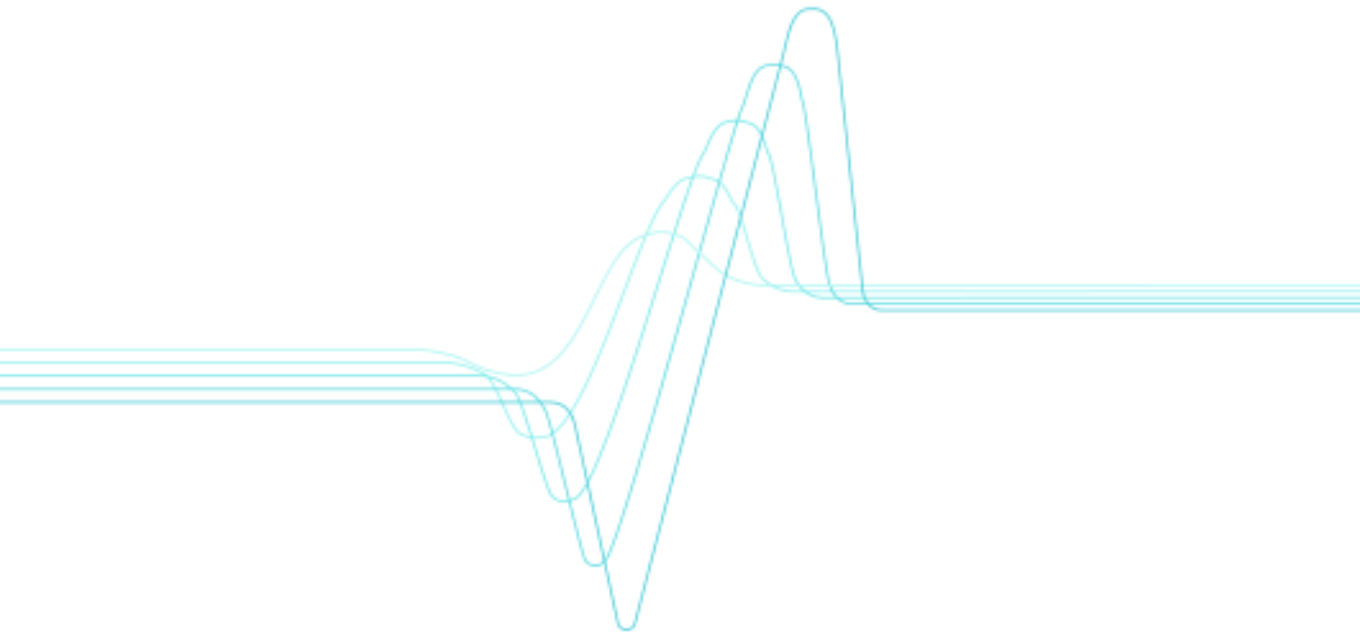
ESPOO 2003 VTT PUBLICATIONS 508

Päivi Parviainen, Hanna Hulkko, Jukka Kääriäinen, Juha Takalo & Maarit Tihinen

Requirements engineering

Inventory of technologies



VTT PUBLICATIONS 508

Requirements engineering

Inventory of technologies

Päivi Parviainen, Hanna Hulkko, Jukka Kääriäinen,
Juha Takalo & Maarit Tihinen

VTT Electronics

ISBN 951–38–6245–3 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6246–1 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Marja Kettunen

Otamedia Oy, Espoo 2003

Parviainen, Päivi, Hulkko, Hanna, Kääriäinen, Jukka, Takalo, Juha & Tihinen, Maarit. Requirements engineering. Inventory of technologies. Espoo 2003. VTT Publications 508. 106 p.

Keywords Requirements engineering (RE), RE methods, RE techniques, RE tools, system and software engineering

Abstract

The purpose of this publication is to describe existing systems and software requirements engineering techniques, methods and tools based on a literature study. This publication covers a wide range of requirements engineering methods and theoretical issues and thus provides a broad view of the field. Also, some RE tools are described.

Requirements engineering is also described in general and RE processes introduced to provide background information about RE and help to understand the method descriptions. The main processes of RE as seen in this publication include: System requirements development, requirements allocation and flow-down, software requirements analysis and specification and continuous processes including requirements documentation, requirements validation and verification and requirements change management. Requirements Management (RM) activities are understood to begin before actual requirements engineering process phases (RM planning) and continuing during design, implementation, testing and maintenance phases.

Preface

This publication has been drawn up during MOOSE (Software engineering methodologies for embedded systems), which is an ITEA project (no 01002). The project's main goal is to seamlessly integrate the different areas of product and software development into a common framework.

The purpose of the literature study is to provide a comprehensive review of the current state of requirements engineering. As this publication is an inventory of existing requirements engineering methods, techniques and tools and contains the requirements engineering process definition, it serves as the basis for further research performed within the MOOSE project. More information about MOOSE project together with its goals and publications can be found from the project's web site, http://www.mooseproject.org.

Contents

List of terminology

The terminology of this publication is presented in the following.

Allocation	The process of distributing requirements, resources, or other entities among the components of a system or program. (IEEE Std 610.12–1990)
Architecture	The organisational structure of a system or component. (IEEE Std 610.12–1990)
Baseline	(1) A specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, that can be changed only through formal change control procedures. (2) A document or a set of documents formally designated and fixed at specific time during the life cycle of a configuration item. (3) Any agreement or result designated and fixed at a given time, from which changes require justification and approval. (IEEE Std 610.12–1990)
Business Requirements	Business requirements represent high-level objectives of the organisation or customer requesting the system or product. (Wiegers, 1999)
COTS	COTS (commercial off-the-shelf) is a ready-made software product, which is supplied by a vendor and it has specific functionality as part of a system (Morisio et al., 2000).
Constraint	A statement that expresses measurable bounds for an element or function of the system. That is, a constraint is a factor that is imposed on the solution by force or compulsion and may limit or modify the design changes. (IEEE Std 1233–1998)
Derived requirement	A requirement deduced or inferred from the collection and organisation of requirements into a particular system configuration and solution. (IEEE Std 1233–1998)

Embedded software	A software that is part of a larger system and performs some of the requirements of that system; for example, software used in an aircraft or rapid transit system. (IEEE Std 610.12–1990)
Embedded System	Embedded systems are products, which are directly incorporated into electromechanical devices other than general-purpose computer hardware, devises known as target systems. (Stankovic, 1996)
End user	The person or persons who will ultimately be using the system for its intended purpose. (IEEE Std 1233–1998)
Hardware	Physical equipment used to process, store or transmit computer programs or data. (IEEE Std 610.12–1990)
Method	Provides a prescription for how to perform a collection of activities, focusing on how a related set of techniques can be integrated, and providing guidance on their use. (Nuseibeh & Easterbrook, 2000.)
Prototyping	A hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process. (IEEE Std 610.12–1990)
Raw requirement	(An environmental or customer) requirement that has not been analysed and formulated as a well-formed requirement. (IEEE Std 1233–1998)
Requirement	- (1) A condition or capability needed by a user to solve a problem or achieve an objective. - (2) 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. - (3) A documented representation of a condition or capability as in (1) or (2). (IEEE Std 610.12–1990)

Requirements Engineering (RE)	Activities that cover discovering, analysing, documenting and maintaining a set of requirements for a system. (Sommerville & Sawyer, 1997)
Requirements Elicitation	Elicitation is one of the first phases in requirements engineering and purpose is to discover requirements for the system being developed. Requirements are elicited from customers, end-users and other stakeholders such as system developers. (Kotonya & Sommerville, 1998)
Requirements analysis	Analysis is one of the first phases in requirements engineering and purpose is to analyse the elicited requirements. When requirements are discovered, conflicts, overlaps, omissions and inconsistencies should be analysed. (Kotonya & Sommerville, 1998; Sommerville & Sawyer, 1997)
Requirements Management (RM)	Requirements management manages changes to agreed requirements, relationships between requirements, and dependences between the requirements document and other documents produced during the systems and software engineering process. (Kotonya & Sommerville, 1998)
Requirements Negotiation	Purpose of negotiation is to discover missing requirements, ambiguous requirements, overlapping requirements and unrealistic requirements. The result of the negotiation process is a definition of the system requirements, which are agreed by requirements engineers and stakeholders. (Sommerville & Sawyer, 1997)
Requirements Traceability (RT)	Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases) (Gotel & Finkelstein, 1994a)

Requirements Validation	Requirements validation is concerned with the process of examining the requirements document to ensure that it defines the right system (i.e. the system that the user expects). (Kotonya & Sommerville, 1998)
Requirements Verification	The process of ensuring, that requirements statements are accurate, complete and that they demonstrate the desired quality characteristics. (Wiegers, 1999)
Specification	A document that fully describes a design element or its interfaces in terms of requirements (functional, performance, constraints, and design characteristics) and the qualification conditions and procedures for each requirement. (IEEE Std 1220–1998)
Stakeholders	Stakeholders mean people or organisations who will be involved by the system and who have an influence on the system requirements. They could be end-users, managers and others involved influenced by the system. They also are for instance engineers responsible for the system development and maintenance, customers of the organisation. (Kotonya & Sommerville, 1998.)
System	A set or arrangement of elements [people, products (hardware and software) and processes (facilities, equipment, material, and procedures)] that are related and whose behaviour satisfies customer /operational needs, and provides for the life cycle sustainment of the products. (IEEE Std 1220–1998)
System breakdown structure (SBS)	A hierarchy of elements, related life cycle processes, and personnel used to assign development terms, conduct technical reviews, and to partition out the assigned work and associated resource allocations to each of the tasks necessary to accomplish the objectives of the project. It also provides the basis for cost tracking and control. (IEEE Std 1220–1998)
System Engineering	System engineering integrates all the disciplines and speciality groups into a team effort forming a structured

	development process that proceeds from concept to production to operation. System engineering considers both the business and the technical needs of all customers with the goal of providing a quality product that meets the needs of the user. (Leffingwell & Widrig, 2000)
Techniques	Technical and managerial procedures that aid in the evaluation and improvement of the software development process. (IEEE Std 610.12–1990)
Technologies	Used here as an upper level term to cover processes, methods, techniques and tools.
Traceability	The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; e.g., the degree to which the requirements and design of a given system element match. (IEEE Std 610.12–1990)
Trade-off analysis	An analytical evaluation of design options/alternatives against performance, design-to-cost objectives, and life cycle quality factors. (IEEE Std 1220–1998)

1. Introduction

This publication has been drawn up during MOOSE (Software engineering methodologies for embedded systems), which is an ITEA project (no 01002). The basis of this publication is a literature study on Systems and Software Requirements Engineering performed at VTT Electronics.

The publication presents both theoretical issues concerning systems and software requirements engineering (i.e. the requirements engineering processes), and practical means for implementing it (i.e. existing methods, techniques and tools). The publication consists of three main parts:

1. Requirements engineering processes (chapter 2) describes requirements engineering in general and introduces RE processes. The purpose of this chapter is to provide the reader with an overview of requirements engineering. The background information presented in this chapter serves as the basis for understanding the method descriptions in chapter 3.

2. Requirements engineering methods (chapter 3) describes the RE methods discovered during the performed literature study. Methods are organised based on the requirements engineering processes presented in chapter 2.

3. Requirements engineering tools (chapter 4) introduces general types of requirements engineering and requirements management tools and technologies. The chapter also provides an insight into the variety of computer-aided RE tools.

This publication covers a wide range of requirements engineering methods and theoretical issues and thus provides a broad view of the field. An important purpose of this document is to function as a background for further, more detailed and focused research. This research will be carried out later on in the MOOSE project, where a shortcomings analysis will be conducted for some of the requirements engineering techniques, methods, and tools presented here.

2. Requirements engineering processes

This chapter introduces the requirements engineering processes, including their purposes, inputs, activities and outputs. Also, methods that can be used during the processes are mentioned; more detailed descriptions of the methods can be found from the method part of the publication (chapter 3).

According to Sommerville & Sawyer (1997), selection of RE process depends on many things: organisation, systems engineering and software development process, type of the software developed, etc. All processes do not fit to all organisations. Usually good requirements engineering process includes the following activities (Sommerville & Sawyer, 1997):

1. Requirements elicitation, where the system requirements are discovered through consulting the stakeholders, from system documentation, domain knowledge and market studies.
2. Requirements analysis and negotiation, where the requirements are analysed in detail, and they are accepted by the stakeholders.
3. Requirements validation, where the consistency and completeness of the requirements is checked.

These activities also need support to accommodate changes in the requirements.

In embedded computer systems, due to physical constraints (e.g., timing, heat production), system requirements engineering is very important, even more important than software requirements engineering. Figure 1 depicts the main processes of system and software requirements engineering and how requirement management (RM) is understood as part of RE. The figure is based on (Kotonya & Sommerville, 1998; Sailor, 1990; Thayer & Royce, 1990). In addition, RM activities are understood to begin before actual requirements engineering process phases (RM planning) and continuing during design, implementation, testing and maintenance phases.

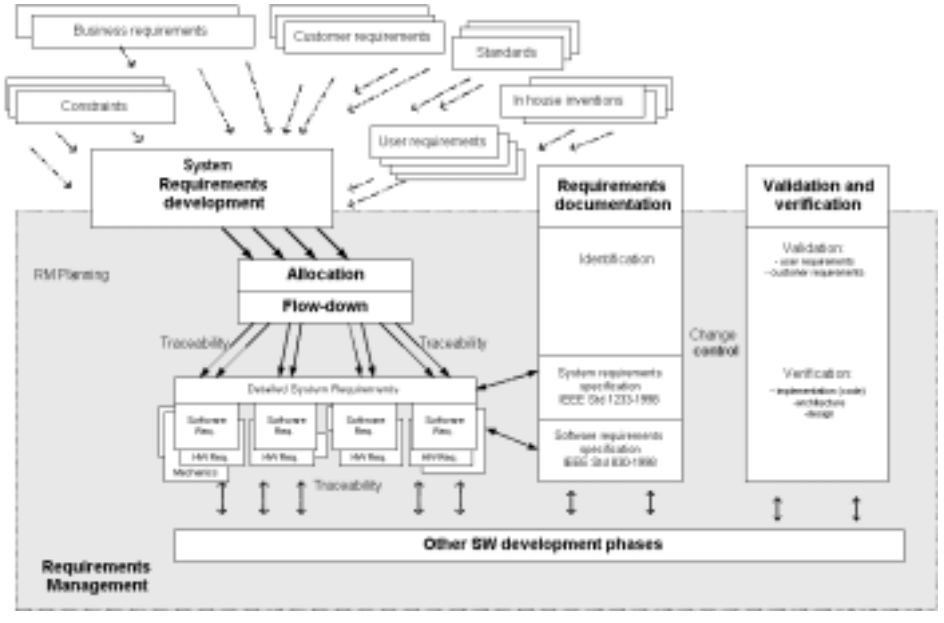


Figure 1. The context of the System and Software requirements engineering processes

The figure shows the different viewpoints that should be considered while gathering requirements during system requirements phase. RE is iterative process that will go into more details during each iterative cycle during development phases. During gathering requirements, some analysis (e.g., cost - benefit and technical feasibility analysis) is also done. In the next phase the captured requirements will be allocated to system components and detailed system requirements are defined and validated. Note that allocation and flow-down may be done for several hierarchy levels. In allocation and flow-down phase requirements identification and traceability have to be taken into account and both system and software requirements specifications will be documented. That phase provides essential input for other SW development phases and maintenance. Requirements management (RM) activities are shown as a continuous and cross-section process that begins from RM planning and continues via activities of identification, traceability and change control during and after RE process phases.

In following Table 1 the RE process phases are presented with CMMI (CMMI-SE/SW, v1.1) engineering process areas.

Table 1. Requirements engineering process phases and CMMI process areas.

RE process phases	The engineering process areas of CMMI and their Specific Goals (SG)	Specific Practices (SP) by goal
System requirements development	***Requirement development:***	
	SG1. Develop customer requirements	SP1.1-1 Collect stakeholder needs SP1.1-2 Elicit needs SP1.2-1 Develop the customer requirements
	SG2. Develop product requirements	SP2.1-1 Establish product and product-component requirements
	SG3. Analyse and validate requirements	SP3.1-1 Establish operational concepts and scenarios SP3.4-3 Analyse requirements to achieve balance
	Requirements management:	
	SG1. Manage requirements	SP1.2-2 Obtain commitment to requirements
Requirements allocation and flowdown	***Requirement development:***	
	SG2. Develop product requirements	SP2.1-1 Establish product and product-component requirements SP2.2-1 Allocate product-component requirements SP2.3-1 Identify interface requirements
	Requirements management:	
	SG1. Manage requirements	SP1.2-2 Obtain commitment to requirements
Software requirements analysis and specification	***Requirement development:***	
	SG3. Analyse and validate requirements	SP3.1-1 Establish operational concepts and scenarios SP3.2-1 Establish a definition of required functionality SP3.3-1 Analyse requirements SP3.4-3 Analyse requirements to achieve balance
	Requirements management:	
	SG1. Manage requirements	SP1.2-2 Obtain commitment to requirements
Continuous activities (documentation, validation and verification, change control)	***Verification:***	
	SG1. Prepare verification	SP1.3-3 Establish verification procedures and criteria

	SG2. Perform peer reviews SG3. Verify selected work products ***Validation*** SG1. Prepare for validation SG2. Validate product or product components ***Requirements management:*** SG1. Manage requirements	SP2.2-1 Conduct peer reviews SP2.3-2 Analyse peer review data SP3.2-2 Analyse verification results and identify corrective action SP1.1-1 Select products for validation SP1.2-2 Establish the validation environment SP1.3-3 Establish validation procedures and criteria SP2.1-1 Perform validation SP2.2-1 Analyse validation results SP1. 5-1 Identify inconsistencies between project work and requirements
Requirements management	***Requirements management:*** SG1. Manage requirements	SP1.1-1 Obtain an understanding of requirements SP1.3-1 Manage requirements changes SP1.4-2 Maintain bi-directional traceability of requirements

In the following sections (2.1–2.3), each RE process is introduced, including general descriptions of activities and references to suitable methods. In section 2.4 continuously ongoing RE activities – documentation, validation and verification – are shortly presented. Section 2.5 describes requirement management (RM) process and its main activities, i.e., traceability and change control. We have distinguished detailed RM consideration from RE phase model to emphasise its support role for RE.

2.1 System requirements development

Purpose:

The main purpose of the system requirement development process is to examine and gather desired objectives of the systems from different viewpoints (e.g. customer, users, system's operating environment, trade and marketing) and these objectives are identified as a set of functional requirements of the system. The requirement development phase includes activities from requirements elicitation process and also some parts of requirements analysis process as described by Kotonya and Sommerville (1998).

Input:

Business requirements, customer requirements, user requirements, constraints, in-house ideas and standards are inputs for this process.

