the system. Functional specification provides guidance for testers for verification of each functional requirement (e.g. a specification is one of test basis).

Functional specification does not describe how the system functions will be implemented and what technology to be used. It focuses on functionality, on interactions between the user and software.

The purpose of functional specification may be:

- Providing the basis for common understanding of the scope and functionality of solution to be implemented
- Ensuring team consensus on what the system has to achieve before starting writing source code, manuals, preparing data and testing
- Providing development team with detailed description of the necessary functionality in terms of interactions between users and software
- Providing basis for test oracle for test team

6.1.5 Important standards (K1)

The following standards can be used for specifications' creation:

- IEEE 1362 (System Performance Specifications)
- IEEE 830 (Software Requirements Specification)
- IEEE 1233 (System Functional Specifications)

6.2 Procedure (K3)

30 minutes

6.2.1 Procedure of Solution Specification (K3)

Specification serves as an activity for formalizing the results of the Requirements Analysis (K2).

The identification, analysis and specification activities lead to requirements agreement (refer to Agreeing on Requirements (K2)).

Requirements, once identified, analyzed and modeled should be documented in clear, unambiguous manner.

A procedure of Solution Specification includes the following activities:

1. Identification of stakeholders
2. Definition of the vision and objective
3. Requirements determination
4. Structured requirements specification
5. Description of the system environment
6. Determination of the solution (the system and scope definition with the relevant aspects outside of, but influencing, the scope like interfaces to external systems)
7. Requirements analysis
8. Modeling of the problem
9. Modeling of the solution

Procedure formalizes the results of the Requirements Analysis process. The solution model is a basis for design and implementation.

The procedure involves a number of stakeholders who are supporting the specification work in different areas. The output of the procedure, the Solution Specification, serves as a starting point for software, hardware and database design. It describes the function (Functional and Non-Functional specifications) of the system, performance of the system and the operational and user-interface constraints.

6.3 Formalization (K2)

20 minutes

Terms

Formal Level, Non-formal Level, Semi-formal Level

6.3.1 Degrees of formalization (K2)

Requirements specification may be created on different degrees of formalization:

- Non-formal
- Semi-formal
- Formal

6.3.1.1 Non-formal Level

Non-formal approach to writing specification means the document is written in common language, without any formal notation. This approach may be used when the readers have no experience with more formal and technical specification languages and would have difficulties in understanding the content of the document. The main weakness of this approach is that it is ambiguous and may lead to misunderstanding and over interpretation. In addition, non-formal specification is not a good base for implementation and testing as is not clear and precise enough.

6.3.1.2 Semi-formal Level

Semi-formal specification will include some formal notation and will be well structured. Usually such specifications are based on specific template (often derived from relevant standards). Semi-formal specifications may express requirements in a form of models and use formalized common language.

One of most common notations used for semi-formal documentation are UML and SysML.

6.3.1.3 Formal level

Formal specification is a mathematical description of software that may be used to develop an implementation. Formal specification describes what the system should do, usually does not describe how the system should do it. As it is based on mathematical formulas and is more difficult to learn, formal specification is used quite rarely and requires mathematical knowledge.

6.4 Quality of Requirements (K2)

20 minutes

Terms

Checklist, Quality Characteristic, Review, Validation, Verification, Traceability

6.4.1 Background

Requirement defects as a cause of high costs (K2)

As the requirements are the basis for system development, any error or missing requirement will propagate to the other development processes in the project. It is important to notice, that defects resulting from low quality requirements are more expensive to fix in later phases of the project than other types of defects. In addition, the later defects are detected, the higher are the costs to fix them.

Therefore the use of verification (are we producing the product correctly) and validation (are we producing the right product) of the requirements is necessary.

Requirements should be documented and then tested against the quality criteria (refer to chapter 1.1 Requirement (K2)).

6.4.2 Measures for quality improvement and quality assurance of requirements (K2)

The following tools and techniques may be used for quality improvement and quality assurance of requirements:

- Standards and templates
- Reviews and inspections
- Traceability
- Prototyping
- Observance of quality criteria (quality criteria may be: completeness, correctness, compliance of the requirements specification with proper standards)

The quality of a Requirements Specification may be improved by including the following elements:

- Outlining the purpose of the document, scope, definitions and glossary
- Outlining objectives at different levels (i.e. high-level requirements specification has different objectives than detailed functional requirements specification)
- Defining design and implementation constraints
- Grading/prioritization of requirements
- Clear statements of what a system should do rather than how it should do it
- Documenting legislation, assumptions, business rules and dependencies
- Avoiding supplementary descriptions of diagrams that are clear and can stand alone (replace difficult, abstract texts with diagrams if possible)
- Clearly specified user catalogue and permission scheme (users rights and privileges)
- Using structured presentation
- Using simple, clear, precise and unambiguous language

7 Requirements Analysis (K2)**140 minutes***Learning Objectives for Foundation Level of Requirements Engineering*

The objectives identify what you will be able to do following the completion of each module.

