Requirements Engineering:

Dealing with the Complexity of Sociotechnical Systems Development

Päivi Parviainen

VTT Technical Research Centre of Finland, VTT Electronics
P.O. Box 1100, FIN-90571 Oulu, Finland
Tel. +358 8 551 2471, Fax +358 8 551 2320

Maarit Tihinen

VTT Technical Research Centre of Finland, VTT Electronics
P.O. Box 1100, FIN-90571 Oulu, Finland
Tel. +358 8 551 2286, Fax +358 8 551 2320

Marco Lormans

Department of Information, Systems and Algorithms,
Delft University of Technology,
Mekelweg 4, 2628CD Delft, The Netherlands.
Tel. +31152788872, Fax +31152786632

Rini van Solingen

LogicaCMG Technical Software Engineering (RTSE1)
PO Box 8566, 3009 AN Rotterdam, The Netherlands
Tel. +31102537247, Fax +31102537033

Requirements Engineering:

Dealing with the Complexity of Sociotechnical Systems Development
This chapter introduces requirements engineering for socio-technical systems. Requirements engineering for socio-technical systems is a complex process that considers product demands from a vast number of viewpoints, roles, responsibilities, and objectives. This chapter explains the requirements engineering terminology and describes requirements engineering process in detail, with examples of available methods for the main process activities. The main activities described include system requirements development, requirements allocation and flow-down, software requirements development and continuous activities, including requirements documentation, requirements validation and verification and requirements management. As requirements engineering is the process with the largest impact on the end-product, it is recommended to invest more effort in both industrial application as well as research to increase understanding and deployment of the concepts presented in this chapter.

INTRODUCTION

The concept of socio-technical systems was established to stress the reciprocal interrelationship between humans and machines and to foster the program of shaping both theoretical and the social conditions of work (Ropohl, 1999). A socio-technical system can be regarded as a theoretical construct for describing and explaining technology generally. This chapter helps to describe a multidisciplinary role of requirements engineering as well as the concept of workflow and patterns for social interaction within the socio-technical systems research area.

Requirements engineering is generally accepted to be the most critical and complex process within the development of socio-technical systems (Juristo et al., 2002; Komi-Sirviö &
Tihinen, 2003; Siddiqi, 1996). The main reason is that the requirements engineering process has the most dominant impact on the capabilities of the resulting product. Furthermore, requirements engineering is the process in which the most diverse set of product demands from the most diverse set of stakeholders is being considered. These two reasons make requirements engineering complex as well as critical.

This chapter first introduces background information related to requirements engineering, including the terminology used and the requirements engineering process in general. Next a detailed description of the requirements engineering process, including the main phases and activities within these phases, is presented. Each phase will be discussed in detail, with examples of useful methods and techniques.

**BACKGROUND**

A requirement is 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 (IEEE Std 610.12 – 1990). A well-formed requirement is a statement of system functionality (a capability) that must be met or possessed by a system to satisfy a customer’s need or to achieve a customer’s objective, and that is qualified by measurable conditions and bounded by constraints (IEEE Std 1233-1998).

Requirements are commonly classified as (IEEE Std 830 – 1998):

- **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. Non-functional requirements are often classified into the following categories:

- **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 requirement which affects or constrains the design of a system or 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 a system possesses attributes that affect quality, e.g., correctness, reliability, maintainability, portability.

Requirements engineering contains a set of activities for discovering, analysing, documenting, validating and maintaining a set of requirements for a system (Sommerville & Sawyer, 1997). Requirements engineering is divided into two main groups of activities, requirements development and requirements management. *Requirement development* includes activities related to discovering, analysing, documenting and validating requirements, where as *requirement management* includes activities related to maintenance, namely identification, traceability and change management of requirements.
Requirements validation consists of activities that try to confirm that the behaviour of a developed system meets its *user needs*. Requirements verification consists of those activities that try to confirm that the product of a system development process meets its technical specifications (Stevens et al., 1998). Verification and validation include:

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

In socio-technical systems software is understood as a part of the final product. System requirements are captured to identify the functioning of the system, from which software requirements are derived. Deciding which functionality is implemented where, and by which means (software, hardware, mechanics, etc.) is merely a technical decision process in which feasibility, dependability and economics play a role. A well-structured and technically sound requirements engineering process is, therefore, of utmost importance.

