

Requirements Engineering - Concepts and Relations

Antti Tevanlinna

Helsinki 16th September 2003

Seminar report

UNIVERSITY OF HELSINKI

Department of Computer Science

Contents

1	Introduction	1
2	Concepts of requirements engineering	1
3	Two processes for requirements engineering	5
4	Discussion	7
5	Conclusions	9
	References	9

1 Introduction

Requirements engineering may be the discipline, where the most anecdotes and truisms are used to describe the authors' message to the reader. One of the most famous phrases is also used by Barry Boehm. He says: "Give a man a hammer, and he will begin to see the world as a collection of nails". The two sentences clarify one of the most central failures made by us, the people who build software. We have often built software that solves the problem in our perception, but doesn't solve the real needs of the user [Boe81].

Another commonly used phrase is the Chinese proverb, that states: "if you don't know where you're going, any road will do". This could be applied to requirements engineering so that it states: "if you don't know what you're building, you can build anything". As a more mathematically oriented person would assert: "a problem unstated is a problem unsolved". [RS77].

Boehm provides us the last figurative tale of the consequences of bad requirements engineering [Boe73].

"...a genie comes up to a man and tells him he has three wishes and can ask for anything in the world. Typically, he spends his first two wishes asking for something like a golden castle and a princess, and then when he discovers the operations, maintenance, and compatibility implications of his new acquisitions, he is happy to spend the third wish getting back to where he started."

Requirements engineering is a most important activity in systems development. As noted by Brooks: "no other part of the work so cripples the resulting system if done wrong" [Bro87]. Overlooking requirements engineering or getting it wrong may lead to cancellation of the project (Bray lists five multi-million projects that have been canceled [Bra02]) or extensive modifications after initial delivery of the system (Boehm reports of recodes of up to 67% in systems and 95% in one particular system [Boe73]).

This report will survey the concepts of requirements engineering and their relations as they are seen classically. In the next section the most central definitions of requirements engineering are introduced. Following that two processes for requirements engineering are sketched out. In Section 4 some aspects of the validity of requirements engineering as introduced by this report are discussed. The report is concluded in the last section.

2 Concepts of requirements engineering

To many requirements engineering is an unfamiliar discipline in software engineering. Requirements engineering has been established since the 70's [Ros77a]. Also formal techniques for requirements engineering have as long history as the discipline [RS77, Ros77b].

Systematic methods to information system construction involving requirements engineering have been studied also in Finland as early as 1970 [Jah70]. The interest in requirements engineering was renewed in the 90's with the publications of several special issues in journals ¹, the forming of requirements engineering conferences ² and the starting of the "Requirements Engineering" journal in 1996.

In this report a system is seen in a broad view. A system is a set of hardware, software, people, data, facilities, procedures and their organization.

To be very short requirements engineering is the application of "engineering" -methods to the gathering of the "requirements" of a system. This, however, does not state much about anything without knowing what "engineering" and "requirement" mean. A longer definition includes decompositions of the both.

By engineering we mean using systematic and controlled, and therefore repeatable techniques to solve something. Engineering is the application of the scientific method to some problem. The problem is analyzed and a solution that is as precise and efficient as possible is created.

Before we can introduce the definitions of the concept of requirement, we must explore the context it exists in. A requirement is always bound to a *problem domain*, that is that part of the universe within which the problem exists [Bra02]. For example a problem domain for a conventional lift system could be the building where the lift is to be built, its users and the current lift building hardware. As we will see later there are more elements in the problem domain of a lift control system.

Now a requirement can be defined as "the effects that the client wishes to be brought in the problem domain" [Bra02]. The standard vocabulary of the software engineering includes a very similar, but distinctly different, definition of a requirement: "a condition or a capability that must be met or possessed by the system or system component to satisfy a contract, standard, specification,..." [IEE90]. The distinction is made clear in the last definition that states that "a software requirements specification (SRS) is a document that defines *what* the software will do without describing *how* it will do it" [Dav90]. The activity of collecting requirements is most often called *elicitation*.