7.1 Requirements and Solutions (K1)

- LO-7.1.1 Recall the goal for Requirements Analysis (K1)
- LO-7.1.2 Describe the procedure during Requirements Analysis (K2)
- LO-7.1.3 Explain the concept of structural break between requirements and solutions (K2)

7.2 Methods and Techniques (K2)

- LO-7.2.1 Recall the different models of Requirements Analysis (K1)
- LO-7.2.2 Apply different analysis methods for specific types of models (K3)

7.3 Object-Oriented Analysis (K2)

- LO-7.3.1 Describe the characteristics of UML (K2)
- LO-7.3.2 Describe the characteristics of SysML (K2)

7.4 Cost Estimates (K2)

- LO-7.4.1 Recall the reason for cost estimate (K1)
- LO-7.4.2 Recognize the important factors for cost estimates (K1)

7.5 Prioritization (K2)

- LO-7.5.1 Apply the procedure for prioritizing for a given scenario (K3)

7.6 Agreeing on Requirements (K2)

- LO-7.6.1 Identify what should be considered when agreeing on requirements (K2)

7.1 Requirements and Solutions (K1)

20 minutes

7.1.1 Goal of the Requirements Analysis (K2)

Goal of the Requirements Analysis is creating a solution for the implementation of the requirements. Requirements Analysis takes into account the different stakeholders of the solution, possibly conflicting requirements, analyses relationships and dependencies among requirements etc.

7.1.2 Procedure of the Requirements Analysis (K2)

Procedure of the Requirements Analysis contains the following steps:

1. Analysis of the needs
2. Description of the solution
3. Cost estimate and prioritization

7.1.3 Structural break between requirements and solutions (K2)

Structural break between requirements and solutions means there is a difference between the requirements and solutions. A solution is an implementation of a requirement. Requirements model can be seen as a business model, presenting the business problem to be solved by the solution model. A solution model is more detailed and includes technical specification of the requirements; it is a base for development and testing.

There are three basic levels of models:

- Requirements model
 - Often non-formal specification
 - Business modeling notations (BPMN – Business Process Modeling Notation)
- Solution model
 - Functional structures (algorithms, procedures, workflows)
- Conceptual model
 - Technological software/hardware specifications

There should be traceability between the models.

7.2 Methods and Techniques (K2)

20 minutes

Terms

Context model, Data flow model, Entity Relationship Model, Functional decomposition, Conceptual models, Requirements models, Solution models, State transition model

7.2.1 Analysis methods and models

Different aspects of a system are represented through different views.

Models are developed through suitable methods of analysis.

Analysis method	Analysis model
Context analysis	Context model
Architectural analysis	Functional decomposition
Data stream analysis	Data stream model
Condition analysis	Condition model
Decision analysis	Decision table
Data analysis	Semantic data model Entity-relationship model Data dictionary

7.2.2 Types of models (K2)

There are three basic types of models:

- Requirements model
 - Describes the problem area
 - Designed at the early stages of the project
 - Serves for the Requirements Analysis and cost estimation
 - Provides a basis for the solution model
 - Example: Business Use Case model, Business Class model, Business Process model
- Solution model
 - Describes the solution area from different views on the system
 - Designed simultaneously to the requirement model
 - Determines the shape of implementation of functional and non-functional requirements
 - Provides a basis for the system design
 - Example: Use Case model, Sequence diagram, Activity diagram, State transition diagram
- Conceptual model
 - Technological software/hardware specifications (modules, hardware components, PC characteristics)

7.2.3 Different perspectives of the system (K2)

Some of the points of view related to the system are the following:

- Logical view
 - Functional requirements
- Process view
 - Communication
 - Interaction
 - Non-functional requirements
- Implementation view
 - Components (modules)

- Installation view
 - Integration
 - System architecture

7.2.4 Different models (K1)

The following models can be used:

- Context model
 - Static description of the system
 - Expresses basic architecture
- Functional decomposition
 - Static description of the system
 - Expresses successive decomposition of the system
- Data flow model
 - Dynamic description of the system
 - Graphical representation of the flow of data through a system
 - Does not provide an information about the timing of processes and whether processes will operate in sequence or in parallel
- State transition model
 - Dynamic description of the system
 - Represents the behavior of the system in series of events, that could occur in one or more possible states
- Entity Relationship Model
 - Abstract and conceptual representation of data
 - Contains building blocks: entities, relationships, and attributes

When should specific model be applied? Different models answer other type of questions regarding the solution.

- Context Model ⇔ WHAT (representation of the main flows between the system and outside)
- Functional Decomposition ⇔ WHAT (what functions and features are in the scope of the system)
- Data Flow Model ⇔ WHAT (flows between business process/ functions /activities)

- Entity Relationship Model ⇒ WHAT (what relationships exist between specific entities (objects) of the system)
- State Transition Model ⇒ WHY (cause and effect)

7.3 Object-Oriented Analysis (K2)

30 minutes

Terms

Activity Diagram, Behavioral Diagram, Class Diagram, Communication Diagram, Component Diagram, Composite Structure Diagram, Deployment Diagram, Interaction Overview Diagram, Object Diagram, Object-Oriented Analysis, Package Diagram, Requirements Diagram, Sequence Diagram, State Machine Diagram, Structural Diagram, SysML, Timing Diagrams, UML, Use Case Diagram

7.3.1 UML (K1)

Unified Modeling Language is a unified notation for the analysis and design of systems. It contains different diagram types for different views on the system (structural or behavioral diagrams).

7.3.1.1 Behavioral diagrams (K2)

Behavioral diagrams depict behavioral features of a system or business process.

These diagrams include the following diagram types:

- Activity Diagrams
 - Model the behaviors of a system, and the way in which these behaviors are related in an overall flow of the system
- Use Case Diagrams
 - Capture Use Cases and relationships among Actors and the system
 - Describe the functional requirements of the system, the manner in which external operators interact at the system boundary, and the response of the system
- State Machine Diagrams
 - Show how an element can move between states, classifying its behavior according to transition triggers and constraining guards
- Timing Diagrams
 - Define the behavior of different objects within a time-scale
 - Provide a visual representation of objects changing state and interacting over time
- Sequence Diagrams
 - Structured representation of behavior as a series of sequential steps over time

- Used to depict work flow, message passing and how elements in general cooperate over time to achieve a result
- Communication Diagrams
 - Show the interactions between elements at run-time, visualizing inter-object relationships
- Interaction Overview Diagrams
 - Visualize the cooperation between other interaction diagrams to illustrate a control flow serving an encompassing purpose

7.3.1.2 Structural diagrams (K2)

Structural diagrams depict the structural elements composing a system or function. These diagrams reflect the static relationships of a structure, such as Class or Package diagrams, or run-time architectures such as Object or Composite Structure diagrams.