Activities:

The process phase begins by gathering (eliciting) system requirements by communication with all stakeholders. Typically, defining system requirements is started by observing and interviewing people (Ambler, 1998). This is not a straightforward task, because users may not realise, which their work activities can be implemented by a computer system, and on the other hand, sometimes users' requirements are misunderstood because of their own terminology used to describe the desired functionality. In addition, different types of users will have different requirements. System requirements evolve from user requirements, however, user and system requirements must be kept separate (Stevens et. al., 1998). On the other hand, user requirements have to be analysed within the context of business requirements (management perspective) like cost-effectiveness, organisational and political requirements. Also, environment of the system, e.g., external systems and technical constraints, have to be examined and explicated. The main result of the system requirement development phase are top-level system requirements (Sailor, 1990), which are the initial system requirements on the top of the requirements hierarchy.

System requirements development process is also the phase where requirements documentation has to be considered for the first time, e.g., gathered requirements have to be identified. The following figure shows the context for developing system requirements specification (SyRS).

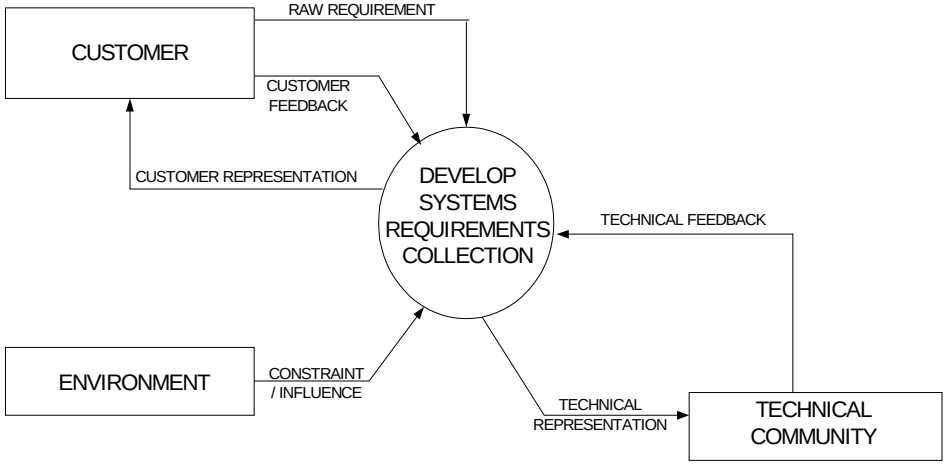


Figure 2. The context for developing of SyRS (IEEE Std 1233-1998).

IEEE Standard (1233–1998) states that customers are the prime system drivers providing their objectives and needs, or describing problems or ideas for the desired system. Feedback to the customer includes SyRS representations, e.g., technical interchange or communication clarifying or confirming requirements. Also, the environment can set constraints for system requirements. Environmental influences can be classified as follows: political, market, standards and technical policies, cultural, organisational and physical factors. The technical community includes activities of system design, implementation, integration, test, manufacturing, deployment, operations and maintenance.

According to IEEE Std 1233–1998 the SyRS development includes four (4) sub-processes:

1. Identify requirements from customer, environment and technical community (see Figure 2)
2. Build well-formed requirements
3. Organise the requirements into SyRS
4. Present the SyRS in various representations for different audiences.

The context of the first sub-process was already described. In the second phase, well-formed requirements should be built. A well-formed requirement is a statement of system functionality (a capability) that must be met or possessed by a system to solve a customer problem or to achieve a customer objective, and that is qualified by measurable conditions and bounded by constraints. In the third subprocess, structure should be added to the set of requirements by relating the requirements to one another according to some comparative definition method. In the last phase, analysts (who work with customer) should identify the best means of communicating the requirements to all individuals that need to understood, review, accept or use the SyRS.

The following table (Table 2) presents requirements categorisation based on Buede, D.M. (1997).

Table 2. Categorisation model for system requirements (Buede, 1997).

Input/output requirements	This requirements category includes sets of acceptable inputs and outputs, trajectories of inputs to and outputs from system, interface constraints imposed by the external systems and eligibility functions that match system inputs with system outputs for the life-cycle phase of interest. This category can be partitioned into four subsets: - Inputs. - Outputs. - External interface constraints. - Functional requirements.
Technology and system-wide requirements	This requirements category consists of constraints and performance index thresholds (e.g. the length of the operational life for the system, the system's availability) that are placed upon the physical resources of the system. This category can be partitioned into four subsets: - Technology. - The different attributes of the system, e.g. maintainability and availability. - Cost. - Schedule, e.g. development time period, operational life of the system.

Trade-off requirements	Algorithms for comparing any two alternative designs on the aggregation of cost and performance objectives. These algorithms can be divided into three sub categories: - Performance trade-offs. - Cost trade-offs. - Cost-performance trade-offs.
System test requirements	These requirements have four primary elements: - Observance: stating how the test data for each requirement will be obtained. - Verification plan: stating how the test data will be used to determine that the real system conforms to the design that has been developed. - Validation plan: stating how the test data will be used to determine that the real system complies with the originating requirements. - Acceptability: stating how the test data will be used to determine that the real system is acceptable to the stakeholders.

During system requirements development phase, different categorisation models can be used to ensure that requirements are considered from all relevant viewpoints. The categorisation models are useful from documentation (specification) viewpoint, too.

Sommerville and Sawyer (1997) categorise requirements to functional and non-functional requirements: functional requirements describe what the system should do and non-functional requirements place constraints on how these functional requirements are implemented. Stevens et al. (1998) points that functionality itself is useless if the function is, for example, unreliable or not fast enough. They list following types of requirements that also may be necessary:

- performance requirements
- information relation ship and history requirements
- temporal and dynamic behaviour requirements
- requirements for parallelism or concurrency

- logical behaviour (e.g. conformance to a mathematical model)
- flow of control
- flows of data or material
- non-functional requirements (constraints)
- interactions with external systems
- end-to-end scenarios

SPICE (1998) processes and base practices that are related to system requirements development include:

- **CUS.3 Requirements elicitation process:** The purpose of the Requirements elicitation process is to gather, process, and track evolving customer needs and requirements throughout the life of the software and/ or service so as to establish a requirements baseline that serves as the basis for defining the needed software work products. Base practices include:
    - CUS.3.BP1: Obtain customer requirements and requests.
    - CUS.3.BP2: Agree on requirements.
    - CUS.3.BP3: Establish customer requirements baseline
    - CUS.3.BP4: Manage customer requirement changes
    - CUS.3.BP5: Understand customer expectations
    - CUS.3.BP6: Keep customers informed.

- **ENG.1.1 System requirements analysis and design process:** The purpose of the System requirements analysis and design process is to establish the system requirements (functional and non-functional) and architecture, identifying which system requirements should be allocated to which elements of the system and to which releases. Some of the following base practices (BP3, BP4, BP7) are related to the allocation and flow-down process phase (section 2.2):
    - ENG.1.1BP1: Identity system requirements
    - ENG.1.1BP2: Analyse system requirements (partly allocation and flow-down)

- ENG.1.1BP3: Describe system architecture (allocation and flow-down)
- ENG.1.1BP4: Allocate requirements (allocation and flow-down)
- ENG.1.1BP5: Develop release strategy
- ENG.1.1BP6: Communicate system requirements
- ENG.1.1BP7: Establish traceability

The following methods and techniques (of the RE methods described in section 3) are applicable to system requirements development activities:

- **Gathering/elicitation:**
  - Section 3.1: Brainstorming, Contextual inquiry, interviewing, observation, prototyping and scenarios.
  - Section 3.2: CORE, Ethnography, JAD, Protocol Analysis, QFD, REVEAL, SCRAM, SSM, VOSE and VORD.
  - Section 3.4: Volere.
  - Section 3.5: requirements reuse.
- **Analysis:**
  - Section 3.1: prototyping, scenarios.
  - Section 3.2: MPARN, QFD, REVEAL, SCRAM, SSADM, VORD and VOSE.
  - Section 3.3: SRA.
  - Section 3.4: SCR and WinWin.
  - Section 3.5: requirements modelling.

Outcome:

The results of system requirements development phase are captured as top-level requirements that are input for allocation and flow-down phase.

According to SPICE (1998) results of customer requirements elicitation process include:

- continuing communication with the customer will be established

- agreed customer requirements will be defined
- a mechanism will be established to incorporate new customer requirements into the established requirements baseline
- a mechanism will be established for continuous monitoring of customer needs
- a mechanism will be established for ensuring that customers can easily determine the status and disposition of their requirements
- enhancements arising from changing technology and customer needs will be identified and their impact managed.

According to SPICE (1998) results of system requirements analysis and design process include (partly results of allocation and flow-down):

- requirements of the system will be developed that match the customer's stated needs
- a solution will be proposed that identifies the main elements of the system (allocation and flow-down)
- the requirements will be allocated to each of the main elements of the system (allocation and flow-down)
- a release strategy will be developed that defines the priority for implementing system requirements
- the system requirements will be approved and updated as needed
- the requirements, proposed solution, and their relationship will be communicated to all affected parties.

2.2 Requirements allocation and flow-down

In this process, requirements and design (or architecture) work are closely linked. Requirements analysis is often defined as the “what” of a problem, it is implementation-free and contains objectives, not methods. Design is then the “how”, it describes the implementation that will meet the requirements. The two are supposed to be kept distinct, although, the feasibility of meeting a requirement must always be considered. (Dorfman, 1990)

The development of system-level requirements is, to the extent possible, a pure "what" addressing the desired characteristics of the complete system. Determining levels of the hierarchy and allocating system requirements to the elements, are a "how". They do not address objectives beyond the system requirements, but they define a subsystem structure that enables the requirements to be met. Flow-down is again a "what", determining what each element should do (functions, performance, etc.). Development of the architectural design is then a process in which the steps of requirements analysis and design alternate, with more detail brought out at each cycle. The output of requirements analysis is input to the next state of design and the output of design is input to the next stage of requirements analysis. (Dorfman, 1990).

Pure requirements analysis, like pure design, can only go so far, both disciplines are needed to achieve the desired result, a system that meets its users' needs. The characteristics of requirements analysis, like that of design, change as the work proceeds down in the hierarchy: requirements analysis for a lower-level element is much more detailed and involves knowledge of previous design decisions. The tools and methods used for analysis do in fact support all aspects of architectural design development: partitioning, allocation, and flow-down. They are useful, therefore, in both the requirements analysis and design stages of the process. (Dorfman, 1990) The methods include Shlaer-Mellor Object-Oriented Analysis Method, OMT, UML etc.

Purpose:

Requirements allocation and flow-down process is primarily a matter of making sure that all system requirements are fulfilled by a subsystem somewhere or by a set of subsystems collaborating together. Top-level system requirements will need to be organised hierarchically, helping to view and manage information at different levels of abstraction (Leffingwell & Widrig, 2000; Stevens et al., 1998).

Architectural models provide the context for defining how applications and subsystems interact with one another to meet the requirements of the system. Goal of architectural modelling, also commonly referred to as high-level modelling or global modelling, is to define a robust framework within which applications and component subsystems may be developed. Component diagrams may be used to describe the framework (component diagrams show

software components their interfaces and their interrelationships. (Ambler, 1998) More information of component diagrams can be found from (Rational, 1997; Booch, 1994).

Input:

Top level system requirements defined in system requirements development (see section 2.1) are the main input for requirements allocation and flow-down phase. As the system level requirements are being developed, also the elements that should be defined in the hierarchy, should be considered. By the time the system requirements are complete in draft form, a tentative definition of at least one and possibly two levels of system hierarchy should be available. This definition will include names and general functions of the elements. Definition of the system hierarchy is often referred to as "partitioning". (Dorfman, 1990).

Activities:

Requirements Allocation: Each system level requirement (with its requisite performance and interface characteristics) is allocated to one or more elements at the next level (i.e., it is determined which elements will participate in meeting the requirement). The allocation process is iterative; in performing the allocation, needs to change the system requirement (additions, deletions, and corrections) and/or the definitions of the elements may be found. (Dorfman, 1990; Pressman, 1992).

Allocation includes also allocating the non-functional requirements to system elements. Each system element will need an apportionment of the non-functional requirements (e.g., performance requirement). (Nelsen, 1990). However, all requirements are not allocable; non-allocable requirements are items such as environments, operational life and design standards, which apply unchanged across all elements of the system or its segments. (Sailor, 1990)

The flowing-down of top-level system requirements through lower levels of the hierarchy until the hardware and software component levels are reached is known as "top-down" analysis. In theory, it produces a system in which all elements are completely balanced or "optimised". In the real world, complete balance is seldom achieved, because fiscal, schedule, and technological

constraints result in some “flow-up” requirements. (Sailor, 1990). “Top-down” simply means that the requirements are decomposed from the top, or system level, and allocated down through subsystems, hardware equipment, and software programs in an orderly fashion. The requirements are decomposed down to the level at which the requirement can be designed and tested. Top-down analysis can be done in structured manner, “Structured” means that the analysis is logical in nature and is documented and implemented in a consistent manner throughout all levels of the system. (Nelsen, 1990). Methods that can be used in this include e.g., SSADM.

In addition to top-down analysis, also exploring options from a bottom-up and middle -out perspective is needed. Upper-level diagrams summarise the lower levels of functionality and behaviour, but we must not neglect the advantage of top-down analysis, particularly in analysing the interactions between levels. (Stevens et al., 1998)

The overall process of the evaluation of alternative system configurations (allocations) includes:

- Definition of alternative approaches.
- Selection of evaluation criteria – e.g. performance, effectiveness, life-cycle cost factors.
- Evaluation of alternatives.
- Application of analytical techniques (e.g., models).
- Data generation.
- Evaluation results.
- Sensitivity analysis (exposing areas, in which a small change in capability of the item would cause a large change in total system performance or cost).
- Definition of risk and uncertainty.
- Selection of the configuration. (Pressman, 1992; Blanchard & Fabrycky, 1981).

Example steps of allocation include (based on structural analysis) (Nelsen, 1990):

1. System is taken as a whole and all of the external inputs and outputs associated with the system are generally defined, including their source and destination. All of the overall system requirements are defined that would apply to this single transfer function.
2. The single system transfer function is decomposed into major elements. By using the functional requirements isolated from the customer's documentation by the system engineers, the requirements are categorised and grouped according to these general functions. All inputs and outputs from the first level will be included at this level, but rather than go into or out of one element they must be directed toward that element to which they apply. At this point the derivation of additional requirements begins, there are additional interfaces required between these second-level elements, which are not explicit system requirements, but are requirements derived from the system decomposition.
3. Each level 2 element is individually decomposed into the next lower-level functions. Level 3 allocations are those elements that will functionally satisfy all the requirements associated with one of the level elements and that will include all of the interfaces of that element. These elements may not be the lowest level, but may still be able to be decomposed further at the next level. As the process continues to lower functional levels, the derivation of requirements becomes a more significant part of the process and calls upon the expertise at the more detailed design level.

Once the functions of the system have been allocated, the system engineer can create a model that represents the interrelationship between system elements and sets a foundation for later requirements analysis and design steps. For example, an architecture template (Hatley & Pirbhai, 1987) can be used to develop system model (Hatley–Pirbhai Methodology (HPM)). (Pressman, 1992)

Requirements Flow-down: Flow-down consists of writing requirements for the lower level elements in response to the allocation. When a system requirement is allocated to a subsystem, the subsystem must have at least one requirement that responds to the allocation. Usually more than one requirement will be written. The lower-level requirement(s) may closely resemble the higher level one, or may be very different if the system engineers recognise a capability that the

lower level element must have to meet the higher-level requirements. In the latter case, the lower-level requirements are often referred to as “derived”. (Dorfman, 1990).

Derived requirements are requirements that must be imposed on the subsystem(s). These requirements are derived from the systems decomposition process, as such, alternative decompositions would have created alternative derived requirements. Typically there are two subclasses of derived requirements:

- Subsystem requirements are those that must be imposed on the subsystems themselves but do not necessarily provide a direct benefit to the end user.
- Interface requirements may arise when the subsystems need to communicate with one another to accomplish an overall result. They will need to share data or power or a useful computing algorithm. (Leffingwell & Widrig, 2000).

The level of detail increases as the work proceeds down in the hierarchy. That is, system-level requirements are general in nature, while requirements at low levels in the hierarchy are very specific. When flow-down is done, errors may be found in the allocation, the hierarchy definition and the system requirements. (Dorfman, 1990).

Software requirements analysis and specification is part of this activity, but is described in its own section (see section 2.3).

Interfaces: As each level of partitioning, allocation and flow-down takes place, the interfaces of each element to the rest of the system must be specified. This definition has two parts. First, interfaces defined at higher levels are made more specific, i.e., the external interfaces to the entire system are identified as to which subsystem(s) actually perform the interface. Second, internal interfaces at that level are defined, i.e., the subsystem-to-subsystem interfaces needed to enable each subsystem to meet the requirements allocated to it. Any interface between pairs of parent functions must appear between some of their children, with the flows in the same direction(s) and forming continuous chains at the lower level(s). (Dorfman, 1990; Stevens et al., 1998)

Traceability: The number of requirements proliferates rapidly during the allocation and flow-down process. Keeping track of these requirements is essential to make sure that all requirements are properly flowed down to all levels with no requirements lost and no "extras" thrown in. Reading the requirements becomes impossible, without a way to keep track of the flow-down path in the hierarchy. Establishment of traceability as allocation and flow-down are done helps to assure the validity of the work. Then, if changes are needed traceability enables locating the related requirement at higher and lower levels, that must be reviewed to see if they need to be changed. (Dorfman, 1990). Traceability is part of requirements management and is discussed in more detail in section 2.5.

The following methods (of the RE methods described in chapter 3) are applicable to requirements allocation and flow-down activities:

- **Allocation:**
  - Section 3.3: ATAM, Hatley-Pirbhai methodology and SRA.
  - Section 3.4: Booch methodology, HOORA and Jacobson method.
- **Flow-down:**
  - Section 3.3: ATAM and SRA.

Outcome:

The result of this process is detailed system level requirements and the "architectural design" or "top-level design" of the system. Although it is called a design, requirements engineering is involved throughout the process. (Dorfman, 1990) Also, needs to change the system requirement (additions, deletions, and corrections) and/or the definitions of the system elements may be found.

The decomposition is done right when:

- Distribution and partitioning of functionality are optimised to achieve the overall functionality of the system with minimal costs and maximum flexibility.
- Each subsystem can be defined, designed, and built by a small, or at least modest-sized team.

- Each subsystem can be manufactured within the physical constraints and technologies of the available manufacturing processes.
- Each subsystem can be reliably tested as a subsystem, subject to the availability of suitable fixtures and harnesses that simulate the interfaces to the other system.
- Appropriate deference is given to the physical domain - the size, weight, location, and distribution of the subsystems - that has been optimised in the overall system context. (Leffingwell & Widrig, 2000).