**REQUIREMENTS ENGINEERING PROCESS**

Figure 1 describes a requirements engineering process where the main processes of system and software requirements engineering are depicted. Requirements engineering activities
cover the entire system and software development lifecycle. On the other hand, the requirements engineering process is iterative and will go into more details in each iteration. In addition, the figure indicates, how requirements management (RM) is understood as a part of the requirements engineering process. The process is based on (Kotonya & Sommerville, 1998; Sailor, 1990; Thayer & Royce, 1990).

The process describes requirements engineering for socio-technical systems, where software requirements engineering is a part of the process. Traditionally, requirements engineering is performed in the beginning of the system development lifecycle (Royce, 1970). However, in large and complex systems development, developing an accurate set of requirements that would remain stable throughout the months or years of development has been realised to be impossible in practice (Dorfman, 1990). Therefore, requirements engineering is an incremental and iterative process, performed in parallel with other system development activities such as design.
Figure 1. **System and software requirements engineering (Parviainen et al., 2003)**

The main high level activities included in the requirements engineering process are

1) *System requirements development*, including requirements gathering/elicitation from various sources (Figure 1 shows the different sources for requirements), requirements analysis, negotiation, priorisation and agreement of raw requirements, and system requirements documentation and validation.

2) *Requirements allocation and flow-down*, including allocating the captured requirements to system components and defining, documenting and validating detailed system requirements.

3) *Software requirements development*, including analysing, modelling and validating both the functional and quality aspects of a software system, and defining, documenting and validating the contents of software subsystems.

4) *Continuous activities*, including requirements documentation, requirements validation and verification, and requirements management.

Each of these high level activities will be further detailed in the following sections.

**System Requirements Development**

The main purpose of the system requirements development phase is to examine and gather desired objectives for the system from different viewpoints (e.g., customer, users, system's operating environment, trade, and marketing). These objectives are identified as a set of
functional and non-functional requirements of the system. Figure 2 shows the context for developing system requirements specification (SyRS).

![Figure 2. The context for developing SyRS (IEEE Std 1233-1998)](image)

1. **Requirements gathering/elicitation from various sources**

Requirements gathering starts with identifying the stakeholders of the system and collecting (i.e., eliciting) raw requirements. Raw requirements are requirements that have not been analysed and have not yet been written down in a well-formed requirement notation. Business requirements, customer requirements, user requirements, constraints, in-house ideas and standards are the different viewpoints to cover. Typically, specifying system requirements starts with observing and interviewing people (Ambler, 1998). This is not a straightforward task, because users may not possess the overview on feasibilities and opportunities for automated support. Furthermore, user requirements are often misunderstood because the requirements collector misinterprets the users’ words. In addition to gathering requirements from users, also standards and constraints (for example, the legacy systems) play an important role in systems development.
2. Requirements analysis and documentation

After the raw requirements from stakeholders are gathered, they need to be analysed within the context of business requirements (management perspective) such as cost-effectiveness, organisational and political requirements. Also, the requirements relations, that is, dependencies, conflicts, overlaps, omissions and inconsistencies, need to be examined and documented. Finally, the environment of the system, such as external systems and technical constraints, need to be examined and explicated.

The gathering of requirements often reveals a large set of raw requirements that due to cost and time constraints cannot entirely be implemented in the system. Also, the identified raw requirements may be conflicting. Therefore, negotiation, agreement, communication and priorisation of the raw requirements are also an important part of the requirements analysis process.

The analysed requirements need to be documented to enable communication with stakeholders and future maintenance of the requirements and the system. Requirements documentation also includes describing the relations between requirements. During requirements analysis it gives added value to record the rationale behind the decisions made to ease future change management and decision making.

3. System requirements validation and verification

In system requirements development, validation and verification activities include validating the system requirements against raw requirements and verifying the correctness of system
requirements documentation. Common techniques for validating requirements are requirements reviews with the stakeholders, and prototyping.

Table 1 contains examples of requirements engineering methods and techniques used during the system requirements development phase. The detailed descriptions of the methods have been published in (Parviainen et al., 2003).