Structural diagrams include the following diagram types:

- Class Diagrams
 - Capture the logical structure of the system, the classes and objects creating the model, describing what exists and what attributes and behavior it has
- Composite Structure Diagrams
 - Reflect the internal collaboration of classes, interfaces and components (and their properties) to describe a functionality
- Component Diagrams
 - Present the pieces of software creating a system as well as their organization and dependencies
- Deployment Diagrams
 - Describe the execution architecture of the system
- Object Diagrams
 - Present object instances of classes and their relationships at a point in time
- Package Diagrams
 - Depict the organization of model elements into packages and the dependencies amongst them

7.3.2 SysML (K2)

System Modeling Language is a special modeling language for System Engineering. It is an extension of UML 2.1.

SysML offers some improvements over UML. These improvements are the following:

- More flexible and expressive semantic
 - Reduces UML's software-centric restrictions and adds two new diagram types, requirement and parametric diagrams
 - Allows modeling a wide range of systems which include hardware, software, information, processes, personnel and facilities.
- Easy to learn and apply
 - Smaller than UML (does not utilize many of UML's software-centric constructs)
- Supporting models and views
 - Models and views architecturally aligned with IEEE-Std-1471-2000
- Reusing seven of UML's diagrams and providing two new diagrams (requirements and parametric diagrams) for a total of nine diagram types
 - Requirement diagrams to capture functional, performance and interface requirements
 - Parametric diagrams to define performance and quantitative constraints

7.4 Cost Estimates (K2)

20 minutes

Background

Cost estimates connect Requirements Engineering with the Project Management.

7.4.1 Types of estimates (K2)

Most common aspects to be a subject of estimation are:

- Costs
- Time
- Requirements

Cost estimates help to recognize the cost for development, change etc.

7.4.2 Influences on the development costs (K2)

The cost of the project depends on various factors:

- Project type
- Process maturity
- Design and testing methods and tools
- Technology
- Complexity of the planned solution
- Quality objectives (for example desired level of software quality)
- Team qualifications
- Team distribution
- Experiences

The exactness of the cost estimations depends upon the progress of the project and its maturity.

7.4.3 Cost estimation approaches

Cost estimation can be conducted using:

- Analogical conclusion
- Algorithmic procedure

7.4.3.1 Conclusion by analogy (K2)

Cost estimation is based on comparison with similar project's costs. It is based on experience, not on mathematical formulas. With this technique the current project is compared with past projects. The comparison may include:

- The number of requirements
- Scope of the solution
- Technology used
- Personnel's characteristics (skills, experience)

Based on the results of comparison, an estimate can be created.

Estimation procedures are always based on historical data and framework conditions.

Delphi method

It is a structured communication technique used to conduct interactive forecasting. It involves a panel of experts [Linstone75].

The experts are asked to answer questionnaires in some number of rounds. After each round, there is an anonymous summary of the experts' forecasts together with the reasons of the judgments provided. Then experts revise their earlier answers taking into account replies of other members of their panel.

It is believed that during this process the range of the answers will decrease and the group will converge towards the "correct" answer.

The process is stopped at a pre-defined stop criterion. The mean scores of the final rounds determine the results.

Agile estimations

In Agile projects often managed by Scrum, there is an estimation method called Planning Game or Playing Poker. This method is used to gain consensus in the team. The team ability is then measured through something called Burn Down Rate and improved through Retrospective sessions conducted after every Sprint where planned figures are compared to the actual. This allows improving the team's ability to estimate.

7.4.3.2 Algorithmic procedure (K2)

In this approach the costs are calculated on the basis of parameters. Parameters can describe the product (volume, duration), the boundary conditions (efficiency) etc.

The following methods can be used:

- Putnam formula (equation)
- Function Points
- Constructive Cost Model (CoCoMo)

Putnam equation [Putnam91]

$$\text{Effort} = [\text{Size}/\text{Productivity} * \text{Time}^{4/3}]^3 * B$$

Where:

- Size – the product size (estimated by any size estimate used by an organization). Putnam uses ESLOC (Effective Source Lines of Code)
- B – a scaling factor and is a function of the project size
- Productivity – the Process Productivity, the ability of a particular software organization to produce software of a given size at a particular defect rate
- Time – the total schedule of the project in years
- Effort – the total effort applied to the project in person-years

Function Points

A function point is a unit of measurement to express the amount of business functionality provided by an information system to a user. The cost of a single Function Point is calculated on the basis of past projects.

There are five standard "functions" to be counted as Function Points.

- Data Functions:
 1. Internal logical files
 - Tables in a relational database
 - Application control information
 2. External interface files
 - Tables in a relational database, application control information not maintained by the application under consideration

- Transactional Functions:
 1. External Inputs
 - Data entry by users
 - Data or file feeds by external applications
 2. External Outputs
 - Reports created by the application including derived data
 3. External Inquiries
 - Reports created by the application being counted, where the report does not include any derived data

Identified functions are weighted according to three complexity levels; the number of points assigned to each level differs depending on the type of the Function Point.

An example is shown below:

Complexity	Points
Low	7
Average	10
High	15

CoCoMo

CoCoMo allows computing software development effort and cost as a function of program size. The size of software is expressed in EKLOC (estimated thousands of lines of code).