Let us explore possible requirements in an imaginary problem domain of WWW-browsing programs. An example of a requirement could be: "the user can enable or disable cookies". Although it is obvious to WWW-programmers what "cookies" and "enabling" mean, this is not the case for all the people that are dependent of the system. The requirement is incomprehensible to most of the end users, the management and many of the other

¹Special issues in major journals include IEEE Trans. on Software Eng., Mar. '91, IEEE Software, Mar. '94, Communications of the ACM, May '95, IEEE Software mar. '96, Communications of the ACM, Dec. '98, IEEE Software Jan. '03.

²The First International Symposium on Requirements Engineering, ISRE, was held in '93 and the First International Conference on Requirements Engineering, ICRE, was held in '94.

people who are affected by the system. Therefore also definitions to "cookie" and "enabling" must be formed. Sommerville and Sawyer point out that there is no set precision in which the requirements should be stated. It depends on the organization, project and usage of the requirements [SS97]. Nor is there a best way to write requirements [SS97].

As noted before, many different people have interests in the designed system. These people are often referred as *stakeholders*. The first thought is that the stakeholders include the users, the designers, and the builders of system. This is only a partial view because every person that might be associated with the system is a stakeholder. In the context of a lift control system stakeholders include also managements of both the contractor and the supplier, lift security inspectors, lift repairmen, the house building company, and so on.

Every stakeholder sees the problem domain and the designed system in her own view. A stakeholder's *viewpoint* is partial subjective information about a requirement as seen by the stakeholder. In elicitation an understanding of the requirements, that is as comprehensive and representative, is achieved by collecting all the viewpoints together.

There are two classic ways to classify requirements. The first classification is the division between *functional* and *non-functional* requirements. The second classification is the separation of *explicit* and *implicit* requirements.

The functional requirements (also known as behavioral requirements) state the required effects to problem domain caused by the system. All the examples so far have been functional requirements on a system. The non-functional requirements (also known as performance or non-behavioral requirements) place requirements on how the functional requirements are fulfilled. Common non-functional requirements include security, reliability, usability and performance related requirements. A non-functional requirement could be something like "the application shall be able to serve 100 concurrent customers".

The distinction between the explicit and the implicit requirements is that the explicit requirements are stated by the stakeholders, and the implicit requirements are only assumed by the stakeholders. The implicit requirements include general usability, absence of errors, contemporaneity, availability of maintenance, and general efficiency. To generalize, implicit requirements are the stakeholders' notions of software quality. By not meeting some implicit requirements software can be unsatisfactory although it fulfills its explicit requirements.

When a set of requirements is collected (elicited), it is quite obvious that the produced system should fulfill the requirements as well as possible. This system is often referred as a *solution system*, an application or just a system. The solution system interacts in various ways with the problem domain to create the required effects in the problem domain. The relationship can be visualized in two ways. In Figure 1 the system is located in the problem domain. This depicts how a system interacts with a subset of the problem domain. On the other hand, the system is not a part of the problem domain. It interacts with it through

an interface which is the functions of the system. This is depicted in the Figure 2.

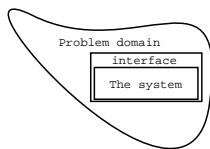


Figure 1: Problem domain and the system [Bra02, Dav90].

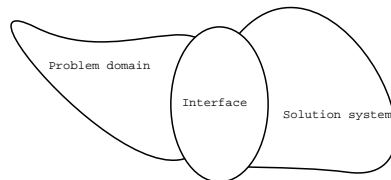


Figure 2: Problem domain, the system and their interface [Bra02].

Fulfilling the raw requirements elicited from the stakeholders is usually not possible because the requirements are subjective. The raw requirements are partly overlapping and in the same time partly incomplete. Being subjective they are also often erroneous and contradictory. Contradictions also emerge from different expectations expressed by different stakeholders. *Requirements analysis* is the activity, where the raw requirements are melded into a set of requirements that is compact, uncontradictory and comprehensive. An activity related to analysis is *requirements negotiation*. In it negotiations between the stakeholders and the requirements engineers are held. In those negotiations the conflicting requirements are redefined or removed to resolve the conflicts.