2.3 Software requirements analysis and specification

Requirements analysis is a software engineering task that bridges the gap between system level software allocation and software design. Requirement elicitation and analysis are closely linked processes. As requirements are discovered during the elicitation process, some analysis is done. When requirements are discovered, conflicts, overlaps, omissions and inconsistencies should be analysed. (Kotonya & Sommerville 1998; Sommerville & Sawyer, 1997; Pressman, 1992)

Purpose:

The software requirements analysis is the activity of determining what functions the software will perform and documenting those functions and other requirements in a software requirements specification. Requirements analysis enables the specification of software functions and performance, indication of software's interface with other system elements, and establishment of design constraints that the software must meet. Requirement analysis also refines the software allocation and builds models of the process, data, and behavioural domains that will be treated by software. Requirements analysis provides a representation of information and function that can be translated to data, architectural and procedural design. Finally the requirements specification provides the developer and the customer with the means to assess quality once the software is built. (Pressman, 1992)

Through the system mechanism of flow-down, allocation, and derivation, a software requirements specification will be established for each software subsystem, software configuration item, or component. A requirements traceability mechanism will be established that will generate an audit trail between the system/software requirements and final tested code. (Thayer & Royce, 1990)

Input:

The starting point for the analysis is the detailed system level requirements allocated for the system component software.

Activities:

According to SPICE (1998) the following base practices belong to software requirements analysis process:

- Specify software requirements: Determine and analyse requirements of the software components of the system and document in a software requirements specification.
- Determine operating environment impact: Determine the interfaces between the software requirements and other components of the operating environment, and the impact that the requirements will have.
- Evaluate and validate requirements with customers: Communicate the software requirements to the customer, and based on what is learned through this communication, revise if necessary.
- Develop validation criteria for software: Use the software requirements to define the validation criteria for the software. The validation criteria are used in developing the software tests.
- Develop release strategy: Prioritise the software requirements and map them to future releases of the software.
- Update requirements for next iteration: After completing an iteration of requirements, design, code, and test, use the feedback obtained from use to modify the requirements for the next iteration.

- Communicate software requirements: Establish communication mechanisms for dissemination of software requirements, and updates to requirements to all parties who will be using them.
- Establish traceability: Establish traceability between the system requirements and the software requirements.

Software requirements stage has two distinct activities: problem analysis and product description:

- *Problem analysis i*s the activity that encompasses learning about the problem to be solved (often through brainstorming and/or questioning), understanding the needs of potential users, trying to find out who the user really is, and understanding all the constraints on the solution. Problem analysis is defining the product space, i.e., the range of all possible software solutions that meet all known constraints. The needs of user, customer and the development organisation need to be defined. The needs of users can be uncovered by watching the users to do the job (methods such as Ethnography or SSM) and allowing users to play with experimental early versions of the product (prototyping). Techniques that can be used in problem analysis include: Data Flow diagrams (DFD), Entity-Relation diagrams (ERD), Coad Object diagrams, Listing all Inputs and Outputs, Listing major functions, Structured Requirements Definition (SRD), Structured Analysis and Design technique (SADT), Structured Analysis and System Specification (SASS), and Object Oriented Analysis (OOA). (Davis, 1990; Thayer & Royce, 1990). Also Object oriented methods and UML can be used to this activity (see Douglass, 1999; Jacobson et al., 1999).

- *Product description* mainly involves writing the software requirements specification (SRS), including both behavioural and non-behavioural requirements. SRS contains a complete description of the external behaviour of the software system. It is possible to complete the entire problem analysis before starting to write the SRS. However, it is more likely that as the problem analysis decomposition process yields aspects of the problem that are well understood, the corresponding section of the SRS is written. SRS's purpose is to provide means of communicating among customers, users, analysts, and designers, supporting system-testing activities and controlling

the evolution of the system. For example, IEEE defines contents of an SRS. (IEEE Std 830–1998). The standard doesn't describe sequential steps to be followed, but defines the characteristics of a good SRS and provides a structure template for the SRS. This template can be used in documenting the requirements, and also as a checklist in other phases of the requirements engineering process. (Davis, 1990)

Usually the requirements are documented using natural language and diagrams, so that all the system stakeholders can understand them. (Kotonya & Sommerville, 1998; Sommerville & Sawyer, 1997). The primary purpose for using a requirements specification technique is to reduce the inherent ambiguity of natural language and to ease detecting inconsistencies, redundancies, incompleteness, and ambiguities. Examples of techniques for real time systems include: Finite state machines, decision tables and decision trees, Program Design Language (PDL also known as structured English and pseudo code), Statecharts, Requirements Engineering Validation system (REVS), Requirements Language Processor (RLP), Specification and Description Language (SDL), PAISLey (Zave, 1982), Petri nets. (Davis, 1990)

Software requirements are commonly classified to (Jacobson et al., 1999; Thayer & Royce, 1990):

- Functional: A requirement that specifies an action that a system must be able to perform, without considering physical constraints; a requirement that specifies input/output behaviour of a system
- Non-functional: A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. A requirement that specifies physical constraints on a functional requirement. (Jacobson et al., 1999).
    - *Performance requirements*. A requirement which specifies performance characteristics that a system or system component must possess, e.g., max. CPU-usage, max. memory footprint.
    - *External interface requirements.* A requirement which specifies hardware, software, or database elements with which a system or

system component must interface, or that sets forth constraints on formats, timing or other factors caused by such an interface.

  - *Design constraints.* A requirements which affects or constrains the design of a software system or software system component, e.g. language requirements, physical hardware requirements, software development standards, and software quality assurance standards.
  - *Quality attributes.* A requirement which specifies the degree to which software possesses of attributes that affects quality, e.g. correctness, reliability, maintainability, portability.

- or behavioural and non-behavioural requirements (Davis, 1990).

Methods used in Requirements Engineering are commonly not isolated, but used within a certain software engineering methodology, which encompasses some other software lifecycle phases as well, or combined with another approach. For example, object oriented analysis methods and agile methods are used also during some other software development phases.

The following methods (of the RE methods described in section 3) are applicable to software requirements analysis and specification activities:

- **Problem analysis**:
  - Section 3.1: Prototyping and scenarios.
  - Section 3.4: Booch methodology, HOORA, Jacobson method, OMT, Planguage, RUP, SADT, SASS, SCR, Shlaer-Mellor, SREM, Storyboarding, UML, and COTS-specific methods CARE and OTSO.
  - Section 3.5: Requirements modelling and requirements reuse.
- **Product description:**
  - Section 3.2: CORE and VORD.
  - Section 3.4: B-method, HOORA, Jacobson method, Planguage, RUP, SADT and SCR.
  - Section 3.5: B-method, Petri nets, VDM and Specification Language Z.

Outcome:

Formal, agreed upon baseline of software requirements. According to SPICE (1998) as a result of successful implementation of the process:

- The requirements allocated to software components of the system and their interfaces will be defined to match the customer's stated needs.
- Analysed, correct, and testable software requirements will be developed.
- The impact of software requirements on the operating environment will be understood.
- A software release strategy will be developed that defines the priority for implementing software requirements.
- The software requirements will be approved and updated as needed.
- Consistency will be established between software requirements and software designs.
- The software requirements will be communicated to affected parties.

2.4 Continuous activities in RE

This section describes requirements engineering activities that are continuously ongoing. The activities include requirements documentation, requirements validation and verification and requirements change management.

2.4.1 Requirements documentation

Requirements have to be documented to enable communication and agreement of the requirements as well as traceability to other work products. These documents, i.e., specifications, can be written in a natural language or by using a requirements statement language (e.g., formal methods). Also, modelling techniques should be used to describe requirements (e.g., UML, Data Flow Diagrams, etc.). System requirements specification is discussed in more detail in section 2.1 and software requirements specification in section 2.3.

Requirements specification document should follow a standard format, several formats have been proposed in literature, e.g., IEEE Std 830–1998; Davis, 1990 and Leffingwell & Widrig, 2000.

The specifications are defined in a specification tree that follows the system hierarchy. Each deliverable item (or assembly of items) should have a specification, against which the acceptance of the item is reflected. After the specifications are ready, an overall baseline design is established. (Sailor, 1990) Different requirements documents may be needed also for different viewpoints (Leffingwell & Widrig, 2000):

- The system is very complex → One "parent" document, that defines requirements of the overall system (system requirement specification), including hardware, software, people, and procedures, and another document defines requirements for just the software piece (software requirements specification i.e., SRS).

- The customer's needs are being documented prior to documenting detailed requirements → One document defines the features of the system in general terms (vision document), and another defines requirements in more specific terms (SRS).

- The system may be a member of a family or related products or the system being constructed satisfies only a subset of all the requirements identified → One document defines the full set of requirements for a family of products (product family requirements document or product family vision document), and another defines requirements for just one specific application and one specific release (SRS).

- Marketing and business goals need to be separated from the detailed product requirements → One document describes the overall business requirements and business environment in which the product will reside (business requirements document or marketing requirements document), and another defines the external behaviour of the system being built (SRS).

The following methods (of the RE methods described in section 3) are applicable to requirements documentation activities:

- Section 3.5: B-method, Petri Nets, VDM, Specification language Z and requirements modelling.

Part of requirements documentation is identification of requirements, that is, assignment of unique identifier for each requirement. This is discussed more in section 2.5.

2.4.2 Requirements validation and verification

Validation means actions to confirm that the behaviour of a developed system meets user needs whereas verification means actions to confirm that the product of a system development process meets its specification, for example, the design must meet the system requirements. (Stevens et al., 1998) Requirements verification is a continuous process starting with design reviews in the early phases of system development and ending with review of data from the system tests. (Sailor, 1990)

Tasks of verification and validation include:

- Defining the verification requirements, i.e., principles on how the system will be tested.
- Planning the verification.
- Capturing the verification criteria (when the requirements are defined).
- Planning of test methods and tools.
- Planning and running of reviews.
- Implementing and performing the tests and managing the results.
- Maintaining traceability.
- Auditing.

Generally used methods for requirements validation and verification include reviews (section 3.5), prototyping (section 3.1) and various testing methods (not described in more detail in this publication). Also, methods used in system

requirement development, requirement allocation and flow-down and software requirements analysis and specification activities often include activities for requirements verification and validation.

2.4.3 Requirements change management

After the requirement baseline is established all changes to the requirements must be controlled. This is discussed in more detail in section 2.5.

2.5 Requirements management viewpoint

Requirements management (RM) can be seen as a parallel support process for other requirements engineering processes (Sommerville & Sawyer, 1997; Kotonya & Sommerville, 1998). It also continues after requirements specification phase, because the requirements continue to change during system development and these changes must be managed (Kotonya & Sommerville, 1998). Sommerville and Sawyer (1997) define principles concerning requirements management:

- Changes to the agreed requirements need to be managed.
- Relationships between requirements should be described and managed.
- Relations between requirements documents and other documents produced during the systems and software engineering need to be managed.

According these principles the main concerns of requirements management are change control and traceability. The last principle emphasises traceability support between requirements and design, implementation, and test artefacts. This principle connects managed requirements with the design, implementation and test items to ensure traceability, for example, during verification and change. On the other hand, according Kotonya and Sommerville (1998) requirements identification is an essential pre-requisite for requirements management. It focuses on the assignment of unique identifier for each requirement (Sommerville & Sawyer, 1997). These unique identifiers are used to refer to requirements during product development and management. Sommerville and

Sawyer (1997) propose guidelines for RM, which extend RM concepts to cover also RM planning and automation issues.

RM can be organised under four main RM activities as follows (Figure 3):

- Requirements identification: all guidelines which relate to the identification and storage of requirement items.
- Requirements traceability: all guidelines which relate to the requirements traceability.
- Requirements change management: all guidelines, which relate to the requirements change management.
- Requirements management planning: guidelines, which relate to the planning and documentation of identification, traceability and change management activities as well as the definition of other RM goals, responsibilities and policies for a project. Planning provides means to select and define suitable RM procedures when considering RM for a project.

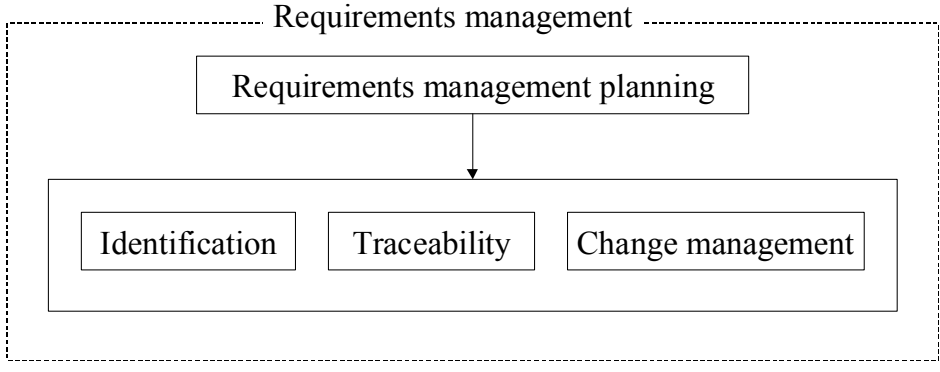


Figure 3: Main activities for RM.

Requirements identification

Requirements identification focuses on the assignment of unique identifier for each requirement, which can be used to unambiguously refer to a requirement during product development and management (Sommerville & Sawyer, 1997). It is an essential pre-requisite for requirements management (Kotonya &

Sommerville, 1998). Requirements may evolve during product development (Sommerville & Sawyer, 1997). For example, customer's expectations may change or be misunderstood, which causes updates for system's requirements. Thus the management of requirement versions is worthwhile. Requirements are normally also organised hierarchically (Stevens et al., 1998; Crnkovic et al., 1999). This allows increasing their specificity through parent-child -relations (Leffingwell & Widrig, 2000). Leffingwell and Widrig (2000) introduce that identification scheme could allow the description of parent-child relations. For example, if parent requirement is identified as SR63.1 then child requirements could be identified as SR63.1.1, SR63.1.2, etc.

Leffingwell and Widrig (2000) emphasise that by using attributes you get better management of complexity of information. Requirement attributes can be used to record additional information about requirements (Leffingwell & Widrig, 2000; Stevens et al., 1998). They can be included into requirements to describe information e.g. about requirement classification, priority, status, version, relations, rationale, etc. Some attributes can have pre-defined set of values. For example, status -attribute could have values identifying requirement's maturity and its progress through product development (Leffingwell & Widrig, 2000):

- proposed (under discussion)
- approved (approved for implementation)
- incorporated (to product baseline).

Attributes used in a project and their meaning should be documented to ensure their better understanding and correct usage in the project (Leffingwell & Widrig, 2000). Clearly defined attributes can systematise the capture of requirement-related information.

Requirement baseline forms the foundation for design phase (Hooks & Farry, 2001). Hooks and Farry (2001) describe baselining as a technique for "drawing the line in the sand". This means that requirement baselining unambiguously identifies certain agreed set of requirements (requirements specification) for product's design. Baselining does not mean that requirements can not change after it. However, after baselining the changes to the requirements need to be incorporated using change control procedures to avoid uncontrolled modifications to the requirement baseline.

Requirements traceability

Requirements traceability (RT) refers to the ability to describe and follow the life of a requirement in both a forwards and backwards direction (Gotel, 1995). The importance of traceability is recognised also in standards (e.g. (ISO/IEC 12207, 1995) and (IEEE Std 830–1998)). Kotonya and Sommerville (1998) present requirements traceability as a critical part of requirements change management. During change management it is used to assess the impact of a change to understand related elements which likely will be affected.

Gotel (1995) emphasises the life cycle aspect of the traceability. Requirements form the basis for design and implementation activities and they should be traceable through product's life. Requirements' traceability information need to be recognised, stored, and retrieved to support, for example, verification and change management activities. Gotel (1995) distinguishes traceability into vertical and horizontal elements. Furthermore, these can be divided into forwards and backwards traceability as illustrated in *Figure 4*.

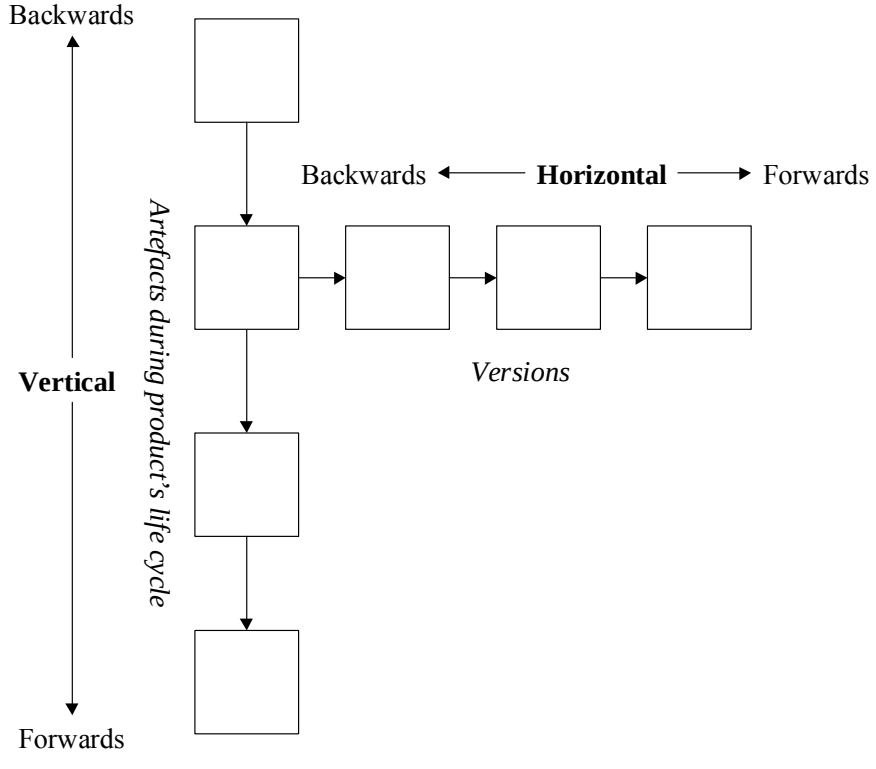


Figure 4: Horizontal and vertical requirements traceability (adapted from (Gotel, 1995), p.37).

The dimensions of traceability (forwards, backwards, vertical and horizontal) are further explained in *Table 3*.

Table 3: Dimensions of traceability.

Dimensions	Forwards	Backwards
Vertical	Traceability of requirements forward from specification to implementation and test.	Traceability of requirements into its sources (customer requirements).
Horizontal	Traceability of requirement versions to forward direction.	Traceability of requirement versions to backwards direction.