CoCoMo applies to three classes of software projects:

- Organic projects
- Semi-detached projects
- Embedded projects

The basic CoCoMo equations:

- Effort Applied (E) = $a_b(\text{KLOC})^b$ [man-months]
- Development Time (D) = $c_b(\text{Effort Applied})^d$ [months]
- People required (P) = Effort Applied / Development Time [count]

KLOC – the estimated number of delivered lines (expressed in thousands) of code for project
 a_b , b_b , c_b and d_b :

Software project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

7.5 Prioritization (K2)	20 minutes
--------------------------------	-------------------

Terms

Scale, Three-Level Scale

7.5.1 Prioritization (K2)

Prioritization allows establishing requirement's relative importance and the implementation to complete the most crucial requirements first.

Prioritization supports incremental development as it allows grouping requirements and establishing priorities of implementation.

7.5.2 Procedure of prioritization (K2)

The procedure of establishing requirements' priorities involves the following activities:

1. Requirements grouping
 - Identifying requirements influencing each other and dependent upon each other (example: requirements creating one complex functionality)
2. Requirements Analysis
 - Analysis by all involved stakeholders in order to agree the level of importance
 - Impact analysis as a mean to support analysis of the requirements
 - Establishing priorities of requirements
3. Creation of the requirements project plan
 - Establishing a plan where requirements with high priorities are to be developed at first
 - Assigning responsibilities (who implements the requirement)
4. Planning system increments testing
 - Designing test cases for testing each system increment (established on the basis of prioritized requirements) on the basis of requirements' priorities

7.5.3 Prioritization scale (K2)

A common approach to prioritization is to group requirements into priority categories. Usually the three-level scale is used (example: High, Medium and Low).

As such scales are subjective and imprecise, all involved stakeholders must agree on the meaning of each level in the scale they use. The definition of the priority should be clearly stated and should be a key attribute of each requirement.

Examples of three-level scale are:

Name	Description
High	A critical requirement, required for the first software release
Medium	Supports necessary system operations but could wait until a later release if necessary
Low	A functional or quality enhancement, so called “would be nice to have”

Name	Description
Essential	The product is not acceptable unless these requirements are fulfilled
Conditional	Would enhance the product, but the product is not unacceptable if absent
Optional	Functions that may or may not be worthwhile

7.6 Agreeing on Requirements (K2)	20 minutes
--	-------------------

Terms

Signoff

7.6.1 Agreement (K2)

Agreeing on requirements, often called requirements signoff is a formal agreement that the content and scope of the requirements are accurate and complete.

Formal agreement is a base for the project. High-level requirements (business requirements) should be agreed before the project starts. Detailed requirements (functional and non-functional requirements) should be agreed and signed off before moving to implementation phase.

Obtaining requirements signoff is typically the final task of Requirements Analysis and Design.

The requirements signoff should be done by project's stakeholders, including:

- Project Managers on both the customer and the vendor sides
- The customer's business representative
- The Business and System Analysts
- The Requirements Engineers
- The representatives of Quality Assurance, testing and development teams.

The list of requirements must be binding for both, the customer and the vendor side.

One of the purposes of requirements signoff is ensuring the requirements are stable and any changes are to be managed via formal Change Requests. Therefore, formal agreement reduces the risk of introducing new requirements during or subsequent to implementation.

Agreeing on requirements is considered complete, when all relevant projects' stakeholders have signed off the requirements document.

Completion of requirements signoff should be communicated to the project team and usually is a project milestone.

7.6.2 Advantages of requirements' agreement (K1)

- Agreement ensures development of the right product (works based on agreed list of requirements and covers only what is needed to obtain added value for the stakeholders)
- Minimized risk of misunderstanding between the customer and the vendor on the field of the project's scope
- Basis for the further design works

8 Tracking of Requirements (K2)	60 minutes
--	-------------------

Learning Objectives for Foundation Level of Requirements Engineering

The objectives identify what you will be able to do following the completion of each module.

8.1 Tracing within the Project (K2)

- LO-8.1.1 Recall the meaning of traceability (K1)
- LO-8.1.2 Recall typical reasons for changes in requirements (K1)
- LO-8.1.3 Describe the purpose of traceability (K2)
- LO-8.1.4 Recognize the different kinds of traceability (K1)

8.2 Change Management (K2)

- LO-8.2.1 Describe the characteristics of Change Management (K2)
- LO-8.2.2 Recall the constitution of Change Control Board (K1)

8.1 Tracing within the Project (K2)	20 minutes
--	-------------------

Terms

Horizontal and vertical tracing

8.1.1 Evolution of requirements (K1)

Requirements are not stable, but continue to develop during the project lifecycle.

Reasons for continued development and proposed changes may be the following:

- New insights
- New customer needs (resulting i.e. from new regulation, changes in the business, new products etc.)
- Continued work (i.e. next project phase, improving and optimizing already implemented features etc.)
- New connections within the project (i.e. integration with new systems, new access channels like internet or mobile channels etc.)

8.1.2 Traceability (K2)

Traceability provides a solution for managing developing requirements and other artefacts related to those requirements.

Traceability provides a check that all important steps of the development process have been carried out. It should be implemented bidirectional for all artefacts (for example: from requirements to design artefacts and from design artefacts to requirements). Traceability is also important for Testing, Verification and Validation.

Goals of traceability:

- Impact analysis
- Coverage analysis
- Proof of implementation
- Use of the requirement (requirements tracking as a proof that the requirements are being used and how they are used)

In order to ensure good traceability, it is important to label the requirements uniquely.