**Table 1. System requirements development methods**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Example methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gathering requirements</td>
<td>Ethnography (Suchman, 1983)</td>
<td><strong>Observing methods</strong> use techniques that may help to understand the thoughts and needs of the users, even when they cannot describe these needs or they do not exactly know what they want.</td>
</tr>
<tr>
<td></td>
<td>Protocol Analysis (Ericsson &amp; Simon, 1993)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Focus groups (Maguire, 1998)</td>
<td><strong>Meeting techniques</strong> cover separate techniques for meetings and workshops for gathering and developing requirements from different stakeholders.</td>
</tr>
<tr>
<td></td>
<td>JAD (Joint Application Development) (Ambler, 1998)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Volere (Robertson &amp; Robertson, 1999)</td>
<td>Provides a generic process for gathering requirements, ways to elicit them from users, as well as a process for verifying them.</td>
</tr>
<tr>
<td>Requirements analysis</td>
<td>QFD (Quality Function Deployment) (Revelle et al., 1998)</td>
<td>Identifying customer needs, expectations and requirements, and linking them into the company's products.</td>
</tr>
<tr>
<td></td>
<td>SCRAM (Scenario-based Requirements Engineering)</td>
<td>Develop requirements (whole RE) using scenarios. The scenarios are created to represent paths of possible behaviour through use cases, and these are then investigated to develop requirements.</td>
</tr>
<tr>
<td></td>
<td>(Sutcliffe, 1998)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SSADM (Structured System Analysis and Design Methodology) (Ashworth, 1990)</td>
<td>Can be used in the analysis and design stages of systems development. It specifies in advance the modules, stages and tasks which have to be carried out, the deliverables to be produced and the techniques used to produce those deliverables.</td>
</tr>
</tbody>
</table>
Negotiation and prioritisation

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE (Controlled Requirements Expression) (Mullery, 1979)</td>
<td>The purpose of viewpoint oriented methods is to produce or analyse requirements from multiple viewpoints. They can be used while resolving conflicts or documenting system and software requirements.</td>
</tr>
<tr>
<td>WinWin approach (Bose, 1995)</td>
<td></td>
</tr>
<tr>
<td>System requirements documentation</td>
<td>IEEE Std 1233-1998</td>
</tr>
<tr>
<td>VDM (Vienna Development Model) (Björner &amp; Jones, 1978)</td>
<td>Standards define the contents of a SyRS.</td>
</tr>
<tr>
<td>Specification language Z (Sheppard, 1995)</td>
<td>In formal methods requirements are written in a statement language or in a formal - mathematical - way.</td>
</tr>
<tr>
<td>HPM (Hatley-Pirbhai Methodology) (Hatley &amp; Pirbhai, 1987)</td>
<td>Gives support for documenting and managing of system requirements.</td>
</tr>
<tr>
<td>VORD (Viewpoint-Oriented Requirements Definition) (Kotonya &amp; Sommerville, 1996)</td>
<td>Helps to identify and prioritise requirements and also can be utilised when documenting system and software requirements.</td>
</tr>
</tbody>
</table>

Several methods for gathering, eliciting and analysing requirements from users and other stakeholders can be used. Table 1 includes several observing methods (e.g., ethnography), meeting techniques (e.g., focus groups) and analysing techniques (e.g., QFD) that can be used to gather requirements and avoid misunderstandings of users needs. The methods help identifying needs of individuals and converting them into requirements of a desired product. At the same time social actions and workflows, safety-critical aspects, or technical constraints have to be taken into consideration. The results of the system requirements development phase are captured as top-level system requirements that are used as input for the allocation and flow-down phase.

**Allocation and Flow-down**

The requirements allocation and flow-down process’ purpose is to make sure that all system requirements are fulfilled by a subsystem or by a set of subsystems collaborating together.
Top-level system requirements need to be organized hierarchically, helping to view and manage information at different levels of abstraction. The requirements are decomposed down to the level at which the requirement can be designed and tested, thus, allocation and flow-down may be done for several hierarchy levels. 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. (Leffingwell & Widrig, 2000; Stevens et al., 1998).

The top-level system requirements defined in the system requirements development phase (see previous subsection) are the main input for the requirements allocation and flow-down phase. In practice, system requirements development and allocation and flow-down are iterating; as the system level requirements are being developed, the elements that should be defined in the hierarchy should also be considered. By the time a draft of the system requirements is complete, a tentative definition of at least one and possibly two levels of system hierarchy should be available. (Dorfman, 1990).