Kotonya and Sommerville (1998) emphasise the direction of traceability and further extend vertical traceability with ”to” and ”from” attributes as well as with traceability between requirements and external documents. This classification describes bi-directional nature of RT (extended by Kotonya and Sommerville (1998) based on Davis (1990)):

- Forward-to traceability: traceability of sources (customer requirements, system level requirements, etc) to requirements
- Forward-from traceability: traceability of requirements to design specifications
- Backward-to traceability: traceability of design specifications to requirements
- Backward-from traceability: traceability of requirements to their sources (customer requirements, system level requirements, etc)
- Links between requirements and links between requirements and external documents (e.g. rationale)

Gotel and Finkelstein (1994a) separate two main types of requirements specification traceability from forwards and backwards traceability as follows:

- Pre-traceability: is concerned with those aspects of a requirement's life prior to its inclusion in the requirements specification (RS) (requirements production).
- Post-traceability: is concerned with those aspects of a requirement's life that result from its inclusion in the RS (requirements deployment).

Term "requirements specification (RS)" refers here to requirements baseline (Gotel & Finkelstein, 1994a). Gotel and Finkelstein (1994a) and Gotel (1995) also introduce that post-traceability type is well supported. On the other hand, they state that pre-traceability is poorly understood and not comprehensively supported.

Kotonya and Sommerville (1998) describe the set of RT policies, which need to be defined for an organisation:

- The traceability information which should be maintained
- The techniques used to maintain relations
- Schedule for the collection of traceability information
- Responsibilities for maintaining traceability information
- Description of how to handle and document policy exceptions
- The process used to ensure that the traceability information is updated after the change has been made

Sommerville and Sawyer (1997) concrete requirements traceability by defining information types for it. These types describe specific information, which need to be linked together (Sommerville & Sawyer, 1997): requirements-sources, requirements-rationale, requirements-requirements, requirements-architecture, requirements-design and requirements-interface.

Kotonya and Sommerville (1998) and Stevens et al. (1998) state that after system level requirements specification architectural design divides and assigns system level requirements into sub-system level entities, which are further specified and divided into smaller entities. Thus specification happens in various levels during system decomposition in co-operation with architectural design until sufficient practical implementation level has been reached. Hooks and

Farry (2001) emphasise requirements traceability between these levels to ensure that all requirements are flowed from the top, through all requirements levels.

Requirements change management

Requirements change management refers to the ability to manage changes to the systems requirements (Kotonya & Sommerville, 1998). It also ensures that similar information is collected for each proposed change and that overall judgements are made about the costs and benefits of proposed change. Requirements are baselined when moving to the design phase. However, we can never start the design if we wait until we know everything about the requirements. Thus the requirements should be ”good enough” to proceed with design (Hooks & Farry, 2001). Even if requirement specification is comprehensive something can change during development, for example, customer’s needs or regulations. This causes the need for clear practices, which guide how possible changes to the baselined requirements are handled when changes to requirements are recognised.

Kotonya and Sommerville (1998) describe the set of policies, which need to be defined for the organisation:

- The requirements change management process (consisting tasks like change request and impact analysis) with the description of needed information.
- Responsibles for the tasks described in the change management process (e.g. Change Control Board who consider change requests).
- The software support for the change management process.

Basic elements of postbaseline requirements change management process consists of phases for (Hooks & Farry, 2001):

- Documenting and evaluating the change justification.
- Performing a thorough change impact assessment.
- Making the decision to approve or reject the change.
- Implementing the change if approved.

The change can cause impacts for schedule and budget. Mäkäräinen (2000) emphasise that requirement level modification type of change usually has effects

to the project planning (revision of project plans concerning schedule, budget, and resources). The above process concerns requirement changes, but Leffingwell and Widrig (2000) state that change in design or implementation (e.g. source code) can also lead to the changes to the requirements. So also in these cases the impact to the requirements should be assessed.

Leffingwell and Widrig (2000) introduce five guidelines to manage changes effectively:

1. Recognise that change is inevitable, and plan for it: change procedures for the project should be planned.
2. Baseline the requirements: baseline provides clear concept to identify the set of requirements, which are used in a design phase. It provides mechanisms to distinguish, which set of requirements is ”old” and which requirements have changed or evolved after baselining.
3. Establish a single channel to control change: for example, in large systems Change Control Board (CCB) who share the responsibility about the approval of change requests. In small systems the responsibility can be given e.g. to someone who is an owner of the artefact or person who has an overall understanding of system requirements.
4. Use a change control system to capture changes: change control system should be used to capture all requested system related changes. Change requests should be transmitted to CBB for decision making.
5. Manage change hierarchically: changes to the requirements should be managed in top-down hierarchical fashion. For example, changes to the specification can cause changes to the features or to design, implementation and test.

In addition, Leffingwell and Widrig (2000) state that configuration management (CM) -based requirements management can be useful for a project. Change management involves to the large amounts of information. Versions from requirements and other artefacts are linked to the change objects and this information is transferred from people to people and change related information is collected during the process (Leffingwell & Widrig, 2000). CM-based approach can support the complexity of change management by preserving the revisions to requirements documents, facilitating the retrieval and reconstruction

of previous versions of documents, supporting a managed baseline ”release strategy” for incremental improvements or updates to a system, and preventing simultaneous update (locking) the document (Leffingwell & Widrig, 2000).

RM planning

Sommerville and Sawyer (1997) introduce the definition of requirements management policies as one guideline for RM. These policies define goals and procedures for RM, which should be followed in an organisation. Sommerville and Sawyer (1997) state that an organisation can have general policies for RM, but each project should evaluate their applicability and select and maybe tailor relevant ones for the project. Training is used to ensure that people are aware of the policies and know how to apply them (Sommerville & Sawyer, 1997). It is unrealistic to define the total set of generic policies for an organisation at once. Thus they need to be developed incrementally according practical experiences (Figure 5).

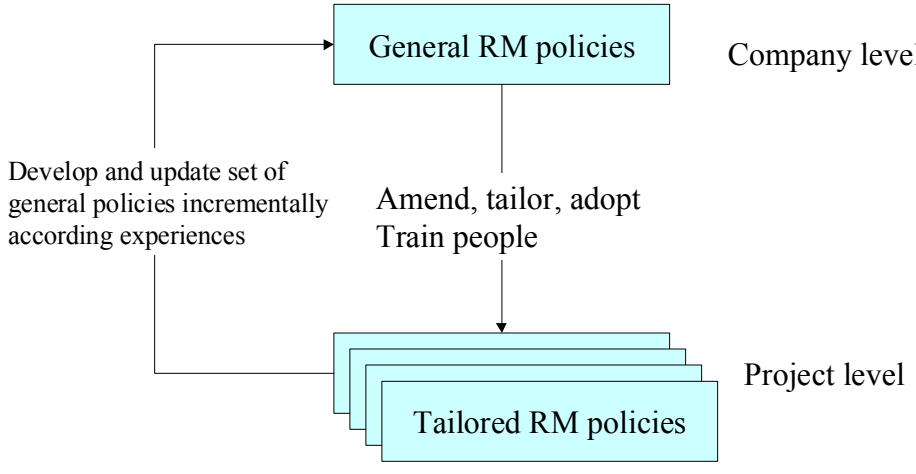


Figure 5: Tailoring general RM policies for projects.

Sommerville and Sawyer (1997) introduce the issues for generic policies, which need to be defined:

- Objectives with rationale for the RM process.

- Reports which will be produced to increase visibility and activities which will produce these reports.
- Standards for the requirements documents and requirements descriptions (templates, structures): this defines the form or template used to record requirements information including fields to collect the detailed information necessary to completely specify the requirement.
- Requirement change management policies: responsibilities, procedures and SW support to requirements change management.
- Review and validation policies.
- Relationship between requirements management and other systems engineering and project planning activities.
- Traceability policies: responsibilities, schedules, information types and techniques for requirements traceability.
- Criteria when these policies can be ignored. These special practices provide flexibility into policies e.g. when there is urgent needs for system changes.

According Sommerville and Sawyer (1997) the implementation of RM policies takes at least a year of calendar time. They also state that the mechanisms for checking that the procedures are being followed are needed. Sommerville and Sawyer (1997) further discuss requirements change management and traceability policies. They state that the implementation of the change management policy is fairly expensive and the automation of process is essential. The automation is needed to due to the volume of information, which must be processed. On the other hand, Sommerville and Sawyer (1997) state that the costs to implement traceability depend on defined policies and the number of requirements for your system. Kotonya and Sommerville (1998) describe which factors influence to the traceability policies: number of requirements, estimated system lifetime, level of organisational maturity, project team size and composition, type of system and specific customer requirements.

3. Requirements engineering methods

This chapter presents requirements engineering methods and techniques, which have been gathered and documented in the performed literature study. General methods, which can be utilised during different phases of the RE process, are presented in the first subsection of this chapter. Then phase-specific methods are presented grouped under respective requirements engineering process phases according to the ones described in chapter 2. A short overview of the methods and techniques is provided followed by pointers to sources of further information.

Table 4 presents the symbols, which have been used in method descriptions to illustrate different types of sources of further information.

Table 4 Symbols of information sources used in method descriptions.

Symbol	Type of information source
	Article about method e.g. in conference proceedings or technical journal.
	Book about method, or parts of a book.
	Information about method's origins, e.g. method's developer or publication where the method was first introduced.
	WWW-resources on method, e.g. method's homepage or a web tutorial.
	Tool supporting the usage of the method.
	Experience report or a case study of method's usage.

The books in the further information section have been selected based on their age, i.e. the most recent ones have been included. The tools in the further information section have been selected so, that an example of found tools has been included (if any tools have been found in the literature). Links to more comprehensive tool lists is presented in section 4.3.

3.1 General methods

In this section general methods, which can be used during several requirements engineering process phases, are described. VTT Publication about agile software development methods (Abrahamsson et al., 2002), presents agile methods, which also provide support for requirements engineering activities.

Brainstorming

Brainstorming is a group technique for generating new, useful ideas and promoting creative thinking. It aims at getting a large number of ideas from a group of people in a short time. (Vierimaa et al., 2001). In requirements engineering, brainstorming can be used to discover unidentified requirements and to elicit requirements from stakeholders.

Further information:


Vierimaa, M., Ronkainen, J., Salo, O., Sandelin, T., Tihinen, M., Freimut, B. & Parviainen, P. 2001. MIKKO Handbook: Comprehensive collection and utilisation of software measurement data. Pages 96–100. Technical Research Centre of Finland. VTT Publications 445.


Rawlinson, J.G. 1981. Creative Thinking and Brainstorming. Gower Publishing Company Limited.


Brainstorming homepage. URL: http://www.brainstorming.co.uk/

Contextual Inquiry

Contextual Inquiry (CI) is a structured field interviewing method for gathering data from the structure of work practice, making unarticulated knowledge about the work explicit so that designers can understand it, and getting at the low-level details of work that has become habitual and invisible. Contextual Inquiry is based on a few core principles:

- Understanding the context in which a product is used.
- Considering the user as a partner in the design process.
- The usability design process must have a focus.
- Interpretation.

- CI is more a discovery process than an evaluative process. It is also more like learning than testing. (Beyer & Holtzblatt, 1998)

Further information:

Beyer, H. & Holtzblatt, K. 1998. Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann Publishers, Inc.

Facilitated meeting

The purpose of these techniques is to try to achieve a comprehensive effect whereby a group of people can bring more insight to their requirements than by working individually. Using facilitated meeting techniques may result in a richer and more consistent set of requirements that might otherwise be achievable. (Sawyer & Kotonya, 2001)

One facilitated meeting and collaboration technique is the Bridge Methodology. Its purpose is to facilitate system stakeholders' communication and involve them all in the phases of system development. In Bridge, a user-centred task flow, a set of abstract task objects, and a paper prototype of a graphical user interface (GUI) are produced. The Bridge has three major steps: task analysing, task-to-object mapping, and object-to-GUI mapping. (Dayton & Tournet, 1999)

Further information:

Dayton, T. & Tournat, K. 1999. How to Design, Prototype, and Test A Usable GUI In Three Days. URL: http://www.baychi.org/meetings/archive/0399.html

Sawyer, P. & Kotonya, G. 2001. Software Requirements. In the Trial Version (0.95) of SWEBOK, Guide to the Software Engineering Body of Knowledge. Chapter 2, pages 1–26. http://www.swebok.org/ (available 26.3.2002)

Interviews

Interviewing is a technique applicable to requirements elicitation. In practice interviews are discussions between the requirements engineer and different

stakeholders about the system with a purpose of building up an understanding of stakeholders' requirements. There are two types of interviews:

- Structured (closed) interview, which is done with pre-defined set of questions by requirement engineer.
- Open interview, which is a discussion about what stakeholders want from the system without pre-defined agenda performed in an open-ended way. (Kotonya & Sommerville, 1998)

Further information:


Judd, C.M.; Smith, E.R. & Kidder, L.H. 1991. Research Methods in Social Relations, International edition. Sixth ed. Holt Rinehart and Winston, Inc.


The University of Western Australia's tutorial on interviewing. URL: http://undergraduate.cs.uwa.edu.au/units/670.300/readings/reqs.elicit.Interviewing.htm

Observation

The purpose of this technique is to observe stakeholders in the field. The requirements engineer learns about user' tasks by immersing themselves in the environment and observing how users interact with their systems and each other. (Kotonya & Sommerville, 1998) One observation technique is Ethnography, which is presented later in subsection 3.2.

Further information:


Lofland, J. & Lofland, L.H. 1995. Analyzing social settings: A Guide to Qualitative Observation and Analysis. Wadsworth Publishing Company.

Prototyping

A prototype of a system is an initial, ideally quick to make and inexpensive version of the system, which is available in an early phase of the development process. Prototypes may be used to help elicit and analyse system requirements.

There are many different techniques and approaches for prototyping, but they usually fall into one of the two main categories: throwaway prototyping and evolutionary prototyping. The main difference in these two is that in throwaway prototyping a disposable prototype is created to help elicit and analyse system requirements, which are hard to understand for the customer. In evolutionary prototyping a workable system prototype with limited functionality is made available to the users early in the development process and modified and extended later to produce the final system. (Kotonya & Sommerville, 1998)

Further information:

Andriole, S.J. 1994. Fast, cheap requirements prototype, or else! IEEE Software. Vol. 11. Issue 2. Pages 85–87.

Scenarios

Scenarios are examples of interaction sessions, and consist of descriptions of sequential actions. Scenarios are useful, because end-users and other system stakeholders find it easier to relate to real-life examples rather than abstract descriptions of the functions. This makes scenarios applicable for eliciting and clarifying system requirements. Scenarios should include at least the following kind of descriptions:

- The state when entering the scenario and state after completion of the scenario.
- The normal flow of events and exceptions to the normal flow of the events.
- Information about concurrently ongoing things.

When end-user simulates the use of the system by following the scenarios the engineer does memos of user's comments, problems and suggestions at the same time. (Kotonya & Sommerville, 1998; Sommerville & Sawyer, 1997)

Further information:

Cox, K. 2000. Fitting scenarios to the requirements process. Proceedings of the 11th Workshop on the Database and Expert Systems Applications. Pages 995–999.

3.2 System Requirements development

This section presents methods and techniques, which can be utilised during the system requirements development phase of the requirements engineering process, either alone or supporting other methods. Some methods and techniques cover several RE phases, and therefore the classification is rather suggestive than explicit. A short overview of the method is provided followed by pointers to sources of further information.

The methods and techniques presented in this section are:

- Controlled Requirements Expression (CORE)
- Ethnography
- Joint Application Development (JAD)
- Multi-criteria Preference Analysis Requirements Negotiation (MPARN)
- Protocol Analysis
- Quality Function Deployment (QFD)
- REVEAL
- Scenario-based Requirements Engineering (SCRAM)
- Structured System Analysis and Design Methodology (SSADM)
- Soft System Methodology (SSM)
- Viewpoint-Oriented Requirements Definition (VORD)
- Viewpoint-oriented Software Engineering (VOSE)

Controlled Requirements Expression (CORE)

CORE is a method, which is applicable to the needs of requirement expression. It is supported by a simple diagrammatic notation, which can represent many vital viewpoints of the system requirements. CORE defines the steps in production of a requirement specification, with particular emphasis on start-up and the link between steps. For each step, CORE also defines the activities involved and the checks to be applied. (Mullery, 1979) In requirements engineering, CORE is best suited for gathering requirements.

Further information:


Mullery, G.P. 1979. CORE – A Method for Controlled Requirement Specification. Proceedings of IEEE Fourth International Conference on Software Engineering.


Developed by G. P. Mullery at Systems Designers Limited.

Ethnography

Ethnography is a technique for observing users in real-life or simulated operating situations. It can be used for evaluating the strengths and weaknesses of an existing system in order to capture design requirements for the system under development, or for task analysis when determining the steps to be performed by the system under development in order to complete the desired tasks. Ethnography also helps in identifying system interfaces. In requirements engineering, ethnography is best suited for gathering requirements from users.

Further information:


Suchman, L. 1983. Office procedures as practical action. ACM Transactions on Office Information Systems. Pages 320–328.


Sommerville, I.; Rodden, T.; Sawyer, P.; Bentley, R. & Twidale, M. 1993. Integrating ethnography into the requirements engineering process. Proceedings of the IEEE International Symposium on Requirements Engineering. Pages 165–173.


Ethnography was originally developed by anthropologists to understand social mechanisms in 'primitive' societies. L. Suchman first identified the potential value of ethnography in deriving computer system requirements.


Coherence project is a three-year EPSCR-funded project investigating the problem of integrating ethnographically informed analysis into a systematic approach to systems design. URL: http://www.comp.lancs.ac.uk/computing/research/cseg/projects/coherence/index.html


The Ethnograph is a software tool, which can be used to support ethnography. URL: http://www.scolari.co.uk/ethnograph/ethnograph.htm

Bentley, R.; Hughes, J.A.; Randall, D.; Rodden, T.; Sawyer, P.; Shapiro, D. & Sommerville, I. 1992. Ethnographically informed systems design for air traffic control. Proceedings of Conference on Computer Supported Cooperative Work. ACM Press. Pages 123-129.

Joint Application Development (JAD)

JAD is a structured, facilitated meeting (a JAD session) where modelling is performed by both users and developers together. A trained facilitator leads JAD sessions. User requirements are often gathered in JAD sessions where the facilitator leads the group through prototyping and/or the definition of use cases. (Ambler, 1998) In requirements engineering, JAD is best suited for gathering requirements from different stakeholders.

Further information:

Ambler, S. W. 1998. Process Patterns. Building Large- Scale Systems Using Object Technology. Cambridge University Press.

Wood, J. & Silver, D. 1989. Joint Application Design. John Wiley & Sons.

JAD was originally developed at IBM in the late 1970's.