Requirements analysis prepares the stakeholders needs for fulfillment. The ultimate product of the requirements engineering is the description of a system that fulfills the requirements. This is done in *specification* by writing down the functionality of the system. Now the relationships of the problem domain, the solution system and their interface can be mapped to various activities in the requirements engineering. In Figure 3 elicitation and analysis are connected to the task of defining the problem domain, and specification is connected to the task of specifying the functionality of the solution system. The internal design and the implementation of the solution system is not regarded as a part of requirements engineering.

The specification is a description of the functions of the system that fulfills the requirements in the problem domain. Some requirements are suitable as specifications themselves, and those requirements can be rewritten into specifications with little effort. Even though the requirements should not be in the form of specifications, it seems to be natural to many stakeholders to state them in such a form. Requirements are often stated simulating

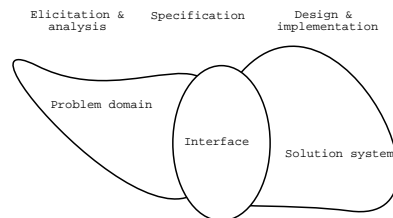


Figure 3: Activities of the requirements engineering and the subjects of their interest (adapted from Bray [Bra02]).

usage. Requirement "when I push a button, I see my account information" is an example of such. Some requirements are not in the form of the specification of system functionality. Consider a requirement of "a person with diminished eyesight should be able to use the system". This requirement does not tell anything about the interaction of the system with the problem domain. It is up to the engineer to specify the functions so that the requirement is met.

There is often only a fine line, if any, between the requirements and the specifications. One of the largest sources of confusion in the requirements engineering is just this. The results of elicitation, analysis and even specification are often called requirements without care about confusion.

3 Two processes for requirements engineering

By doing performing the tasks of requirements engineering in some order we come to a requirements engineering process. A simple requirements engineering process includes elicitation, analysis, specification, validation and construction sequentially (see figure 4). This process bears resemblance to the classical "Waterfall" model [Roy70] of system construction. The tasks in the process have clear inputs and outputs. The output of a task is the input of the next task.

Elicitation creates an understanding of a problem domain and its requirements. The product of the task is seldom written into a formal document. More often the information is preserved in the memory of the people involved or in the form of random elicitation notes.

In analysis the requirements and the problem domain are studied to achieve an understanding of what should be achieved by the system. In this task the holes, overlaps and contradictions in the elicited requirements are alleviated. Often the result of this task is not written down. Like the knowledge from elicitation the results of analysis are preserved in the memories of the stakeholders and requirements engineers.

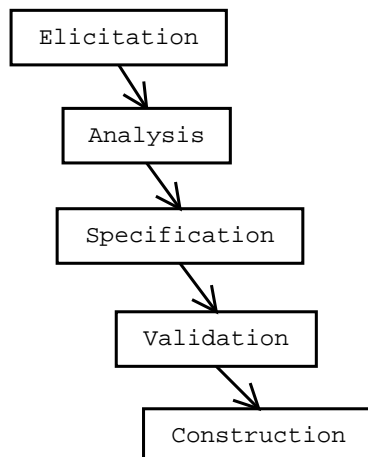


Figure 4: A simple requirements engineering process.

Specification produces a definition of a system that will produce the required effects in the problem domain. This is done by first inventing and then describing the functions of the system. The result of this task is a written specification of the behavior of the system. The names used with this document are more than various. As Bray points out the names include [Bra02]:

- requirements specification,
- system requirements specification,
- requirements definition and
- functional requirements specification.