1. Requirements Allocation

Allocation is architectural work carried out in order to design the structure of the system and to issue the top-level system requirements to subsystems. Architectural models provide the context for defining how applications and subsystems interact with one another to meet the requirements of the system. The 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 (Ambler, 1998).
Each system level requirement is allocated to one or more elements at the next level (i.e., it is determined which elements will participate in meeting the requirement). Allocation also includes allocating the non-functional requirements to system elements. Each system element will need an apportionment of the non-functional requirements (e.g., performance requirement). However, not all requirements are 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. The allocation process is iterative; in performing the allocation, needs to change the system requirements (additions, deletions, and corrections) and/or the definitions of the elements can be found (Dorfman, 1990; Nelsen, 1990; Pressman, 1992; Sailor, 1990).

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

- Definition of alternative approaches.
- Evaluation of alternatives.
- Selection of evaluation criteria; performance, effectiveness, life-cycle cost factors.
- Application of analytical techniques (e.g., models).
- Data generation.
- Evaluation of results.
- Sensitivity analysis.
- Definition of risk and uncertainty.

Once the functionality and the non-functional requirements 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. The decomposition is done right when:

- Distribution and partitioning of functionality are optimized 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 regard is given to the physical domain - the size, weight, location, and distribution of the subsystems - that has been optimized in the overall system context. (Leffingwell & Widrig, 2000).

2. 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 recognize 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 that must be imposed on the subsystems themselves but do not necessarily provide a direct benefit to the end user.

- Interface requirements that 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).

In the allocation and flow-down phase, requirements identification and traceability have to be ensured both to higher level requirements as well as between requirements on the same level. Also, the rationale behind design decisions should be recorded in order to ensure that there is enough information for verification and validation of the next phases work products and change management.

The flowing-down of the top-level system requirements through the lower levels of the hierarchy until the hardware and software component levels are reached, in theory, produces a system in which all elements are completely balanced or “optimized”. In the real world, complete balance is seldom achieved, due to fiscal, schedule, and technological constraints (Sailor, 1990; Nelsen, 1990).

Table 2 includes few examples of methods available for the allocation and flow-down.

Table 2. Allocation and Flow-down methods

<table>
<thead>
<tr>
<th>Activity</th>
<th>Example methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation</td>
<td>SRA (System Requirements Allocation Methodology) (Hadel &amp; Lakey, 1995)</td>
<td>A customer-oriented systems engineering approach for allocating top-level quantitative system</td>
</tr>
</tbody>
</table>
The results of allocation and flow-down are detailed system level requirements and the “architectural design” or “top-level design” of the system. Again, needs to change the system requirements (additions, deletions, and corrections) and/or the definitions of the system elements may be found. These are then fed back to the system requirements development process. Allocation and flow-down starts as a multi-disciplinary activity, i.e., subsystems may contain hardware, software, and mechanics. Initially they are considered as one subsystem, in later iterations the different disciplines are considered separately. Software requirements development will be described in detail in the next section.

**Software Requirements Development**

The software requirements development process is the activity determining which functionality of the system will be performed by software. Documenting this functionality together with the non-functional requirements in a software requirements specification is part of this phase. 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 (Thayer & Royce, 1990).

1. **Software requirements analysis**

Software requirements analysis is a software engineering task that bridges the gap between system level software allocation and software design. Requirements analysis enables the specification of software functions and performance, an indication of the software interfaces with other system elements, and the establishment of design constraints that the software must meet. Requirements analysis also refines the software allocation and builds models of the process, data, and behavioral domains that will be treated by software. Prioritising the software requirements is also part of software requirements analysis. To support requirements analysis, the software system may be modelled, covering both functional and quality aspects.

2. **Software requirements documentation**

In order to be able to communicate software requirements and to make changes to them, they need to be documented in a software requirements specification (SRS). A SRS contains a complete description of the external behaviour of the software system. It is possible to complete the entire requirements analysis before starting to write the SRS. However, it is more likely that as the analysis process yields aspects of the problem that are well understood, the corresponding section of the SRS is written.