Multi-criteria Preference Analysis Requirements Negotiation

MPARN assists stakeholders in evaluating, negotiating, and agreeing upon different alternatives using multi-criteria preference analysis techniques. The purpose of the MPARN process is to generate mutually agreed criteria, assess alternatives' performance on each criterion, and to elicit each group participants' relative preference over each alternative. During the process the needs of all stakeholders and conflicts in these needs are identified and options of conflict-resolution are explored. (In et al., 2001) In requirements engineering, MPARN is best suited for performing top-level analysis to raw requirements.

Further information:

In, H.; Olson, D. & Rodgers, T. 2001. A Requirements Negotiation Model Based on Multi-Criteria Analysis. International Symposium on Requirements Engineering.

Protocol Analysis

Protocol Analysis is a technique, which can be used for forming an understanding of users' thought patterns when performing a given, structured task. In Protocol Analysis, users perform the given task and simultaneously explain their thought processes, while an observer records the verbal data. After the session the verbal data is analysed, and the real thought process can be compared to the system's process. Thus the system's strengths and weaknesses can be identified. (Ericsson & Simon, 1993) In requirements engineering, Protocol analysis is best suited for gathering requirements from users.

Further information:

Ericsson, K.A. & Simon, H. A. 1993. Protocol Analysis - Revised Edition. MIT Press.

Quality Function Deployment (QFD)

Quality Function Deployment (QFD) is a methodology for identifying customer needs, expectations and requirements, and linking them into the company's products. This is done by determining customer preferences and their importance and translating them into product design requirements. QFD utilises different diagrams, tables, charts and matrices to facilitate its different tasks. In requirements engineering, QFD is suitable for gathering requirements, and performing top-level and detailed analyses to them.

Further information:

Revelle, J.B.; Moran, J.B.; Cox, C.A. & Moran, J.M. 1998. The QFD Handbook. John Wiley & Sons.


Developed by Yoji Akao in Japan in the 1970's. Introduced to United States by Akao in 1983.


QFD Institute's home page. URL: http://www.qfdi.org/


QFD2000 is a software tool, which can be used to support QFD. URL: http://www.qfd2000.co.uk/introduction.htm


Graves, S.B.; Carmichael, W.P.; Daetz, D. & Wilson, E. 1991. Improving the Product Development Process. Hewlett Packard Journal. June 1991.

REVEAL

REVEAL is a systematic principled method for the elicitation, specification, and management of systems requirements. It provides guidelines for the RE activities from the early phases of the establishment of problem context, through to the identification of stakeholders, the elicitation and recording of requirements, their verification and validation, and on to their use and maintenance. (Hall, 2001) In requirements engineering, REVEAL is best suited for gathering requirements and performing top-level analysis to them.

Further information:


Hall, A. 2001. A Unified Approach to Systems and Software Requirements. The 5th IEEE Symposium on Requirements Engineering.

Scenario-based Requirements Engineering (SCRAM)

Scenario-based Requirements Engineering (SCRAM) is a method that combines use cases approaches with object oriented development. In SCRAM scenarios are created to represent paths of possible behaviour through a use case, and these are then investigated to develop requirements. SCRAM can also be utilised in requirements documentation. In requirements engineering, SCARM provides support for all system requirements development activities from requirements gathering to requirements validation.

Further information:


Sutcliffe, A. 1998. Scenario-Based Requirement Analysis. Requirements Engineering Journal. Vol. 3. No 1. Pages 48−65.


Sutcliffe, A. & Ryan, M. 1998. Experience with SCRAM, a SCenario Requirements Analysis Method. Proceedings of the Third International Conference on Requirements Engineering.

Structured System Analysis and Design Methodology (SSADM)

Structured System Analysis and Design Methodology (SSADM) is a methodology used in the analysis and design stages of systems development. It consists of seven stages, which include steps to be followed. SSADM utilises a set of complementary, well-known techniques, such as data flow diagrams, entity life histories and relational data analysis. In requirements engineering SSADM can be used when performing detailed analysis of system and software requirements.

Further information:


Ashworth, C. & Goodland, M. 1990. SSADM: A Practical Approach. McGraw-Hill.


Originally developed for UK Government development projects by Learnmonth and Buchett Management Systems and the Government's computing advisory body, the CCTA. SSADM Version 1 was released in June 1981. Since then, new versions have been developed up to Version 4.


Select SSADM is a software tool supporting SSADM. URL: http://www.selectbs.com/products/products/ssadm.htm

Soft System Methodology (SSM)

Soft System Methodology (SSM) aims at establishing an understanding of the problem within an organisation by structuring the situation from the perspectives of all relevant systems (e.g. people, procedures, policies, hardware and software). This broad perspective makes the approach usable in requirements elicitation combined with other requirements analysis methods. (Kotonya & Sommerville, 1998)

Further information:


Checkland, P.B. 1981. Systems Thinking, Systems Practice. John Wiley & Sons.


Wilson, B. & Duffy, M. 2001. Soft Systems Methodology: Conceptual Model Building and Its Contribution. John Wiley & Sons.


Developed by Professor Peter B. Checkland at Lancaster University.


SSM homepage. URL: http://members.tripod.com/SSM_Delphi/ssm4.html


Dang, Q.C.; Baylis, T. & Patel, D. 1995. Modeling business through soft systems methodology in end-user development: A claim and an approach. Proceedings of the 7th International Conference on Advanced Information Systems Engineering.

Viewpoint-Oriented Requirements Definition (VORD)

Viewpoint-Oriented Requirements Definition (VORD) can be used for analysing a system from multiple perspectives in order to capture all possible requirements. First the relevant viewpoints are identified, and broken down into sub-viewpoints. Then each viewpoint's requirements are collected, rated for importance, documented, analysed and specified. VORD can also be utilised when documenting system and software requirements.

Further information:


Kotonya, G. & Sommerville, I. 1996. Requirements Engineering with Viewpoints. Software Engineering Journal. Vol. 11. No 1. Pages 5–11.


Developed by Gerald Kotonya and Ian Sommerville in 1992.


VORD Toolset written in Java from the method's developers. URL: http://www.comp.lancs.ac.uk/computing/resources/re/VORDTool.zip


Kotonya, G. 1999. Practical Experience with Viewpoint-Oriented Requirements Specification. Requirements Engineering Journal. Vol. 4. No. 3. Pages 115–133.

Viewpoint-Oriented System Engineering (VOSE)

In VOSE viewpoints are organised into configurations (collections of related viewpoints) in order to resolve conflicts among system requirements. Templates are used for describing the viewpoints. A configuration for a hypothetical problem domain may consist of templates with different styles (viewing the same partition of the problem domain) or templates with the same style (viewing different partitions of the problem domain). The final system is constructed after combining all of the configurations with problem solved. (Kotonya & Sommerville, 1998). In requirements engineering, VOSE is best suited for performing top-level analysis to raw requirements.

Further information:


Finkelstein, A.; Kramer, J.; Nusebeih, B. & Goedicke, M. 1992. Viewpoints: A framework for intgrating multiple perspectives in systems development. International Journal of Software Engineering and Knowledge Engineering. Vol. 2. Issue 10. Pages 31-58.


Developed at Imperial College, London in early 1990's.

3.3 Requirements allocation and flow-down

This section presents methods and techniques, which can be utilised during the requirements allocation and flow-down phases of the requirements engineering process, either alone or supporting other methods. A short overview of the method is provided followed by pointers to sources of further information. Methods supporting software architecture analysis can be found e.g. from a survey by Dobrica and Niemelä (2002).

The methods and techniques presented in this section are:

- Architecture Tradeoff Analysis Method (ATAM)
- Hatley–Pirbhai Methodology (HPM)
- System Requirements Allocation Methodology (SRA)

Architecture Tradeoff Method (ATAM)

Architecture Tradeoff Analysis Method (ATAM) is an iterative method for evaluating if a system's architecture-level design meets its requirements when considering multiple, desired quality attributes. The method identifies trade-off points between the quality attributes, facilitates communication between stakeholders, clarifies and refines requirements, and provides a framework for an ongoing, concurrent process of system design and analysis. (Kazman et al., 1998) In requirements engineering, ATAM is best suited for performing trade-off analysis and managing non-functional requirements during requirements allocation. It can be also used for recording a rationale of requirements flow-down.

Further information:


Kazman, R.; Klein, M.; Barbacci, M.; Longstaff, T.; Lipson, H. & Carriere, J. 1998. The Architecture Tradeoff Method. Proceedings of the fourth IEEE International Conference on Engineering of Complex Computer Systems. Pages 68−78.


Developed at Software Engineering Institute (SEI).


SEI's ATAM homepage. URL: http://www.sei.cmu.edu/ata/ata_init.html


Kazman, R.; Barbacci, M.; Klein, M.; Carriere, S.J. & Woods, S.G. 1999. Experience with performing Architecture Tradeoff Analysis. Proceedings of the International Conference on Software Engineering. Pages 54−63.

Hatley–Pirbhai Methodology (HPM)

The Hatley-Pirbhai Methodology (HPM) is a well-defined methodology for developing embedded, real-time systems. It handles both the system requirements and the architecture concurrently with a requirements model and an architecture model. HPM provides a means for capturing, recording, allocating and managing system requirements and architecture at all levels, and links the system with its subsystems. In requirements engineering, HPM is best suited for verifying requirements allocation and managing changes to it.

Further information:

- Hatley, D.J. & Pirbhai, I.A. 1987. Strategies for Real-Time System Specification. Dorset House.
- Hatley, D.; Hruscha, P. & Pirbhai, I.A. 2000. Process for System Architecture and Requirements Engineering. Dorset House.
- Developed by Derek Hatley and Imtiaz Pirbhai in 1987.
- TurboCASE/Sys is a software tool, which can be used to support different activities of HPM. URL: http://www.turbocase.com/
- Rader, J. & Haggerty, L. 1994. Supporting Systems Engineering with Methods and Tools: A Case Study. The 28th Asilomar Conference on Signals, Systems and Computers. Vol. 2. Pages 1330–1334.

System Requirements Allocation Methodology (SRA)

System Requirements Allocation Methodology (SRA) is a customer-oriented systems engineering approach for allocating top-level quantitative system requirements. It aims at creating optimised design alternatives, which correspond to the customer requirements using measurable parameters. (Hadel & Lakey, 1995) In requirements engineering, SRA provides support for different activities of requirements allocation.

Further information:

- Hadel, J.J. & Lakey, P.B. 1995. A customer-oriented approach for optimising reliability-allocation within a set of weapon-system requirements. Proceedings of the annual symposium on Reliability and Maintainability. Pages 96–101.
- Developed at McDonnell Douglas.

3.4 Software requirements analysis and specification

This section presents methods and techniques, which can be utilised during the software requirements analysis and specification phase of the requirements engineering process, either alone or supporting other methods. Some methods

and techniques cover several RE phases, and therefore the classification is rather suggestive than explicit. A short overview of the method is provided followed by pointers to sources of further information. At the end of this section also two methods suitable for component-based software production are presented.

The methods and techniques presented in this section are:

- Booch Methodology
- Hierarchical Object Oriented Requirements Analysis (HOORA)
- Jacobson Method
- Object Modeling Technique (OMT)
- Planguage
- Rational Unified Process (RUP)
- Shlaer-Mellor Object-Oriented Analysis Method
- Software Cost Reduction requirements method (SCR)
- Software Requirements Engineering Methodology (SREM)
- Storyboard Prototyping
- Structured Analysis and Design Technique (SADT)
- Structured Analysis and System Specification (SASS)
- Unified Modeling Language (UML)
- Volere method
- WinWin approach
- Component-based methods:
    - COTS-Aware Requirements Engineering (CARE)
    - Off-the-Shelf Option (OTSO)

Booch Methodology

The Booch software engineering methodology is an object-oriented approach for system analysis and design. It can be used for analysing and specifying software requirements, as it produces a high-level description of the system's functions and structure from the customer requirements' point of view. Also domain analysis, which is done by defining object classes, their attributes, inheritance,

and methods, can be utilised in the analysis and specification of software requirements. At the end of the analysis phase, validation is performed. (Booch, 1994) In requirements engineering, the Booch Methodology is best suited for performing high-level and detailed analyses to software requirements. It also provides some support for performing validation of software requirements.

Further information:

Booch, G. 1994. Object-oriented Analysis and Design with Applications, 2nd edition. Addison-Wesley.

Hierarchical Object Oriented Requirements Analysis (HOORA)

Hierarchical Object Oriented Requirement Analysis (HOORA) is a method that can be used to model systems in an object-oriented way, to trace the model elements to user requirements, to translate the model information into software requirements, and to generate an initial architectural design from the model information. (Gennaro, 1995) In requirements engineering, HOORA is best suited for analysing and validating software requirements. It also provides support for requirements allocation.

Further information:

Gennaro, G. 1995. Hierarchical Object Oriented Requirements Analysis (HOORA). Preparing for the Future Vol. 5. No. 4. European Space Agency. http://esapub.esrin.esa.it/pff/pffv5n4/genv5n4.htm

HOORA's homepage. URL: http://www.hoora.org/

HAT, the HOORA Analysis Tool, is a software tool, which can be used to support different activities of HOORA. URL: http://www.hoora.org/

Jacobson Method

Jacobson Method is a software process that defines an object as an entity characterised by number of operations and a state, which remembers the operations. The method uses two models to cover the requirements definition phase: a requirements model and an analysis model. The requirements model is intended for capturing user requirements by specifying all the functionality of

the system from users' perspective. The function of the analysis model is to form the basis for the system's structure by specifying objects in the information space. (Jacobson et al., 1992) In requirements engineering, Jacobson method provides support for analysing, validating and documenting software requirements.

Further information:

Jacobson, I.; Christerson, M.; Jonsson, P., & Gunnar, O. 1992. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley.

Object Modeling Technique (OMT)

Object Modeling Technique is an object-oriented development method, which uses three models to describe a software system: object model, dynamic model and functional model. The object model determines the types of objects that can exist in the system and identifies allowable relationships among objects. The dynamic model described valid changes to system states together with change conditions. The functional model describes the computations to be performed by the system. (Bourdeau & Cheng, 1995) In the field of requirements engineering OMT is best suited for analysing software requirements.

Further information:

Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F. & Lorensen, W. 1991. Object-Oriented Modeling and Design. Prentice Hall.

Derr, K.W. 1995. Applying OMT: A Practical Step-by-Step Guide to Using the Object Modeling Technique. SIGS Books.

Developed by Rumbaugh, Blaha, Premerlani, Eddy and Lorensen in 1991.

Object Modeling Technique tutorial in WWW.
URL: http://www.smartdraw.com/resources/centers/software/omt.htm

ObjectPlant is a software tool, which supports OMT.
URL: http://www.arctaedius.com/ObjectPlant/info.html

A case study of applying OMT to designing and implementing an electronic filing system is included in (Derr, 1995).

Planguage

Planguage consists of a systems engineering language for communicating systems engineering and management specifications, and a set of methods providing advice on best practices. The Planguage Specification Language is used to describe all the requirements, designs and plans. The main Planguage Methods are as follows:

- Requirement Specification: used to capture all the different requirement types.
- Impact Estimation: used to evaluate designs against the requirements.
- Specification Quality Control: used at any stage of a project to check the adherence of any plan, contract, bid or technical specification to best practice specification standards.
- Evolutionary Project Management: used to plan and monitor implementation of the selected designs. (Gilb, 2003)

In requirements engineering, Planguage is best suited for analysing software requirements.

Further information:


Gilb, T. 2003. Competitive Engineering. Addison-Wesley.


Developed by Tom Gilb.


Mr. Gilb's homepage http://www.result-planning.com/

Rational Unified Process (RUP)

RUP is an iterative process for engineering object oriented systems, and it strongly embraces use cases for modelling requirements and building the foundation for a system. RUP project consists of four phases and nine workflows, which are taking place in parallel. (Abrahamsson et al., 2002) The Requirements workflow covers requirements engineering activities, e.g. modelling requirements and continuously validating them with users. In requirements engineering, RUP is best suited for analysing and validating software requirements.

Further information:

- Kruchten, P. 1998. The Rational Unified Process. Addison-Wesley.
- Abrahamsson, P.; Salo, O.; Ronkainen, J. & Warsta, J. 2002. Agile Software development methods: Review and analysis. Pages 55–61. Technical Research Centre of Finland. VTT Publications.
- Developed by Philippe Kruchten, Ivar Jacobsen and others at Rational Corporation.

Shlaer-Mellor Object-Oriented Analysis Method

Shlaer-Mellor Object-Oriented Analysis Method provides a structured means of identifying objects within a system by analysing abstract data types. The identified objects are used as a basis for building three formal models of the system: information, state, and process. In RE the method can be used both for high level and more detailed analysis and specification of software requirements.

Further information:

- Shlaer, S. & Mellor, S. 1988. Object-Oriented System Analysis: Modeling the World in Data. Yourdon Press Computing Series. Prentice-Hall.
- Starr, L. 1996. How to build Shlaer-Mellor object models. Yourdon Press.
- Developed by S. Shlaer and S. Mellor in 1988.
- Project Technology Inc.'s WWW-library on Shlaer-Mellor Method. URL: http://www.projtech.com/pubs/papers.html
- WinA&D is a software tool, which can be used to build different Shlaer-Mellor models. URL: http://www.excelsoftware.com/
- Fayad, M.E.; Hawn, L.J.; Roberts, M.A. & Klatt, J.R. 1991. Mission Generation System (MGS): An Application of Shlaer-Mellor's Object-Oriented Method. The Proceedings of the IEEE/AIAA 10th Conference on Digital Avionics Systems. Pages 91–96.

Software Cost Reduction requirements method (SCR)

Software Cost Reduction requirements method (SCR) is a formal, partially mathematical method, which uses a tabular notation for representing system requirements. In RE SCR is most suitable for analysing, specifying and documenting the system and software requirements of safety-critical and control-intensive software systems.

Further information:


Parnas, D.L. 1978. Software Requirements for the A-7 Aircraft. Technical Report 3876. Naval Research Laboratory. Washington DC.


Developed by David Parnas in 1978 at Naval Research Laboratories to document flight program requirements.


Constance Heitmeyer's homepage including many publications on SCR.
URL: http://chacs.nrl.navy.mil/personnel/heitmeyer.html


Kirby, J. Jr.; Archer, M. & Heitmeyer, C. 1999. Applying Formal Methods to an Information Security Device: An Experience Report. Proceedings of the 4th IEEE International Conference on High Assurance Systems Engineering. Pages 81–88.

Software Requirements Engineering Methodology (SREM)

In Software Requirements Engineering Methodology (SREM) requirements are generated using a requirements specification language (RSL) and a software tool called REVS (Requirements Engineering and Validation System). The basic concept underlying SREM is that design-free functional software requirements should specify the required processing in terms of all possible responses (and the conditions for each type of response) to each input message across each interface. (Alford, 1978) In requirements engineering, SREM provides support for all the activities of software requirements development.

Further information:


Alford, M. 1977. A Requirements Engineering Methodology for Real Time Processing Requirements. IEEE Transaction on Software Engineering. Vol. 3. No.1. Pages 60–69.