8.1.3 Types of traceability (K2)

- Horizontal tracing
 - Presents dependences between requirements on the same level (relationships between different types of requirements)
- Vertical tracing
 - Presents dependencies between various artefacts (requirements and solution and performance specification, test cases, code, modules, plans etc.)

8.2 Change Management (K2)

20 minutes

Terms

Change, Change Control Board, Change Management, Change Request

8.2.1 Changes of requirements (K1)

Changes of the requirements may be requested in any time during the realization of the project and after releasing the final software on the production environment. Changes will always happen and it is important to plan changes in the terms of the process and time.

The source of change may be the following:

- Extension of existing functionality
- Defects found in the software/documentation
- Other things found e.g. in testing such as bad performance etc
- Requesting new functionality or a modification of existing functionality
- Changes resulting from external factors (organizational changes, regulatory changes)

8.2.2 Change Management (K2)

The Change Management process is the process of requesting, determining attainability, planning, implementing, and evaluating of changes to a software system, documents or other products of the project. The purpose of Change Management is to allow and support the processing of changes and ensuring traceability of changes.

Change Management process includes the following activities:

1. Identification of potential change
2. Requesting new functionality
3. Analysis of the change request
4. Evaluating the change
5. Planning the change

6. Implementing the change
7. Reviewing and closure of the change
8. Roll-out the change

Depending of its complexity and impact, a change may have various impacts on the system. Small changes may require minor modifications, while complex ones may radically change the logic of the system. Any change should be carefully analyzed in order to establish related risks and evaluate the value of modification against predicted risks.

8.2.3 Change Request (K2)

A change should be submitted as a formal Change Request document (referred also as Request For Change (RFC)). Usually such document describes the reason for change, priority and requested solution together with additional details like:

- The name of person/department or other entity requesting the change
- Submission date
- Planned date of implementation of the change (if applicable)
- Cost for the change

8.2.4 Change Control Board (K1)

Changes are checked and decided on by a Change Control Board (CCB). Introducing CCB supports a controlled way to implement a change, or a proposed change, to a product or service.

Change Control Board is a committee that based on provided information (like the risk related to a change, its impact, effort required to implement) makes decisions regarding whether or not proposed changes should be implemented. The CCB is constituted of project stakeholders or their representatives.

The Change Control Board may consist of the following roles:

- Project management
- Development management
- Requirements Engineers
- Quality assurance (Quality management, Test management)
- Business management, if applicable
- Customer, if applicable

8.2.5 Life cycle of a requirement (K2)

Depending on the degree of requirement's analysis and/or implementation different status will be assigned to the requirement. The life cycle of a requirement can be expressed by the following sample statuses:

- New (proposed)
- For review
- Approved
- Conflicted
- Implemented
- Modified
- Deleted
- Tested
- Deployed

Different approaches and organizations may use different requirement's life cycle (and different statuses). In many cases life cycles of requirements, change requests and defects are very similar and managed using the same tool.

8.2.6 Distinction between Defect Management and Change Management (K2)

It is important to distinguish between a change and a defect. A defect is defined as a flaw in a component or system that can cause the component or system to fail to perform its required function. It is a deviation from the required state of the system. A change is modification of existing features, requirements or functions or new ones.

8.2.7 Impact of a change on the project (K2)

Changes in requirements may have various impacts on the project. Most common are:

- Change of schedule, budget, resources
- Works resulting from the change (depending on the phase of the project):
 - Updating analysis and design artefacts (example: specifications)
 - Updating technical and user documentation
 - Changes in test strategy and tests

- Updating Test Plan
- Updating training needs/plan
- Extending/shortening of programming work scope
- Changing testing preparation and execution scope

9 Quality Assurance (K2)	30 minutes
---------------------------------	-------------------

Learning Objectives for Foundation Level of Requirements Engineering

The objectives identify what you will be able to do following the completion of each module.

9.1 Influencing Factors (K1)

LO-9.1.1 Recall the factors that influence Requirements Engineering (K1)

9.3 Quality Assurance through Testability (K2)

LO-9.3.1 Explain how products of Requirements Engineering support testing (K2)

LO-9.3.2 Describe the use of acceptance criteria (K2)

LO-9.3.3 Describe how Requirements Engineering can contribute to testing (K2)

9.4 Metrics (K2)

LO-9.4.1 Recall the definition of metrics (K1)

LO-9.4.2 Describe what metrics can be used for Requirements Engineering (K2)

9.1 Influencing Factors (K1)	10 minutes
-------------------------------------	-------------------

9.1.1 Influences on the Requirements Engineering (K1)

The quality of Requirements Engineering products depends on the following factors:

- The product that is being produced
- The environment in which the product is produced
- Domain (complexity of the business domain, the level of innovation, the frequency of changes in the business etc.)
- Legal, safety and environmental factors
- Time and cost pressure (time and cost constraints can reduce the ability to run Engineering Requirement Processes)
- Cultural factors (language, education, etc.)
- Technology and design constraints

Such factors must be taken into consideration when planning Quality Assurance activities.

9.2 Verification of requirement in requirements' elicitation stage (K2)	20 minutes
--	-------------------

Verification has to be done continuously during the development of a solution. For verification of requirements in elicitation stages some of the following techniques can be used:

- Reviews techniques
- Simulations
- Presentations
- Demonstrations
- Demos

It is important that verification is planned from the beginning of a project.

9.3 Quality Assurance through Testability (K2)

20 minutes

Terms

Acceptance Criteria, Testability

9.3.1 Requirements Engineering and testing (K2)

Requirements Engineering is closely connected to testing. Good test cases require good requirements that can be tested. The involvement of testers for the specification is therefore very important.