3. **Software requirements validation and verification**

Software requirements need to be validated against system level requirements and the SRS needs to be verified. Verification of SRS includes, for example, correctness, consistency, unambiguousness and understandability (IEEE Std 830-1998).
A requirements traceability mechanism to generate an audit trail between the software requirements and final tested code should be established. Traceability should be maintained to system level requirements, between software requirements, and to later phases, e.g., architectural work products.

The outcome of the software requirements development phase is a formal document including a baseline of the agreed 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.
- Analyzed, 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.

Table 3 gives examples of methods or techniques available for software requirements development.
<table>
<thead>
<tr>
<th>Activity</th>
<th>Example methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software requirements analysis</td>
<td>OMT (Object Modeling Technique) (Bourdeau &amp; Cheng, 1995)</td>
<td><em>Object oriented methods</em> model systems in an object oriented way or support object-oriented development in the analysis and design phases.</td>
</tr>
<tr>
<td></td>
<td>Shlaer-Mellor Object-Oriented Analysis Method (Shlaer &amp; Mellor, 1998)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UML (Unified Modeling Language) (Booch et al., 1998)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SADT (Structured Analysis and Design Technique) (Schoman &amp; Ross, 1977)</td>
<td><em>Structured methods</em> analyse systems from process and data perspective by structuring a project into small, well-defined activities and specifying the sequence and interaction of these activities. Typically diagrammatic and other modelling techniques are used during analysis work.</td>
</tr>
<tr>
<td></td>
<td>SASS (Structured Analysis and System Specification) (Davis, 1990)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B-methods (Schneider, 2001)</td>
<td>Formal methods are often used for safety-critical systems.</td>
</tr>
<tr>
<td></td>
<td>Petri Nets (Petri, 1962; Girault &amp; Valk, 2002)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XP (Extreme Programming) (Beck, 1999)</td>
<td>Agile methods are not specially focused on RE, but they have an integral point of view, where RE is one of the activities of the whole cycle. See more from (Abrahamsson et al., 2002)</td>
</tr>
<tr>
<td></td>
<td>CARE (COTS-Aware Requirements Engineering) (Chung et al., 2001)</td>
<td>Specific methods for RE when using COTS (Commercial off-the-shelf). The COTS is a ready-made software product, which is supplied by a vendor and has specific functionality as part of a system.</td>
</tr>
<tr>
<td></td>
<td>OTSO (Off-the-Shelf Option) (Kontio, 1995)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Planguage (Gilb, 2003)</td>
<td>Consists of a software systems engineering language for communicating systems engineering and management specifications, and a set of methods providing advice on best practices.</td>
</tr>
<tr>
<td>Requirements documentation</td>
<td>IEEE Std 830-1998</td>
<td>IEEE defines contents of an SRS. 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</td>
</tr>
<tr>
<td>Requirements validation</td>
<td>Volere (Robertson &amp; Robertson, 1999)</td>
<td>Provides process for gathering /eliciting and validating both system and software requirements.</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Storyboard Prototyping (Andriole, 1989)</td>
<td>Sequences of computer-generated displays, called storyboards, are used to simulate the functions of the formally implemented system beforehand. This supports the communication of system functions to the user, and makes the trade-offs non-functional and functional requirements visible, traceable and analysable.</td>
</tr>
<tr>
<td></td>
<td>Also several other methods such as object oriented methods provide some support for validation and verification</td>
<td></td>
</tr>
</tbody>
</table>

**Continuous activities**

Documentation, validation and verification of the continuous activities are included in the main process phase where the activity is mentioned the first time. Only requirements management viewpoints (identification, traceability and change management) are discussed in this section.

Requirements Management controls and tracks changes of agreed requirements, relationships between requirements, and dependencies between the requirements documents and other documents produced during the systems and software engineering process (Kotonya & Sommerville, 1998). Requirements management is a continuous and cross-section process that begins from requirements management planning and continues via activities of identification, traceability and change control during and after requirements development process phases. Requirements management continues also after development during maintenance, because the requirements continue to change (Kotonya & Sommerville, 1998;
Lauesen, 2002). Each of the requirements management activities are introduced in the following.