Alford, M.W. 1978. Software Requirements Engineering Methodology (SREM) at the Age of Two. Proceedings of Computer Software and Applications. IEEE Computer Society's 2nd Edition. Pages 332–339.


Developed by Mack Alford in 1977.


Scheffer, P.A.; Stone, A.H. & Rzepka, W.E. 1985. A case study of SREM. IEEE Computer. Pages 47–54. Vol. 18. Issue 4.

Storyboard Prototyping

Storyboarding is a modelling technique, which combines practices from requirements analysis and simulation methodologies. A storyboard is a sequence of displays that represent the functions, which the system may perform when formally implemented. Storyboards are 'live' tools, which means, that after initial test runs it is usually necessary to make changes and conduct more tests until consensus emerges about the system's features. Storyboards are designed to verify requirements definitions, help designers to decide, if the system can be developed given the known time and financial constraints, and serve as compasses to software engineers. (Andriole, 1989) In requirements engineering, Storyboards can be used for high-level analysis of software requirements and for validating the requirements with users.

Further information:


Andriole, S. 1989. Storyboard Prototyping for Systems Design: A New Approach to User Requirements Analysis. Q E D Pub Co.


Andriole, S. 1992. Rapid Application Prototyping: The Storyboard Approach to User Requirements Analysis. John Wiley & Sons


Developed by Stephen J. Andriole in 1989.


Mr. Andriole's homepage. URL: http://www.andriole.com/


Vermont Views is a software tool, which can be used to support Storyboard prototyping. URL: http://www.vtsoft.com/

Structured Analysis and Design Technique (SADT)

SADT is a proprietary software engineering methodology for analysing and representing software requirements and design. SADT is composed of a graphic language and a method for using it. An SADT model is organised sequence of diagrams, each with supporting text. SADT also defines the personnel roles in a software project. (Schoman & Ross, 1977) In requirements engineering, SADT can be used for analysing and documenting software requirements.

Further information:


Schoman, K. & Ross, D.T. 1977. Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering. Pages 6–15.


Marca, D.A. & McGowan, C.L. 1988. SADT: Structured Analysis and Design Techniques. McGraw-Hill.

Structured Analysis and System Specification (SASS)

SASS is a top-down technique, in which the analyst starts by representing the system in a context diagram showing all system inputs and outputs and repeatedly refines the system, representing each refinement in a more detailed diagram. The purpose of the SASS is to analyse the problem, not to design the software. The SASS consists of five, consecutive phases: first, a survey is conducted, then it is followed by structured analysis, a hardware study, structured design and finally, the implementation. (Davis, 1990) In requirements engineering, SASS can be utilised performing detailed analysis to software requirements.

Unified Modeling Language (UML)

UML, the Unified Modeling Language, is a methodology-independent graphical notation for modelling software systems. It can be used to specify, visualise, and document models of software systems, including their structure and design, in a way that meets all of these requirements. UML defines and uses twelve types of diagrams divided into three categories: structural, behaviour and model management. The behavioural diagrams, including use case diagrams, are best suited for the purposes of requirements engineering. They can be used to

describe functional requirements' interactions with the system in support of their goal(s), which provides a basis for their organisation and makes it meaningful to users. (Booch et al., 1998)

Further information:


Booch, G.; Jacobson, I. & Rumbaugh, J. 1998. The Unified Modeling Language User Guide. Addison-Wesley.


Homepage of UML. URL: http://www.uml.org/

Volere

The Volere method provides a generic process for gathering requirements, ways to elicit them from users, as well as a process for verifying them. The Volere Specification Template provides specific instructions to achieving the appropriate requirements specifications. The template guides each part of the process in detail. The focus of the method is also in understanding customer needs in all phases of the process. (Robertson & Robertson, 1999) In requirements engineering, Volere is suitable for gathering requirements and validating both system and software requirements. It can be utilised also when writing the software requirements specification.

Further information:


Robertson, S. & Robertson, J. 1999. Mastering the Requirements Process. Addison-Wesley.


Developed by Suzanne and James Robertson at Atlantic Systems Guild, Inc.


Volere's homepage. URL: http://www.volere.co.uk/

WinWin Approach

WinWin Approach is a collaboration technique, which aims at constructing a problem/design solution, which meets different stakeholders' needs and expectations (i.e. win conditions) optimally. WinWin supports multi-stakeholder considerations and change management, and can be used for achieving effective

requirements negotiation and attaining increased customer satisfaction. (Bose, 1995) In requirements engineering,

Further information:


Bose, P. 1995. A Model for Decision Maintenance in the WinWin Collaboration Framework. Proceedings of the 10th Conference on Knowledge-Based Software Engineering. Pages 105–113.


The homepage of WinWin. URL: http://sunset.usc.edu/research/WINWIN/

Component based methods:

COTS-Aware Requirements Engineering (CARE)

CARE is a process-driven approach for selecting Commercial Off-The-Shelf (COTS) components, which satisfy the customer requirements. CARE starts by eliciting the customers' goals, which describe high-level objectives for the system. When requirements are elicited, they need to be evaluated against the goal of the system. Requirements for the system need to be developed by considering the capabilities of available COTS components. (Chung et al., 2001)

Further information:


Chung, L.; Cooper, K. & Huynh, D.T. 2001. COTS-Aware Requirements Engineering Technique. Proceedings of the 2001 Workshop on Embedded Software Technology (WEST'01).

Off-the-Shelf Option (OTSO)

OTSO method is developed to offer basis for evaluating and selecting reusable components for SW development. It provides specific techniques for defining the evaluation criteria, comparing the costs and benefits and consolidating the evaluation result for decision making. This multi-phase approach to Commercial Off-The-Shelf (COTS) component selection begins during requirements elicitation. First, all potential candidates for reuse with using existing guidelines and criteria are searched, then it is decided, which are the best candidates to

selected for more detailed evaluation. Then the candidates are evaluated by the evaluation criteria and the evaluation results are documented. Finally, the evaluation results are analysed and the best COTS components selected. (Kontio, 1995)

Further information:

Kontio, J. 1995. OTSO: A Systematic Process for Reusable Software Component Selection. The Hughes Information Technology Corporation and the EOS Program. Version 1.0

3.5 Continuous activities

This section presents methods and techniques, which can be utilised when performing the continuous activities (i.e. requirements documentation, requirements validation and verification, and requirements change management) of the requirements engineering process, either alone or supporting other methods. A short overview of the method is provided followed by pointers to sources of further information. At the end of this section, short descriptions of general approaches related the continuous activities of the requirements engineering process are presented.

The methods and techniques presented in this section are:

- B-method
- Petri Nets
- Vienna Development Model (VDM)
- Specification Language Z
- General approaches related to the continuous activities of the requirements engineering process:
    - Requirements modelling
    - Requirements reuse
    - Requirements reviews

B-method

B-method is a formal specification method and language for iterative development of software specifications as it covers all the software life-cycle activities. The B-specification construction can be done in an incremental or compositional manner. The consistency of the specifications can be formally verified by proving all the proof obligations, which are automatically generated. As a formal method B offers a high degree of confidence, that a system will conform to its specifications. In the field of RE the B-method is most suitable for formally specifying the requirements of safety or security critical systems.

Further information:


Schneider, S. 2001. The B-method: An Introduction. Palgrave.


Developed by J. -R. Abriail.


The B Formal Method WWW-Bibliography. URL: http://www3.inrets.fr/B@INRETS/B-Bibliography/


B-toolkit is a software tool, which supports the B-method. URL: http://www.b-core.com/


Sekerinski, E. & Sere, E. (eds.).1998. Program development and stepwise refinement – Case Studies Using the B method. Springer-Verlag.

Petri Nets

Petri Nets is a formal, mathematically defined technique for modelling and analysing software systems. A Petri Net itself is a directed graph consisting of defined elements, e.g. different types of nodes and transitions. In the area of requirements engineering Petri Nets can be used for systems graphical modelling, specification and prototyping, as the models produced within the technique are executable.

Further information:


Petri, C.A. 1962. Kommunikation mit Automaten. Bonn Institut für Instrumentelle Mathematik. Schriften des IIM Nr. 2.


Girault, C. & Valk, R. 2002. Petri Nets for System Engineering: A Guide to Modeling, Verification and Applications. Springer-Verlag.


Developed by Carl Adam Petri.


Petri Nets homepage. URL: http://www.daimi.au.dk/PetriNets/


GreatSPN is a software tool, which provides support for Petri Nets. URL: http://www.di.unito.it/~greatspn/


McLendon, W.W. & Vidale, R.F. 1992. Analysis of an Ada System Using Coloured Petri Nets and Occurrence Graphs. Proceedings of the 13th International Petri Net Conference. Lecture Notes in Computer Science Vol. 616. Springer Verlag.

Vienna Development Model (VDM)

Vienna Development Model (VDM) is a formal, model-based specification method, which has a lot of similarities with Specification Language Z. It provides a notation supported by a set of techniques for modelling, analysing and specifying software systems, and progressing further into detailed design and coding. In VDM specifications different definitions, such as state and function definitions, are collected in modules. In the field of RE VDM is most suitable for formally specifying the requirements of safety or security critical systems. The VDM Specification language achieved full ISO standardisation in 1996 (ISO/IEC 13817-1).

Further information:


Björner, D. & Jones, C.B. (ed.s). 1978. The Vienna Development Method: The Meta-Language. Volume 61 of Lecture Notes in Computer Science. Springer-Verlag.


Jones, C.B. 1990. Systematic Software Development Using VDM, 2nd ed. Prentice Hall.


Developed by Björner and Jones in 1978.


VDMTools is a software toolkit, which supports writing formal specifications using Vienna Development Model.
URL: http://www.ifad.dk/Products/vdmtools.htm

Specification Language Z

Specification language Z is a formal, model-based method for writing software specifications. The specifications written in Z contain both formal, mathematical parts and informal, textual descriptions. The purpose of this dualistic structure is to provide a precise description of the intended behaviour of the system (formal part), but still keep the specification approachable and understandable (informal part) (Sheppard, 1995). The specifications written in Z are usually simpler and shorter than specifications written in some programming language. In the field of RE Z is most suitable for formally specifying, documenting and validating the requirements of safety or security critical systems.

Further information:


Abrial, J-R. 1980. The Specification Language Z. Oxford University Programming Research Group.


Sheppard, D. 1995. An Introduction to Formal Specification with Z and VDM. McGraw-Hill.


Developed by the Programming Research Group (PRG) at Oxford University in 1980.


Z/EVES is a software tool, which provides support for writing specifications using Z. URL: http://www.ora.on.ca/z-eves/

General approaches related the continuous activities of the requirements engineering process:

Requirements modelling:

Requirements should be modelled at an appropriate level of detail when they are agreed. Requirement models have to be as clear and unambiguous as possible that all system stakeholders can understand requirements correctly. In order to make the requirements easy to understand they must be modelled using natural language and diagrams. (Kotonya & Sommerville, 1998.) Examples of modelling techniques include data flow models, and Unified Modeling Language (UML), see section 3.4.

Requirements reuse:

Reusing requirements means that requirements of other systems in the same application area are reused. When developing new systems it is generally good practice to reuse as much knowledge as possible. By reusing requirements the less effort and time is spend in eliciting requirements and risks of producing requirements are reduced. Reuse of requirements across systems may be possible in situations where the requirement is concerned with providing information about the application domain and/or it is concerned with the style of presentation of information. (Sommerville & Sawyer, 1997; Kotonya & Sommerville, 1998)

Requirements reviews:

Requirements reviews are the most widely used technique of requirements validation. They involve a group of people who read and analyse the requirements, look for problems, meet to discuss these problems and agree on a set of actions to address the identified problems. Examples of review methods include, N-fold Requirements Inspection Method, Walkthroughs, and inspections. Important aspects of requirements reviews include pre-review checking, review team membership and review checklists (Kotonya & Sommerville, 1998).

Further information:

Ciolkowski, M.; Laitenberger, O.; Rombach, D.; Shull, F.; Perry, D. 2002. Software inspections, reviews and walkthroughs. Proceedings of the 24th International Conference on Software Engineering. Pages 641–642.

Gilb, T. & Graham, D. 1993. Software Inspection, 1st ed. Addison-Wesley.

Table 5 summarises all presented requirements engineering methods (except general methods presented in section 3.1 and in the end of section 3.5) and their suitability to different phases and activities of the RE process.

Table 5. Methods and their RE phase coverage.

	PHASE / ACTIVITY												
	System Requirements Development					Allocation & Flowdown		SW requirements development					Publication year	
METHOD	Gathering	Top level analysis	Detailed analysis	Documentation	Validation	Allocation	Flow-down	High level analysis	Detailed analysis	Documentation	Validation	Change mgmt	Latest publication	Latest experiences
ATAM						X	X						2002	2002
B-method				X	X					X	X		2001	1999
Booch						X		X	X		X		1995	1997
CARE													2002	--
CORE	X									X			1979	1992
Ethnography	X												1999	2001
Hatley-Pirbhai				X		X							2000	1994
HOORA	SW					X		X	X	X			2003	1997
Jacobson	SW					X		X	X	X	X		1992	--
JAD	X												1995	1998
MPARN		X											2002	--
OMT								X	X				1999	1995
OTSO													1996	1996
Petri Nets					X						X		2002	2002
Planguage								X	X	X			2003	--
Protocol an.	X												1993	--
QFD	X	X	X									X	2002	2002
REVEAL	X	X										X	2001	--
RUP	SW							X	X	X		X	2003	--
SADT									X	X			1993	2000
SASS			X						X				1979	--
SCR			X	X					X	X			2000	2000
SCRAM	X	X	X	X	X					X		X	1998	1998
Shlaer-Mellor								X	X				1998	1998
SRA		X				X	X						1995	--
SREM	SW							X	X	X	X		1990	1985
SSADM			X			X			X				2002	--
SSM	X												2001	1997
Storyboarding	SW							X			X		1992	1992
UML	SW							X	X				2002	2002
VDM				X	X					X	X		1997	2000
WinWin		X											2000	2001
Volere	X				X						X		2001	--
VORD	X	X	X	X						X			2000	1999
VOSE		X											1994	--
Z										X	X		2002	2000

3.6 Requirements management

This chapter introduces requirements management (RM) related techniques, models and methods. Each main RM activity is considered separately and related techniques, models and methods are presented briefly according to studied literature. Some techniques or approaches cover several RM activities, and therefore the classification is rather suggestive than explicit. For example, RADIX method, presented as a method supporting requirements traceability, provides support for both requirements traceability and identification.

3.6.1 Requirements identification

Requirements identification practices focus on the assignment of unique identifier for each requirement (Sommerville & Sawyer, 1997). Natural identification scheme for hierarchical requirements is to identify child-requirements following parent requirement's identification (Leffingwell & Widrig, 2000). This identification technique describes the item's place in the system or document. For example, when organising requirements into chapters, *4.2.6* would identify sixth requirement in second section in the chapter four (Sommerville & Sawyer, 1997). The technique can describe item's location in requirements structure by using its identification. However, unique identifier is difficult to assign for requirement until all requirements and their structures are clear. This complicates the control and identification of requirements during RE process until the requirements have been frozen. The technique also suggests that certain requirement is closely related to other requirements with similar identifiers even there might be also other relations (Sommerville & Sawyer, 1997). Kotonya and Sommerville (1998) describe the following alternative approaches, which can address these problems:

- **Dynamic renumbering:** This technique allows the automatic inter-document renumbering of paragraphs when inserting information using e.g. word-processing systems. It usually also allows cross-references to the paragraphs and maintains automatically information when there are changes in requirements structure.

- **Database record identification:** Database record identifier is assigned for each requirement entered in a requirements database. By using this technique each requirement is considered to be an entity, which can be uniquely identified, versioned, referenced and managed.

- **Symbolic identification:** Requirements are identified using symbolic names related to the contents of the requirement. For example, EFF-1, EFF-2, …, EFF-n can be used for the identification of efficiency requirements. The technique does not fix the requirement under certain structure. It can be used as interim identification scheme until requirements have been defined.

3.6.2 Requirements traceability

Requirements traceability (RT) refers to the ability to describe and follow the life of a requirement and its relations with other development artefacts in both a forwards and backwards direction (Gotel, 1995).

Sommerville and Sawyer (1997) divide basic techniques for traceability into traceability tables, traceability lists and automated traceability links. Tables and lists (also referred as traceability matrixes) can be used to describe also the relations between requirements and other software development artefacts. For example, Adams and Douthit (2000) use a traceability matrix for presenting requirements and their related allocation, implementation and verification information. On the other hand, Remillard (1996) uses a traceability matrix for identifying relationships between requirements and tests. According Pinheiro (2000) some form of traceability matrix is implemented in almost every current traceability tool. Matrices are also utilised for traceability, for example, as a part of methods like QFD (Quality Function Deployment) (Gotel, 1995; West, 1991).

Traceability tables are used to present relationships between requirements describing how requirement on row depends on requirement on column (Sommerville & Sawyer, 1997). Furthermore, there can be tables to describe all needed relationships, e.g. requirement-source –table and requirement-design –table. The mark used for describing relationship can be simply e.g. an asterisk. More complex presentation of relationship is possible if relationship's type can be expressed by using specific symbols for different types of relationships according project's RM practices (for example, *specifies*, *requires*, *constrains*,

etc.) (Sommerville & Sawyer, 1997). Traceability tables are simple and easy to implement e.g. by using spreadsheet identification (Leffingwell & Widrig, 2000). They are useful when there are relatively small number of requirements. Richer information can also be attached to the matrix by using relationship symbols. On the other hand, they are not very useful when there are hundreds or thousands of requirements (thinly populated matrix). Also meaning of rows and columns can be misunderstood (Sommerville & Sawyer, 1997). Traceability list is a technique, which can be used for compact presentation of requirements (Sommerville & Sawyer, 1997). In this presentation there is list of depending requirements for each requirement in rows. This representation is more compact than table and can also be implemented by using e.g. spreadsheet. Technique can be used for greater number of requirements (do not become unmanageable so easily) (Sommerville & Sawyer, 1997). It is also less error prone than table presentation. However, separate lists are needed for each type of relation. In addition, there is no easy way to assess the inverse of a relationship.

Automated traceability links –technique uses a database to store and maintain requirements. Traceability links are included as fields in the database record (Sommerville & Sawyer, 1997). This is efficient way to manage requirements as separate entities and describe and maintain links between them.

3.6.3 Requirements traceability models, methods and languages

RT comes sometimes as the by-product of certain approach. The following approaches are collected from Gotel (1995) and Pinheiro (2000) and divided into models, methods and languages as follows:

- Models:
  - REMAP and IBIS based solutions
  - RT reference models
  - Contribution structures
  - Document-centred models
  - Database-guided models
- Methods:
  - RADIX
  - QFD
- Languages

Models

Term ”Model” is here used to refer to the establishment of structures containing the elements and the relations used in tracing, usually specifying their types as well as constraints under which the elements of the model can be related (Pinheiro, 2000).