9.3.2 Acceptance Criteria (K2)

According to Prince2™ nomenclature, Acceptance Criteria, called also Success Criteria, are the standards required to satisfy the customer's quality expectations and gain the customer's acceptance of the final product. In other words, they are criteria put on specified function, component or any other item of the software product or other output of the project that must be met to satisfy the customer and gain his acceptance of the product.

The Acceptance Criteria should be agreed upon by both sides – the vendor and the customer – before starting the project (they should be a part of contract documentation) and will form the basis for the Project Quality Plan. Every high level requirement must have at least one acceptance criterion. Such criteria are the basis for the Acceptance Testing.

Each of the criteria must be measurable and the means of measuring the criteria must be realistic and agreed.

An example of Acceptance Criterion is: “Application must respond in less than one second and otherwise display a waiting message”

9.3.3 Methods for testing (K2)

Products of Requirements Engineering support testing by providing so called test basis. Requirements and their specifications can be test basis.

Products of Requirements Engineering can support testing by:

- Enabling functional coverage (covering all functional requirements by test cases; test cases are written to cover all functional requirements)

Functional coverage = $\frac{\sum \text{Requirements tested using test cases}}{\sum \text{Requirements}}$

- Providing a reliable test basis for "black-box" or "specification-based" testing such as:
 - Boundary values for data entry categories
 - Application states
 - Logical and business constraints etc.
- Enabling test techniques, like: Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, State Transition Testing, Use Case Testing

Example: Equivalence partitioning – is a testing technique that divides the input data used in specific module of a software into partitions of data from which test cases can be derived. Test cases are designed to cover each partition at least once.

Example of an equivalence partitioning:

1. The valid range for the age of the user is 1 to 99. The valid range is called a valid partition. There are two further partitions of invalid ranges. The first invalid partition would be ≤ 0 and the second invalid partition would be ≥ 100 .
2. The valid range for answers in a questionnaire contains the following text characters: A, B, C, D. The invalid partition will contain all other text characters, including special characters.

Test cases should consider the valid partition and both invalid partitions.

There are other testing techniques to be used as well, e.g. Boundary Value Analysis, Decision Table Testing, State Transition Testing, Use Case Testing etc. They should be used after a test analysis of the requirements and applied according to requirement contents.

9.3.4 Requirements and test process (K2)

Requirements are the basic input information to the process of development and testing the system. Well-defined requirements reduce the risk of project (or even more, product) failure as they allow careful testing. Stability of requirements allows meeting deadlines of the specific task.

It is important to remember that the requirements should be validated through static tests (what may involve testers) and accepted by Test Managers (as in some cases the requirements may be found as not testable). Testers help improve the quality of requirements by pointing weak points and possible defects. Testers should also participate in reviewing of requirements to ensure their testability.

9.4 Metrics (K2)

20 minutes

Terms

Metric

9.4.1 Metric (K1)

Metric is a measurement scale and the method used for measurement [ISO 14598].

Metrics make it possible to make a quantifiable statement regarding the project status and quality.

It is important to remember that results of measurement (numbers collected during the measurement) must always be compared to reference data (relevant metric).

9.4.2 Metrics for requirements (K1)

The following metrics can be applied to requirements:

- Project costs
 - Number of requirements
- Project tracking
 - Number of requirements traced to other artefacts
- Project stability
 - Number of changeable requirements
- Process improvement
 - Reasons for changes in requirements (defects, improvements)
- Quality of the specification
 - Coverage of the requirements by models
- Number of defects
 - Type of defect
 - Number of faults in requirements with different types (logic, consistency, data defects etc.)

9.4.3 Measurement of the requirements quality (K2)

Some questions allow evaluating the quality of requirements:

- Are the requirements correct?
- Are the requirements understandable?
- Are the requirements feasible?
- Are the requirements traceable?
- Are the requirements identifiable?
- Are the requirements testable?

Some of formulas to be used to measure the requirements quality:

$$\text{Clarity} = \frac{\sum \text{Requirements without defects and issues reported}}{\sum \text{Requirements}}$$

The change rate can also be a measure for the quality of requirements (this does not apply to Agile projects). The higher the change rate of the whole set of requirements (what may be caused by efforts leading to clarify not understandable requirements and/or incomplete or inconsistent description of requirements), the more a project is at risk. Therefore the change rate should be measured to manage risks within a project.

10 Tools (K2)	40 minutes
----------------------	-------------------

Learning Objectives for Foundation Level of Requirements Engineering

The objectives identify what you will be able to do following the completion of each module.

10.1 Advantages of Tools (K2)

- LO-10.1.1 Explain the purpose of tool support for Requirements Engineering (K2)
- LO-10.1.2 Describe what activities can be supported by tools within Requirements Engineering (K2)

10.2 Categories of Tools (K2)

- LO-10.2.1 Explain what requirements are there on tools in the area of Requirements Engineering? (K2)
- LO-10.2.2 Explain what must be taken into consideration in terms of cost regarding tools? (K2)

10.1	Advantages of Tools (K2)	20 minutes
-------------	---------------------------------	-------------------

Terms

Requirements Engineering Tools

10.1.1 Uses of Tools in Requirements Engineering (K2)

Tools for storage and administration of requirements facilitate Requirements Engineering. They can take on repetitive mechanical activities or ensure overview. It is thus possible to keep difficult documents consistent and up-to-date. The selection of a tool must occur before the product is developed. Otherwise such situation may cause substantial problems.