1. Requirements identification

Requirements identification practices focus on the assignment of a unique identifier for each requirement (Sommerville & Sawyer, 1997). These unique identifiers are used to refer to requirements during product development and management. Requirements’ identification support can be divided into the three classes (Sommerville & Sawyer, 1997; Berlack, 1992):

1. Basic numbering systems
   - Significant numbering
   - Non-significant numbering

2. Identification schemes
   - Tagging
   - Structure based identification
   - Symbolic identification

3. Techniques to support and automate the management of items
   - Dynamic renumbering
   - Database record identification
   - Baselining requirements

2. Requirements traceability

Requirements traceability refers to the ability to describe and follow the life of a requirement and its relations with other development artefacts in both forwards and backwards direction (Gotel, 1995). This is especially important for trade-off analysis, impact analysis, and verification & validation activities. If traceability is not present it is very difficult to identify
what the effects of proposed changes are, and whether accepted changes are indeed taken care of.

3. Requirements change management

Requirements change management refers to the ability to manage changes to requirements throughout the development lifecycle. Change management, in general, includes identifying, analysing, deciding on whether a change will be implemented, implementing and validating change requests. Change management is sometimes said to be the most complex requirements engineering process (Hull et al., 2002). A change can have a large impact on the system and estimating this impact is very hard. Requirements traceability helps making this impact explicit by using downwards and upwards traceability. For every change the costs and redevelopment work have to be considered before approving the change. Change management has a strong relationship with baselining. After requirements’ baselining, changes to the requirements need to be incorporated by using change control procedures (Hooks & Farry, 2001).

Examples of requirements management methods, techniques or approaches have been listed in the Table 4.

Table 4. Requirements management methods

<table>
<thead>
<tr>
<th>Activity</th>
<th>Example methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements traceability</td>
<td>Cross reference, Traceability matrices, Automated traceability links. (Sommerville &amp; Sawyer, 1997)</td>
<td>Techniques can be used for presenting and managing requirements as separate entities, and describing and maintaining links between them, e.g., during allocation, implementation or verification.</td>
</tr>
<tr>
<td></td>
<td>IBIS derivatives (Pinheiro, 2000) Document -centred models</td>
<td>Methods present traces and provide information to capture design</td>
</tr>
<tr>
<td>Change management</td>
<td>Olsen’s ChM model (Olsen, 1993)</td>
<td>Change management process models and approaches.</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>V-like model (Harjani &amp; Queille, 1992)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ince's ChM model (Ince, 1994)</td>
<td></td>
</tr>
</tbody>
</table>

Requirements management tools have been developed because of the problems of managing unstable requirements and the large amount of data collected during requirements engineering process. A large set of tools - both commercial and non-commercial) – is available, for examples see (Parviainen et al, 2003). Requirements management 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). 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.

**CONCLUSIONS**

Requirements engineering for socio-technical systems is a complex process that considers product demands from a vast number of viewpoints, roles, responsibilities, and objectives. In this chapter we have explained the activities of requirements engineering and their relations to the available methods. A large set of methods is available, each with their specific strengths and weaknesses. The methods’ feasibility and applicability do, however, vary between phases or activities. Method descriptions also often lack the information of the methods' suitability to different environments and problem situations, thus making the selection of an applicable method or combination of methods to be used in a particular real-life situation, complicated.

Requirements engineering deserves a stronger attention from practice as the possibilities of available methods are largely overlooked by industrial practice (Graaf et al., 2003). As requirements engineering is the process with the largest impact on the end-product, it is recommended to invest more effort in both industrial application as well as research to increase understanding and deployment of the concepts presented in this chapter.
This chapter has only listed a few examples of methods. For a more comprehensive listing of methods see (Parviainen et al., 2003).
REFERENCES


Mathematik. Schriften des IIM Nr. 2.

Requirements Engineering, Rio de Janeiro, Brazil.

McGraw-Hill Inc.

Wiley & Sons.


Technology 4(3), URL:
http://scholar.lib.vt.edu/ejournals/SPT/v4_n3pdf/ROPOHL.PDF (available 05.02.2004).

California. USA.


---

1 This chapter describes the requirements engineering process based on work done in the MOOSE (Software engineering methodologies for embedded systems) project (ITEA 01002). The authors would like to thank the partners in the MOOSE project (http://www.mooseproject.org/), as well as the support from ITEA and the national public authorities.