REMAP and IBIS based solutions

According Pinheiro (2000) IBIS model (Issue Based Information Systems method) related models intend to capture design rationale by providing automated support for discussion and negotiation of design issues. IBIS model can be understood as a decision support system (Lee & Liu, 1993). Traceability related models, which are originally based on IBIS are, for example, “REMAP model” and its further development to “RT reference models”. Ramesh and Dhar (1992) enumerate also other systems, which aim to support co-operative work and argumentation: gIBIS, IBE and SIBYL.

REMAP describes conceptual model that relates process knowledge to the objects that are created during the requirements engineering process (Ramesh & Dhar, 1992). This process knowledge is actually reasons behind design decisions, or design decisions that shape the design (Ramesh & Dhar, 1992). The main input and output elements of the model are *Requirements* and *Design objects*. REMAP allows collecting rationale information related to requirement’s evolution and requirement’s refinement into designs. Ramesh and Dhar (1992) have developed also a prototype environment based on the model.

Further information:

- Lee, S., Liu, B. 1993. IBO: an issue based object model for software design. Proceedings of IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering. Vol. 1. Pages 270–274.
- Ramesh, B., Dhar, V. 1992. Supporting systems development by capturing deliberations during requirements engineering. IEEE Transactions on Software Engineering. Vol. 18. No. 6. Pages 498–510.

Ramesh, B., Jarke, M. 2001. Toward reference models for requirements traceability. IEEE Transactions on Software Engineering. Vol. 27. Issue 1. Pages 58–93.

REMAP model introduced in Ramesh and Dhar (1992).

According Ramesh and Jarke (2001) REMAP has been served as a starting point for several commercial tool developments, for example, Knowledge Based Software Assistant (KBSA) environment.

RT reference models

The development of RT reference models is based on REMAP model (Ramesh & Jarke, 2001). The purpose of reference models is to significantly reduce the task of creating application-specific models and systems for traceability: the user selects relevant parts of the reference model, adapts them to the problem at hand, and configures an overall solution from these adapted parts (Ramesh & Jarke, 2001). Different stakeholders involve to the development of the product. For example, project managers, designers, maintainers, end users, etc. The traceability needs of these stakeholders differ due to differences in their goals and priorities. According Ramesh and Jarke (2001) the users can be divided into two levels: low-end users and high-end users with different requirements for RT. The reference models describe preliminary models for both of these user types.

Further information:

Ramesh, B., Jarke, M. 2001. Toward reference models for requirements traceability. IEEE Transactions on Software Engineering. Vol. 27. Issue 1. Pages 58–93.

RT reference models introduced in Ramesh and Jarke (2001).

High-end models have been demonstrated using SLATE RM tool (demonstration is documented by Ramesh and Jarke (2001)). Design rationale capture has been demonstrated using KBSA environment (demonstration is documented by Ramesh and Jarke (2001)).

Contribution structures

Contribution structures make traceable the human sources of requirements, requirements related information, and requirements related work (Pinheiro, 2000).

Gotel and Finkelstein (1995) have considered support for pre-traceability since it is not well known and supported area. They proposed approach based on contribution structures to describe and manage traceability links (Gotel & Finkelstein, 1994b; Gotel & Finkelstein, 1995). They emphasise that requirements' sources are in important role when considering the problem of traceability. The links between sources and requirements artefacts are usually uncontrolled. This lack of traceability can cause problems on later life cycle phases when details of who originally generated the requirement are lost. Contribution structures offer a way to model the network of people who have participated in the requirements engineering process (Gotel & Finkelstein, 1997). They provide the opportunity to extend conventional forms of artefact-based requirements traceability with the traceability of contributing persons (Gotel & Finkelstein, 1997). Also prototype tool has been constructed to demonstrate, refine and evaluate the model (Gotel, 1995).

Further information:


Gotel, O. & Finkelstein, A. 1994b. Modelling the Contribution Structure Underlying Requirements. Proceedings of the First International Workshop on Requirements Engineering: Foundations of Software Quality.


Gotel, O. & Finkelstein, A. 1995. Contribution structures [Requirements artifacts]. Proceedings of the Second IEEE International Symposium on Requirements Engineering. Pages 100–107


Gotel, O. & Finkelstein, A. 1997. Extended requirements traceability: results of an industrial case study. Proceedings of the Third IEEE International Symposium on Requirements Engineering. Pages 169–178.


Contribution structures for RT introduced by Gotel and Finkelstein (Gotel & Finkelstein, 1994b; Gotel & Finkelstein, 1995). Detailed model is presented in PhD thesis (Gotel, 1995).


Experiences from industrial case introduced in Gotel and Finkelstein (1997)

Document-centred models

These models usually present traces as relations between documents of different types (Pinheiro, 2000). For example, in SODOS (Software Documentation Support model) model RT results, from model's documentation strategy. On the other hand, HYDRA (Hypertext Model for Requirements Engineering) model (Pohl & Haumer, 1995) is the specialisation of generic hypertext model (GHYM). HYDRA is used to structure informal information during the requirements engineering process by creating formal hypertext object, which refer to the informal representations (or part of it) (Pohl & Haumer, 1995). These formal objects are used to relate informal information with other representations (e.g. entity relationship diagrams, first order logic constraints). Formal structure enables situated and selective retrieval of informal information (Pohl & Haumer, 1995).

Further information:


Horowitz, E. & Williamson, R. 1986. SODOS : A Software Documentation Support Environment - its Use. IEEE Transactions on Software Engineering. Vol. 12. No. 1. Pages 1076–1087.


Pohl, K. & Haumer, P. 1995. HYDRA: A Hypertext Model for Structuring Informal Requirements Representations. Second International Workshop on Requirements Engineering: Foundations of Software Quality.


SODOS introduced in Horowitz and Williamson (1986). HYDRA introduced in Pohl and Haumer (1995).

Database-guided models

Database guided models are used to store trace information on databases for later retrieval (Pinheiro, 2000). Usually these models describe explicit information related to the traceability. They could also be used as concrete alternatives when implementing RT. Examples from database guided models are the simple model developed by Sommerville and Sawyer (1997) which allows item identification and traceability, and more advanced model called Multiview++ (Toranzo & Castro, 1999). Multiview++ tries to provide comprehensive and explicit model of which information should be captured

containing item and attribute information. The model contains several perspectives, which identify and differentiate elements used by different stakeholders.

Further information:

Toranzo M. & Castro J. 1999. Multiview++ Environment: RequirementsTraceability from the perspective of different stakeholders. II IberoAmerican Workshop on Requirements Engineering.

Methods

Term ”method” is used here to refer to orderly arrangements of procedures to be followed in the development process, accompanied by guidelines for their execution (Gotel, 1995). These methods include also those requirements engineering or software engineering methods, which embed support for requirements traceability. For example, Kotonya and Sommerville (1998) introduce traceability practices as part of VORD method (traceability between viewpoints and requirements). On the other hand, agile development method, extreme programming (XP), implements traceability using cross-references between user stories (user requirements) and task cards. Next two methods, RADIX and QFD, are discussed more detailed.

RADIX

RADIX is methodology and tool for requirements traceability (Yu, 1994). It has been developed to support specific SW (5ESS switch SW) (Yu, 1994)). It aims to support product development by providing the identification and management of product features. Methodology describes also concept of the tool, based on documentation macros, called RADIX. The methodology is originally planned to support the ”waterfall” type of the development methodology where the exit criteria at each phase must be satisfied before the next phase is entered.

The practical implementation of RADIX is based on the set of documentation macros (Yu, 1994). RADIX uses cross-references to describe relations between artefacts (Pinheiro, 2000). The macros provide a means of identifying, labelling, and adding supportive information to requirements. For example, *.gD doc-12345R* identifies that document’s ID is doc-12345R. The methodology

describes traceability procedure to follow during the product development to ensure requirements traceability.

Further information:


Yu, W. D. 1994. Verifying software requirements: a requirement tracing methodology and its software tool-RADIX. IEEE Journal on Selected Areas in Communications. Vol. 12. No. 2. Pages 234–240.


RADIX introduced in Yu (1994).

QFD

Quality Function Development (QFD) is method used for RE. It has been introduced already in section 3.2, thus this section focuses on its traceability facilities. QFD utilises matrixes to implement the traceability of requirements through product development process (Brown, 1991; West, 1991). For example, to develop design requirements from customer requirements, product functions from design requirements, test requirements and process requirements from customer and design requirements (West, 1991). These matrixes are linked together by linking output from one matrix to input for another enabling requirements traceability through design, implementation and test (Brown, 1991; West, 1991).

Further information:


Brown, P. 1991. QFD: Echoing the Voice of the Customer. AT&T Technical Journal. March/April 1991. Pages 21–31.


West, M. 1991. Quality function deployment in software development. IEE Colloquium on Tools and Techniques for Maintaining Traceability During Design. Pages 5/1–5/7.

Languages

According Gotel (1995) and Pinheiro (2000) traceability can also be supported using languages or notations. For example, ALBERT (Dubois et al., 1994) is agent-oriented language for building and eliciting requirements for real-time systems. On the other hand, RML (Requirements Modelling Language,

Greenspan et al., 1994) provides traceability as a consequence of its structuring and organising principles. Regular expressions are used as part of the tracing model implemented by TOOR tool (tool for traceability) to support requirements traceability (Pinheiro, 2000).

Further information:

- Greenspan, S.; Mylopoulos, J. & Borgida, A. 1994. On formal requirements modeling languages: RML revisited. Proceedings of the 16th International Conference on Software Engineering. Pages 135–147.
- Dubois, E.; Du Bois, P. & Petit, M. 1994. ALBERT: an agent-oriented language for building and eliciting requirements for real-time systems. Vol. IV: Information Systems: Collaboration Technology Organizational Systems and Technology. Proceedings of the 27th Hawaii International Conference on System Sciences. Pages 713–722.
- Pinheiro, F. 2000. Formal and Informal Aspects of Requirements Tracing. III Workshop on Requirements Engineering.

3.6.4 Requirements change control

Requirements change management refers to the ability to manage changes to the systems requirements (Kotonya & Sommerville, 1998). Change management (ChM) process models and approaches are discussed in this section. Term ”process model” is here used to refer support for actual workflows and activities taking place when people work in the context of change management. These process models are specifically designed for general change management not especially for requirements changes. The following process models are collected and presented by Mäkäräinen (2000):

- Olsen’s ChM model
- V-like model
- Ince's ChM model
- AMES process model
- Spiral-like ChM model
- Generic ChM model

Olsen's ChM model

Model created by Olsen (1993) is based on general knowledge of software engineering principles. Olsen also uses mathematical and statistical data to explain the software "rush-hour". "Rush-hour" refers to the overload of software engineers.

Model views the whole software development process as a dynamically overloaded queue of changes and views all work done by software designer as changes. Change is anything that requires some work to be done. The source for the change can be change manager, user or verification activity. Sponsors offer the funding for the projects and they also monitor the schedules.

Further information:


Olsen, N. 1993. The software rush hour. IEEE Software. Vol. 10. Issue 5. Pages 29–37.


Model presented in Olsen (1993).

V-like model

The model aims at investigating the concept of integrated environments for software maintenance. This change management model describes the activities needed when implementing a change. According to Harjani and Queille (1992), following types of maintenance activities are considered:

- user support (answers to requests coming from users)
- corrective maintenance (activities to fix an error)
- evolutive maintenance (new functionalities)
- adaptive maintenance (adapt software to a change in its operational environment)
- perfective maintenance (improving non-functional requirements)
- preventive maintenance (improves the maintainability)
- anticipative maintenance (anticipate future problems).

Further information:


Harjani, D. -R., Queille, J. -P. 1992. A process model for the maintenance of large space systems software. Proceedings of Conference on Software Maintenance. IEE Computer Press. Pages 127−136.


The research of the model has been carried out in the ESF/EPSOM project. "EPSOM (European Platform for SOftware Maintenance) is subproject of the Eureka Software Factory project (ESF) (Harjani & Queille, 1992).

Ince's ChM model

According to Mäkäräinen (2000), Ince discusses how software configuration management relates to software change management. Ince's model covers activities related both software development and maintenance phases. Change request can be originated from customer requests for new features or error corrections. Other source for change request can be the development, which identifies problems in validation phase. Ince does not discuss about other external change sources like changes in the hardware environment or improvement ideas coming from software development team.

Further information:


Ince, D. 1994. Introduction to software quality assurance and its implementation. McGraw-Hill.


Original source of the model is Ince (1994).

AMES process model

Mäkäräinen (2000) states that AMES model supports maintenance process of a company. It defines three layers: strategic, management and technical (Mäkäräinen, 2000).

Strategic layer makes decisions on planning the future of the product and how the customer or user relationship will be taken care of. Management layer plans, organises and controls the actions and people who provide the change service. Technical layer carries out the changes.

Further information:


Boldyreff, C.; Burd, E. & Hather, R. 1994. An evaluation of the state of the art for application management. Proceedings of the International Conference on Software Maintenance. Pages 161−169.


AMES project.

Spiral-like ChM model

Compared with other model, Ince's model describes the outer cycle of the spiral model and the last quarter of the systems engineering cycle (technical implementation of the change). V-model defines the outermost cycle of the spiral model. Every round of spiral has been divided into four quarters: problem understanding, evaluate alternative solutions as well identify and solve risks, develop, and plan for the next phases.

According to Mäkäräinen (1996), the execution of the process starts from the innermost circle. Process proceeds clockwise. Mäkäräinen (2000) states that "the same tasks are performed by each cycle, but the viewpoint is different in each cycle.

Further information:


Mäkäräinen M. 2000. Software change management processes in the development of embedded software. Technical Research Centre of Finland. VTT Publications 416.


According to Mäkäräinen (2000) the spiral-like model was derived in the AMES project.

Generic ChM model

Mäkäräinen (2000) presents phases for generic change management and divides changes into the product level changes and project level changes. Product level changes generate new requirements for the new products or new product releases. It also generates feature modification, addition and deletion requests for development projects. Project level changes are initiated and managed internally

by the project and can be generated because of product level changes. Project level changes can also generate product level changes.

Mäkäräinen (2000) distinguishes four instances of the generic model for different types of change: trivial defect correction, defect correction, requirement level modification and improvement proposals.

Further information:


Mäkäräinen M. 2000. Software change management processes in the development of embedded software. Technical Research Centre of Finland. VTT Publications 416.


Introduced in Mäkäräinen (2000).


Case study related to the generic ChM process model presented in Mäkäräinen (2000).

Some approaches to support requirements ChM

Requirements change management support by using certain approaches. For example, Crnkovic at al. (1999) introduce how SCM functions and tool features can support RM. The intention of the approach is to utilise SCM to support requirements change management during product's life cycle. The approach handles each requirement as separate configuration item and connects them to implementation items (e.g. code modules) and utilises change management facilities to control changes to the requirements. This enables the change management of individual requirements during their life cycle. Also Lindsay et al. (1997) have emphasised fine-grained configuration management support. Lam et al. (1999) introduce the usage of metrics to support requirement change management. They emphasise measurement as central activity to underpinning sound and rational management decisions. They present measurement-action framework, which provides a set of indicators that can assist managers in measuring requirements change as well as generic action plans that provide managers with practical guidance on how to use the indicators as a basis for taking managerial action.

Further information:

- Crnkovic, I.; Funk, P.& Larsson, M. 1999. Processing requirements by software configuration management. Proceedings of the 25th EUROMICRO Conference. Vol. 2. Pages 260–265.
- Lindsay, P.; Yaowei, L. & Traynor, O. 1997. A generic model for fine grained configuration management including version control and traceability. Proceeding of the Australian Software Engineering Conference. Pages 27–36.
- Lam, W.; Loomes, M. & Shankararaman, V. 1999. Managing requirements change using metrics and action planning. Proceedings of the Third European Conference on Software Maintenance and Reengineering. Pages 122–128.

4. Requirements engineering tools

4.1 Introduction

Kotonya and Sommerville (1998) state that tools, which support requirements engineering, can be divided into following two types:

- *Modelling and validation tools* support the development of system models. The completeness and consistency of the models can be checked.
- *Management tools* help to manage the requirements database and support the requirements change management.

According to Kotonya and Sommerville (1998) modelling tools can be based on structured methods (e.g. SADT) or specialised requirements modelling languages (e.g. RSL). These tools enable the creation of graphical and textual models of the requirements and consistency checks. Validation tools, according to Kotonya and Sommerville (1998), can "analyse the description for mathematical inconsistencies which imply either mistakes in the formal specification or requirements errors".

On the other hand, the main functions of requirements management are according to Kotonya and Sommerville (1998) management of changes, relationships and the dependencies between the requirement documents and other documents during the whole product life cycle. RM management tools have been developed because of the problems of managing unstable requirements and the large amount of data collected during RE process. Management tools support the management of requirement database and changes to these requirements.

Organisation selects appropriate tools for use. Tool selection can depend of various factors, such as (Sommerville & Sawyer, 1997):

- The tools that are already in use in the organisation, and new tools that are compatible with the existing systems.
- Tool and training budget.
- The size of the system being developed.
- The specific guidelines which you wish to implement.
- Stability of the tool providing companies (that also means the availability of the tools).

4.2 Basic RM tool features

In this sub section the basic requirements management (RM) tool features are considered according literature. Detailed list of characteristics of requirements management tools can be found from Sommerville and Sawyer (1997) and from publication of Lang and Duggan (2001). According to Lang & Duggan (2001), software requirements management tool must be able to:

- Maintain unique identifiable description of all requirements.
- Classify requirements into logical user-defined groups.
- Specify requirements with textual, graphical, and model based description.
- Define traceable associations between requirements.
- Verify the assignments of user requirements to technical design specifications.
- Maintain an audit trail of changes, archive baseline versions, and engage a mechanism to authenticate and approve change requests.
- Support secure, concurrent co-operative work between members of a multidisciplinary development team.
- Support standard systems modelling techniques and notations.
- Maintain a comprehensive data dictionary of all project components and requirements in a shared repository.
- Generate predefined and ad hoc reports.
- Generate documents that comply with standard industrial templates.
- Connect seamlessly with other tools and systems.

RM tools collect together the system requirements in a database or repository and provide a range of facilities to access the information about the requirements (Kotonya & Sommerville, 1998). Firstly, information browsing, querying and reporting facilities allow the retrieval of needed information from database. Secondly, traceability support facilities are used to generate traceability information. Thirdly, converters and integrations with word-processors are needed for maintaining links between the database and the natural language representation of the requirements. Finally, change control facilities maintain information about requested requirements changes and links to the requirements affected by the change.

Many RM tools are based on database. Requirement database may have relatively few records but each of them may include many links - to documents, to text files and to other requirements. Commonly used database types are relational database systems and object-oriented database systems.