Tools may support the following activities in Requirements Engineering:

- Requirement identification and storage
- Requirements modeling (including prototyping)
- Requirement documenting (requirements specification creating)
- Defining and maintaining requirements traceability

10.1.2 The advantages of using tools (K2)

- Ensuring that all requirements are stored in one place and accessible for all involved stakeholders
- Supporting requirements traceability (of test cases etc.) and allowing to verify the relevant requirements coverage
- Allowing to manage requirements changes in easy way
- Improving the quality of requirements specification by forcing usage of defined document templates and modeling notation
- Time savings by automation on some activities (like generating complete specifications from the tool)

10.2	Categories of Tools (K2)	20 minutes
-------------	---------------------------------	-------------------

Terms

Categories of Tools

10.2.1 Categories of Tools (K2)

- Requirements Elicitation Tools
 - Mind-mapping
- Modeling Tools
 - UML tools
 - SysML tools
- Prototyping Tools
- Requirements Management Tools
- Defect Management Tools
- Change Management Tools
- Project Management Tools

The cost of purchasing tools varies greatly. Commercial tools can be very expensive while open source tools are free. The choice of a tool must therefore be made very carefully. Before selecting a tool, there should be an analysis conducted. The analysis should take into account:

- What modeling notation is used within an organization and should be supported by the tool
- Is the organization planning to use other modeling notations in the future (in this case it may be reasonable to purchase a tool supporting the planned notation together those with already being in use within the organization)
- An organization's requirements regarding the functionality desired from the tool (usually commercial tools provide more complex functions than open source tools)
- The cost of the tool considering if the tool is to be used only for one specific project, or will be used on all/most projects
- Whether the tool can be integrated with other necessary tools (like defect management tool, project management tool, other – depending on the needs of the organization)

- Whether the tool can exchange information with a tool used by a customer organization (in some cases the customer conducts own requirements analysis; products of the vendor's detailed requirements analysis should be then migrated to the customer's environment)
- Ease of use and learning (potential cost of trainings), availability of online help, manuals, tutorials, other support

A hasty choice can result in high costs (K2):

- Cost of purchasing a tool that does not meet the users requirements and does not meet its purpose
- Cost of purchasing an expensive tool that is used only on one project or purchasing an expensive tool while there are open source tools with similar functionality
- Cost of trainings in case of purchasing a tool without sufficient help system (especially in case when only basic functionality is required from the tool)
- Cost of extending the purchased tool with additional, required from the users functions not supported by the tool (while there were other tools with such functionality)
- Cost of integrating the tool with other tools used in the organization

11 Literature

Beck, K.: *Extreme Programming*. Munich 2003

Beck, K.: *Extreme Programming Explained: Embrace Change*. Boston 2000

Beck, K.: *Test Driven Development. By Example*. Amsterdam 2002

Beck, K.: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman 1999

Boehm, B.: *Software Engineering Economics*. Englewoods Cliffs, NJ 1981

Bohner, S.A. and R.S. Arnold, Eds. *Software Change Impact Analysis*. Los Alamitos, California, USA, IEEE Computer Society Press 1996.

Bundschuh, M.; Fabry, A.: *Aufwandschätzung von IT-Projekten*. Bonn 2004

Cockburn, A.: *Agile Software Development*. Addison Wesley 2002

Cockburn, A.: *Writing Effective Use Cases*. Amsterdam 2000

Cohn M.: *Estimating With Use Case Points*, Fall 2005 issue of Methods & Tools

Cotterell, M. and Hughes, B.: *Software Project Management*, International Thomson Publishing 1995

Newman, W.M. and Lamming, M.G.: *Interactive System Design*, Harlow: Addison-Wesley 1995

Davis A. M.: *Operational Prototyping: A new Development Approach*. IEEE Software, September 1992. Page 71

Davis, A. M.: *Just Enough Requirements Management. Where Software Development Meets Marketing*, Dorset House, 2005, ISBN 0932633641

DeMarco, T. et al.: *Adrenalin-Junkies und Formular-Zombies – Typisches Verhalten in Projekten*. Munich 2007

DeMarco, T.: *Controlling Software Projects: Management, Measurement and Estimates*. Prentice Hall 1986

DeMarco, Tom: *The Deadline: A Novel About Project Management*. New York 1997

Dorfman, M. S.: *Introduction to Risk Management and Insurance (9 ed.)*. Englewood Cliffs, N.J: Prentice Hall 2007. ISBN 0-13-224227-3.

Dynamic Systems Development Method Consortium. See: <http://na.dsdm.org>

Ebert, Ch.: *Systematisches Requirements Management. Anforderungen ermitteln, spezifizieren, analysieren und verfolgen*. Heidelberg 2005

Evans, E. J.: *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Amsterdam 2003

Graham, D. et al: *Foundations of Software Testing*. London 2007

Gilb, T.; Graham, D.: *Software Inspection*. Reading, MA 1993

Gilb, T.: *What's Wrong with Requirements Specification*. See: www.gilb.com

Heumann, J.: *The Five Levels of Requirements Management Maturity*, see: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/feb03/ManagementMaturity_TheRationalEdge_Feb2003.pdf

Linstone H. A., Turoff M.: *The Delphi Method: Techniques and Applications*, Reading, Mass.: Addison-Wesley, ISBN 9780201042948, 1975

Hull, E. et. All: *Requirements Engineering*. Oxford 2005

IEEE Standard 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology

IEEE Standard 829-1998 IEEE Standard for Software Test Documentation

IEEE Standard 830-1998 IEEE Recommended Practice for Software Requirements Specifications

IEEE Standard 1012-2004: IEEE Standard for Software Verification and Validation

IEEE Standard 1059-1993: IEEE guide for software verification and validation plans

IEEE Standard 1220-1998: IEEE Standard for Application and Management of Systems Engineering Process

IEEE Standard 1233-1998 IEEE Guide for Developing System Requirements Specifications