Validation is concerned with the quality of the work done. Before actually implementing the specified system, it is sound to check for errors made in the process. As requirements engineering is an activity performed by humans and humans are known to make mistakes, the specification of a system is never error-free. The quality-related properties of requirements specification include [IEE98]:

- correctness,
- unambiguity,
- completeness,
- consistency,
- verifiability,

- modifiability and
- traceability.

In validation the failures to meet these properties are corrected.

In construction a system is constructed to meet the specification. As construction is about the inner workings of the system, it is not an activity of requirements engineering.

In Figure 5 a more complex requirements engineering process is introduced. The process is very similar to the simple process. The main difference is the addition of iteration. The process shows how the basic process of requirements engineering can be made iterative. If questions arise in any task, the previous tasks are consulted. If some questions are open in the elicitation task the stakeholders are consulted. Other addition is the separation of the requirements document and the specification document.

4 Discussion

The view presented in this report does not comprehensively reflect the current understanding of state-of-the-art in requirements engineering. Processes have evolved in all the fields of software engineering. In course of this we have also found the simplistic requirements engineering processes and methods lacking.

In many senses the metaphor of "building" software like houses is outlived as we have learned to evolve systems into maturity [Bro87]. The classical methods and processes of the last decades in the requirements engineering have also become outdated. As our ways of creating systems have evolved so have also evolved the practices of requirements engineering. The first wave of development washed in the code-centric time when everything was about writing code. Life cycles were introduced and processes were formed. The next wave was the wave of evolution that considered some of the general problems of software production such as complexity and change [SS96].

In the 90's many inadequacies in classical view of requirements engineering have been pointed out and many improvements have been proposed. In the next chapters a brief survey of the main defiances is presented. This survey is compiled from two introductory surveys [SS96, BL98] and an article by Jarke and Pohl [JP94].

Today the systems built are not one-of-a-kind one-shot products as supposed by the classical methods. As we know almost no system is unique. Therefore especially system evolution and reuse are not taken into consideration. The product of the requirements engineering cannot be something that is thrown "over the wall" to the next task in the process and then forgotten.

Other problems relate to coping with incomplete and changing requirements. The assump-