Relational databases are used for storing and managing large number of records, which have the same structure and minimal links between them. Currently many RM tools are based on relational databases. According to Sommerville and Sawyer (1997) it is possible to maintain many links with a relational database, but it is insufficient as it requires operations on several different tables. Sommerville and Sawyer (1997) state that object-oriented databases are structurally more suited to requirements management. They allow different types of information to be maintained in different objects and the way they manage links between objects is fairly straightforward.

4.3 Examples of RE tools

This section presents some tools for Requirement Engineering (RE). In the chapter 3 examples of the method-specific tools are included into the descriptions of the methods (if any tools have been found in the literature). There are numerous tools, which contain functionality related to RE support. Different tools have focused on different aspects of RE depending on their background. Thus, the intention is to provide the reader with a high level insight into the variety of RE related tools. It should be kept in mind, that the following presentation of RE tools is by no means comprehensive.

Collections of tool information can be found from these web sites (available 11.11.2003):

- Rabi Archrafi's descriptions of available requirements engineering tools. URL: http://www.volere.co.uk/tools.htm

- INCOSE's pages, which contain taxonomy of requirements engineering tools divided into hierarchical categories according to the purpose of use. URL: http://www.incose.org/tools/tooltax/reqengr_tools.html

- Ian Alexander's pages, which present a list of RE tool vendors and freeware suppliers. URL: http://easyweb.easynet.co.uk/~iany/other/vendors.htm

5. Summary

This publication is based on a literature study on systems and software requirements engineering. It discusses a diverse and plentiful set of available requirements engineering technologies and aims at providing a comprehensive view of the current state in requirements engineering practices.

Based on several requirements engineering processes described in literature, an iterative requirements engineering process suitable for the requirements engineering of embedded systems and software is defined. The phases of the process are described in detail with their respective purpose, input, output, and activities to be performed.

A survey of over 60 existing methods, techniques and approaches supporting the different phases and activities of requirements engineering is presented using the process model as a suggestive categorisation framework. The methods, techniques and approaches are introduced briefly together with pointers to sources of further information.

Also main types and basic features of tools and technologies, which provide support for different activities of requirements engineering are discussed. In addition, a short summary of existing RE tools is presented in order to offer an insight to the variety of RE tools.

Shortcomings analysis of the existing technologies presented in this publication will be performed later during the MOOSE project. Some limitations of the technologies were already discovered when performing this literature study. For example, experiences of the methods usage are not that widely published, yet in order to utilise a technology, information about its limitations and suitability to different situations is clearly needed.

Acknowledgements

We would like to thank all MOOSE project partners, especially people at LogicaCMG and Océ for their valuable comments regarding this publication. In addition, we would like to express our appreciation to the official reviewer of this publication, Dr. Serguei Roubtsov. We would also like to thank the support from ITEA and Tekes.

References

Abrahamsson, P.; Salo, O.; Ronkainen, J. & Warsta, J. 2002. Agile Software development methods: Review and analysis. Pages 55–61. Technical Research Centre of Finland. VTT Publications 478.

Adams, G. & Douthit, D. 2000. Managing concurrent development-a systems engineering approach. AUTOTESTCON Proceedings. Pages 248–255.

Alford, M.W. 1978. Software Requirements Engineering Methodology (SREM) at the Age of Two. Proceedings of Computer Software and Applications. IEEE Computer Society's 2nd Edition. Pages 332–339.

Ambler S. W. 1998. Process Patterns Building Large-Scale Systems Using Object Technology. Cambridge University Press/SIGS Books.

Andriole, S. 1989. Storyboard Prototyping for Systems Design: A New Approach to User Requirements Analysis. Q E D Pub Co.

Beyer, H. & Holtzblatt, K. 1998. Contextual Design: Defining Customer-Centered Systems. Morgan Kaufmann Publishers, Inc.

Blanchard B.S. & W. J. Fabrycky. 1981. Systems Engineering and Analysis. Prentice Hall.

Booch, G. 1994. Object-Oriented Analysis and Design with Applications, 2nd ed. Addison-Wesley.

Booch, G.; Jacobson, I. & Rumbaugh, J. 1998. The Unified Modeling Language User Guide. Addison-Wesley.

Bose, P. 1995. A Model for Decision Maintenance in the WinWin Collaboration Framework. Proceedings of the 10th Conference on Knowledge-Based Software Engineering. Pages 105–113.

Brown, P. 1991. QFD: Echoing the Voice of the Customer. AT&T Technical Journal. March/April, 1991. Pages 21–31.

Bourdeau, R.H. & Cheng, B.H.C. 1995. A Formal semantics for object model diagrams. IEEE Transactions on Software Engineering. Vol. 21. No. 10. Pages 799–821.

Buede, D.M. 1997. Integrating requirements development and decision analysis. IEEE International Conference on Systems, Man and Cybernetics. Computational Cybernetics Simulations. Vol. 2. Pages 1574–1579.

Chung, L.; Cooper, K. & Huynh, D.T. 2001. COTS-Aware Requirements Engineering Techniques. Proceedings of the 2001 Workshop on Embedded Software Technology (WEST'01).

CMMI-SE/SW V1.1. CMMI^{SM} for Systems Engineering/Software Engineering, Version 1.1.

Crnkovic, I.; Funk, P. & Larsson, M. 1999. Processing requirements by software configuration management. Proceedings of the EUROMICRO'99 Conference. Volume: 2. Pages 260–265.

Davis, A. M. 1990. Software requirements: analysis and specification. Prentice Hall.

Dayton, T. & Tournat, K. 1999. How to Design, Prototype, and Test A Usable GUI In Three Days. URL: http://www.baychi.org/meetings/archive/0399.html

Derr, K.W. 1995. Applying OMT: A Practical Step-by-Step Guide to Using the Object Modeling Technique. SIGS Books.

Dobrica, L. & Niemelä, E. 2002. A Survey of Software Architecture Analysis Methods. IEEE Transactions on Software Engineering. Vol. 28. No. 7. Pages 638–653.

Dorfman M. 1990. System and Software Requirements Engineering, In: Thayer, R.H. & Dorfman, M. IEEE System and Software Requirements Engineering. IEEE Software Computer Society Press Tutorial. IEEE Software Society Press.

Douglass, P. B. 1999. Doing Hard Time: Developing Real-Time Systems with UML - Objects, Frameworks, and Patterns. Addison-Wesley.

Dubois, E.; Du Bois, P. & Petit, M. 1994. ALBERT: an agent-oriented language for building and eliciting requirements for real-time systems. Vol. IV: Information Systems: Collaboration Technology Organizational Systems and Technology. Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences. Pages 713–722.

Ericsson, K.A. & Simon, H. A. 1993. Protocol Analysis - Revised Edition. MIT Press.

Gennaro, G. 1995. Hierarchical Object Oriented Requirements Analysis (HOORA). Preparing for the Future Vol. 5. No. 4. European Space Agency. http://esapub.esrin.esa.it/pff/pffv5n4/genv5n4.htm

Gilb, T. 2003. Competitive Engineering. Addison-Wesley.

Gotel, O. 1995. Contribution Structures for Requirements Traceability. Ph.D. Thesis. Imperial College of Science, Technology and Medicine. University of London.

Gotel, O. & Finkelstein, A. 1994a. An Analysis of the Requirements Traceability Problem. Proceedings of the First International Conference on Requirements Engineering. Pages 94–101.

Gotel, O. & Finkelstein, A. 1994b. Modelling the Contribution Structure Underlying Requirements. Proceedings of the First International Workshop on Requirements Engineering: Foundations of Software Quality (REFSQ-94).

Gotel, O. & Finkelstein, A. 1995. Contribution structures [Requirements artifacts]. Proceedings of the Second IEEE International Symposium on Requirements Engineering. Pages 100–107.

Gotel, O. & Finkelstein, A. 1997. Extended requirements traceability: results of an industrial case study. Proceedings of the Third IEEE International Symposium on Requirements Engineering. Pages 169–178.

Greenspan, S.; Mylopoulos, J. & Borgida, A. 1994. On formal requirements modeling languages: RML revisited. Proceedings of the 16th International Conference on Software Engineering. Pages 135–147.

Hadel, J.J. & Lakey, P.B. 1995. A customer-oriented approach for optimising reliability-allocation within a set of weapon-system requirements. Proceedings of the annual symposium on Reliability and Maintainability. Pages 96–101.

Hall, A. 2001. A Unified Approach to Systems and Software Requirements. The 5th IEEE Symposium on Requirements Engineering.

Harjani, D.-R. & Queille, J.-P. 1992. A process model for the maintenance of large space systems software. Proceedings of Conference on Software Maintenance. IEE Computer Press. Pages 127–136.

Hatley, D. J. & Pirbhai, I. A. 1987 Strategies for Real-Time System Specification. Dorset House.

Hooks, I. & Farry, K. 2001. Customer-Centred Products. Amacom.

Horowitz, E. & Williamson, R. 1986. SODOS : A Software Documentation Support Environment - its Use. IEEE Transactions on Software Engineering. November 1986. Pages 1076–1087.

IEEE Guide for Developing System Requirements Specifications (IEEE Std 1233–1998). 1998. Institute of Electrical and Electronics Engineering Inc.

IEEE Standard for Application and Management of the Systems Engineering Process (IEEE Std 1220–1998). 1998. Institute of Electrical and Electronics Engineering, Inc.

IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12 –1990). 1990. Institute of Electrical and Electronics Engineering, Inc.

IEEE Recommended Practice for Software Requirements Specifications (IEEE Std 830–1998). 1998. Institute of Electrical and Electronics Engineering, Inc.

In, H.; Olson, D. & Rodgers, T. 2001. A Requirements Negotiation Model Based on Multi-Criteria Analysis. International Symposium on Requirements Engineering.

Ince, D. 1994. Introduction to software quality assurance and its implementation, McGraw-Hill.

ISO/IEC 12207. 1995. Information technology - Software life cycle processes, International Standard.

ISO/IEC 13817-1. 1996. Information technology -- Programming languages, their environments and system software interfaces -- Vienna Development Method -- Specification Language -- Part 1: Base language. International Standard.

Jacobson, I.; Booch, G. & Rumbaugh, J. 1999. The Unified Software Development Process. Addison-Wesley.

Jacobson, I.; Christerson, M.; Jonsson, P., & Gunnar, O. 1992. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley.

Kazman, R.; Klein, M.; Barbacci, M.; Longstaff, T.; Lipson, H. & Carriere, J. 1998. The Architecture Tradeoff Method. Proceedings of the fourth IEEE International Conference on Engineering of Complex Computer Systems. Pages 68–78.

Kontio, J. 1995. OTSO: A Systematic Process for Reusable Software Component Selection. The Hughes Information Technology Corporation and the EOS Program. Version 1.0

Kotonya, G. & Sommerville, I. 1998. Requirements Engineering: Process and Techniques. John Wiley & Sons.

Lam, W.; Loomes, M. & Shankararaman, V. 1999. Managing requirements change using metrics and action planning. Proceedings of the Third European Conference on Software Maintenance and Reengineering. Pages 122–128.

Lang, M. & Duggan, J. 2001. A Tool to Support Collaborative Software Requirements Management. Requirements Engineering. Vol. 6. No. 3. Pages 161–172.

Lee, S. & Liu, B. 1993. IBO: an issue based object model for software design. Proceedings of the IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering. Vol. 1. Pages 270–274.

Leffingwell, D. & Widrig, D. 2000. Managing Software Requirements - A Unified Approach. Addison-Wesley.

Lindsay, P.; Yaowei Liu & Traynor O. 1997. A generic model for fine grained configuration management including version control and traceability. Proceedings of the Australian Software Engineering Conference. Pages 27–36.

Morisio, M.; Seaman, C.B.; Parra, A.T.; Basili, V.R.; Kraft, S.E. & Condon, S.E. 2000. Investigating and Improving a COTS-Based Software Development Process. Proceedings of the International Conference on Software Engineering.

Mullery, G.P. 1979. CORE – A Method for Controlled Requirement Specification. Proceedings of IEEE Fourth International Conference on Software Engineering.

Mäkäräinen, M. 1996. Application management requirements for embedded software. Technical Research Centre of Finland. VTT Publications 286. 99 p.

Mäkäräinen M. 2000. Software change management processes in the development of embedded software. Technical Research Centre of Finland. VTT Publications 416. 185 p. + app. 56 p.

Nelsen, E. D. 1990. System Engineering and Requirement Allocation. In Thayer, R.H. & Dorfman, M. IEEE System and Software Requirements Engineering. IEEE Software Computer Society Press Tutorial. IEEE Software Society Press.

Nuseibeh B. & Easterbrook S. 2000. Requirements Engineering: A Roadmap. In Finkelstein, A. (ed.). The Future of Software Engineering . Companion volume

to the proceedings of the 22nd International Conference on Software Engineering. IEEE Computer Society Press.

Olsen, N. 1993. The software rush hour. IEEE Software. Vol. 10. Issue 5. Pages 29–37.

Pinheiro, F. 2000. Formal and Informal Aspects of Requirements Tracing. III Workshop on Requirements Engineering.

Pohl, K. & Haumer, P. 1995. HYDRA: A Hypertext Model for Structuring Informal Requirements Representations. RWTH-Aachen, Informatik V.

Pressman, R. S. 1992. Software Engineering, A Practitioner's Approach, 3rd Edition. McGraw-Hill.

Ramesh, B.& Dhar, V. 1992. Supporting systems development by capturing deliberations during requirements engineering. IEEE Transactions on Software Engineering. Vol. 18. No. 6. Pages 498–510.

Ramesh, B. & Jarke, M. 2001. Toward reference models for requirements traceability. IEEE Transactions on Software Engineering. Vol. 27. Issue 1. Pages 58–93.

Rational Software Corporation. 1997. The Unified Modeling Language for Object-Oriented Development Documentation. v1.1. Rational Software Corporation.

Rawlinson, J.G. 1981. Creative Thinking and Brainstorming. Gower Publishing Company Limited.

Remillard, A. 1996. Software validation made simple. Proceedings of the Ninth IEEE Symposium on Computer-Based Medical Systems. Pages 36–40.

Robertson, S. & Robertson, J. 1999. Mastering the Requirements Process. Addison-Wesley.

Sailor, J. D. 1990. System Engineering: An Introduction. In: Thayer, R.H. & Dorfman, M. IEEE System and Software Requirements Engineering. IEEE Software Computer Society Press Tutorial. IEEE Software Society Press.

Sawyer, P. & Kotonya, G. 2001. Software Requirements. In the Trial Version (0.95) of SWEBOK, Guide to the Software Engineering Body of Knowledge. Chapter 2, pages 1-26. http://www.swebok.org/ (available 26.3.2002)

Schoman, K. & Ross, D.T. 1977. Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering. Pages 6–15.

Sheppard, D. 1995. An Introduction to Formal Specification with Z and VDM. McGraw-Hill.

SPICE. 1998. ISO/IEC TR 15504-2: Information Technology Software Process Assessment Part 2: A Reference Model for Processes and Process Capability. Technical Report type 2. International Organisation for Standardisation (ed.).

Sommerville, I. & Sawyer, P. 1997. Requirements Engineering: A Good Practice Guide. John Wiley & Sons.

Stankovic, J. A. 1996. Real-Time and Embedded Systems. ACM Computing Surveys. Vol. 28. No. 1. Pages 205–208.

Stevens, R.; Brook, P.; Jackson, K. & Arnold, S. 1998. Systems Engineering - Coping with Complexity. Prentice Hall.

Thayer, R. H. & Royce, W. W. 1990. Software Systems Engineering. In: Thayer, R.H. & Dorfman, M. IEEE System and Software Requirements Engineering. IEEE Software Computer Society Press Tutorial. IEEE Software Society Press.

Toranzo M. & Castro J. 1999. Multiview++ Environment: Requirements Traceability from the perspective of different stakeholders. II IberoAmerican Workshop on Requirements Engineering.

Wiegers, K.E. 1999. Software requirements. Microsoft Press.

Vierimaa, M., Ronkainen, J., Salo, O., Sandelin, T., Tihinen, M., Freimut, B. & Parviainen, P. 2001. MIKKO Handbook: Comprehensive collection and utilisation of software measurement data. Pages 96–100. Technical Research Centre of Finland. VTT Publications 445.

West, M. 1991. Quality function deployment in software development. IEE Colloquium on Tools and Techniques for Maintaining Traceability During Design. Pages 5/1–5/7.

Yu, W. D. 1994. Verifying software requirements: a requirement tracing methodology and its software tool-RADIX. IEEE Journal on Selected Areas in Communications. Vol. 12. No. 2. Pages. 234–240.

Zave, P. 1982. An Operational Approach to Requirements Specification for Embedded Systems. IEEE Transactions on Software Engineering. Vol. 8. No. 5. Pages 250–269.

Published by

Series title, number and report code of publication

VTT Publications 508
VTT–PUBS–508

Author(s)
Parviainen, Päivi, Hulkko, Hanna, Kääriäinen, Jukka, Takalo,Juha & Tihinen, Maarit

Title

Requirements engineering
Inventory of technologies

Abstract

The purpose of this publication is to describe existing systems and software requirements engineering techniques, methods and tools based on a literature study. This publication covers a wide range of requirements engineering methods and theoretical issues and thus provides a broad view of the field. Also, some RE tools are described.

Requirements engineering is also described in general and RE processes introduced to provide background information about RE and help to understand the method descriptions. The main processes of RE as seen in this publication include: System requirements development, requirements allocation and flow-down, software requirements analysis and specification and continuous processes including requirements documentation, requirements validation and verification and requirements change management. Requirement Management (RM) activities are understood to begin before actual requirements engineering process phases (RM planning) and continuing during design, implementation, testing and maintenance phases.

Keywords
Requirements engineering (RE), RE methods, RE techniques, RE tools, system and software engineering

Activity unit
VTT Electronics, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland

ISBN			Project number
951–38–6245–3 (soft back ed.) 951–38–6246–1 (URL:http://www.vtt.fi/inf/pdf/)			E2SU00054
Date	Language	Pages	Price
November 2003	English	106 p.	C
Name of project MOOSE (Software engineering methodologies for embedded systems)		Commissioned by ITEA, National Technology Agency of Finland Tekes	
Series title and ISSN VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)		Sold by VTT Information Service P.O.Box 2000, FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374	

VTT PUBLICATIONS

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU PL 2000 02044 VTT Puh. (09) 456 4404 Faksi (09) 456 4374	VTT INFORMATIONSTJÄNST PB 2000 02044 VTT Tel. (09) 456 4404 Fax (09) 456 4374	VTT INFORMATION SERVICE P.O.Box 2000 FIN–02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374

ISBN 951–38–6245–3 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–6246–1 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0849 (URL: http://www.vtt.fi/inf/pdf/)