IEEE Standard 1362-1998 IEEE Guide for Information Technology-System Definition – Concept of Operations (ConOps) Document

ISO 9000

ISO/EIC 25000

ISO 12207

ISO 15288

ISO 15504

ISO 31000: Risk Management - Principles and Guidelines on Implementation

IEC 31010: Risk Management - Risk Assessment Techniques

ISO/IEC 73: Risk Management – Vocabulary

ISTQB: ISTQB Glossary of Testing Terms 2 1

ISTQB: Certified Tester, Foundation Level Syllabus, version 2011

Jacobsen, I. et al.: *The Unified Software Development Process*. Reading 1999

Jacobson, I. et al.: *Object-Oriented Software Engineering. A Use Case Driven Approach*. Addison-Wesley 1993

Kilpinen, M.S.: *The Emergence of Change at the Systems Engineering and Software Design Interface: An Investigation of Impact Analysis. PhD Thesis*. University of Cambridge. Cambridge, UK 2008.

Lauesen, S.: *Software Requirements: Styles and Techniques*. London 2002

Mangold, P.: *IT-Projektmanagement kompakt*. Munich 2004

- McConnell, S.: *Aufwandschätzung für Softwareprojekte*. Unterschleißheim 2006
- McConnell, S.: *Rapid Development: Taming Wild Software Schedules (1st ed.)*. Redmond, WA: Microsoft Press. ISBN 1-55615-900-5, 1996
- Paulk, M., et al: *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA 1995
- Pfleeger, S. L.: *Software Engineering: Theory and Practice, 2nd edition*. Englewood Cliffs, NJ 2001
- Pfleeger, S.L. and J.M. Atlee: *Software Engineering: Theory and Practice*. Upper Saddle River, New Jersey, USA, Prentice Hall 2006.
- Pohl, K.: *Requirements Engineering. Grundlagen, Prinzipien, Techniken*. Heidelberg 2007
- Project Management Institute: *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)*. PMI 2004
- Putnam, L.e H.; Ware M. *Measures for excellence: reliable software On time, within budget*. Yourdon Press. ISBN 0-135676-94-0, 1991
- Robertson, S.; Robertson, J.: *Mastering the Requirements Process*, Harlow 1999
- Rupp, C.: *Requirements-Engineering und Management. Professionelle, Iterative Anforderungsanalyse in der Praxis*. Munich 2007
- Sharp H., Finkelstein A. and Galal G.: *Stakeholder Identification in the Requirements Engineering Process*, 1999
- Sommerville, I.: *Requirements Engineering*. West Sussex 2004
- Sommerville, I.: *Software Engineering 8*. Harlow 2007
- Sommerville, I.; Sawyer, P.: *Requirements Engineering: A Good Practice Guide*. Chichester 1997
- Sommerville, I.; Kotonya, G.: *Requirements Engineering: Processes and Techniques*. Chichester 1998
- Spillner, A. et all: *Software Testing Foundations*. Santa Barbara, CA 2007
- SWEBOK - The Guide to the Software Engineering Body of Knowledge:
<http://www.computer.org/portal/web/swebok/home>
- Thayer, R. H.; Dorfman, M.: *Software Requirements Engineering, 2nd edition*. Los Alamitos, CA 1997
- V-Modell® XT: <http://www.vmodellxt.de/>
- Wieggers, K.E.: *First Things First: Prioritizing Requirements*. Software Development, September 1999
- Wieggers, K. E.: *Software Requirements*. Redmond 2005
- Wieggers, K. E.: *More About Software Requirements: Thorny Issues and Practical Advice*. Redmond, Washington 2006
- Young, R. R.: *Effective Requirements Practices*. Addison-Wesley 2001

12 Index

Activity Diagram, 74
 Agreeing on Requirements, 64, 68, 84
 Behavioral Diagram, 74
 Business Analysis, 3
 Change Management, 86, 89
 Change Request, 90
 Class Diagram, 74
 Component Diagram, 74
 Cost Estimates, 68, 77
 Customer, 43, 44
 Description of Requirements, 43, 57
 Documentation of Requirements, 60
 Failure Mode and Effects Analysis, 36
 FMEA, 36, 37
 Formalization, 60, 65
 Function Points, 79
 Functional and Non-functional Requirements, 55
 Identifying Requirements, 49
 IEC 31010, 105
 IEEE 1233, 63
 IEEE 1362, 63
 IEEE 830, 63
 Impact Analysis, 104, 105
 ISO 12207, 105
 ISO 15288, 105
 ISO 15504, 105
 ISO 31000, 34, 105
 ISO 9000, 105
 ISO/EIC 25000, 105
 ISO/IEC 73, 105
 Object Diagram, 74
 Object-oriented Analysis, 62, 68, 74
 Prioritization, 68
 Process Models, 23
 Product risk, 34
 Project Management, 31, 32, 104, 106
 Project risk, 34
 Quality Assurance, 93, 96
 requirement, 12, 96
 Requirement document, 59
 Requirement Manager, 39
 Requirements Analysis, 68
 Requirements Diagram, 74
 Requirements Engineering, 1, 3, 9, 11, 22, 31, 32, 35, 38, 43, 60, 68, 86, 93, 100, 101, 105, 106
 Risk, 31, 34, 35, 36, 104, 105
 Risk Assessment, 36, 105
 Risk Management, 31, 34, 35, 36, 104, 105
 Signoff, 84
 Software Requirements Specifications, 61, 105
 Specification, 60, 61
 Structural Diagram, 74
 SysML, 74
 Testability, 93, 96
 Three-Level Scale, 82
 Tools, 100, 101, 102
 UML, 74
 Use Case Diagram, 74

Verification and Validation, 105