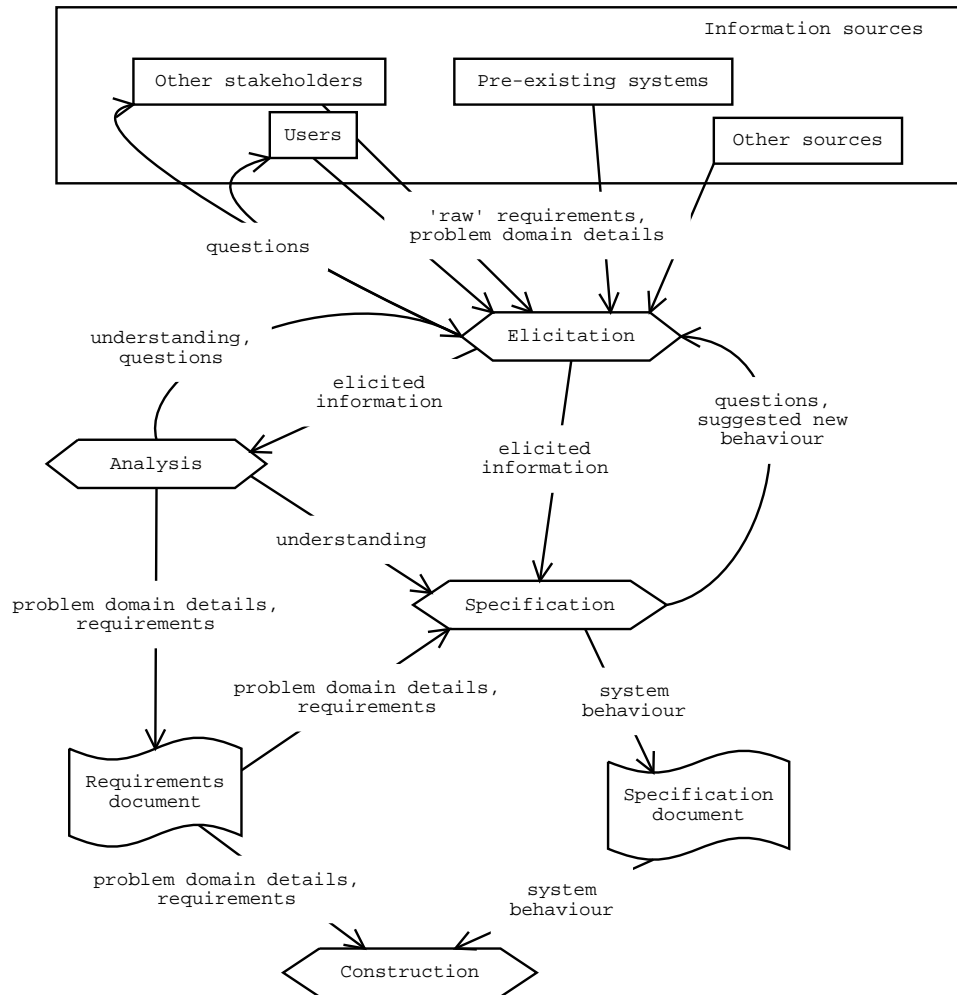


Figure 5: A more complex requirements engineering process (adapted from [Bra02]).

tion made also in this report is that there is a definable problem that needs to be solved and that the problem can be solved by a system. Experience has taught us that such an assumption is restricting as complete definition of requirements or a solution system is almost always impossible or infeasible.

Nowadays the concept of customers from whom the requirements are elicited is fading because it not possible to reach them all. How do you reach all of your customers if your product is distributed to millions of people all around the world? Today evolution of software is much more market-driven than before. This means that all the requirements are not known, all of them cannot be met nor do they need to be.

Requirements engineering has also been too focused on the technical side and the larger context has remained not researched. It is suggested that the social factors have been

forgotten too often in requirements engineering. Therefore much tacit information has not been discovered. One main reason to this is that the current elicitation and analysis methods are inadequate. The users do not know what they want. Even if they seem to know what they want they cannot express themselves, and therefore the information is partly false. All the analysis and reconciliation cannot make two pieces of false information into one truthful fact.

There has been an acknowledged gap between the theory and the practice of requirements engineering [BL98]. Actions have been taken to close the gap. Still many of the classical methods of requirements engineering that imitate and generalize former system development methods need to be replaced.

5 Conclusions

The classical requirements engineering is about forming a problem and defining a solution to it. This involves defining the problem domain, that is the part of the universe where the problem exists, and forming of a solution system, that solves the problem.

A collection of views of the problem domain is extracted from the stakeholders of the system in elicitation. The views are most often called requirements. The requirements state the required effects that the client wishes to be brought about to the problem domain. This information is then analyzed and a model of the problem domain is created. Thereafter a set of functions of a system is invented to solve the problem and to fulfill the requirements.

This classical view of requirements engineering has been found lacking in the 90's. The most critique has been toward the assumption that systems are unique and that the systems do not need to evolve. The other problem has been the technical orientation of requirements engineering. We have throughout times been so interested in the code that we have forgotten the human side of creating systems.

It is common sense to look ahead of one's nose. Requirements engineering is about knowing what the customer needs. Therefore the impact of requirements engineering to product quality cannot be emphasized too much. As the Chinese have known for ages, it is better to look before you leap. "One year's seeding is seven years' weeding."

References

- BL98 Berry, D. M. and Lawrence, B., Requirements engineering. *IEEE Software*, 15,2(1998), pages 26–29.
- Boe73 Boehm, B., Software and its impact: A quatitative assessment. *Datamation*, 19,5(1973), pages 48–59.

- Boe81 Boehm, B., *Software Engineering Economics*. Prentice-Hall, 1981.
- Bra02 Bray, I. K., *An Introduction to Requirements Engineering*. Addison-Wesley, 2002.
- Bro87 Brooks, F., No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 20,4(1987), pages 10–19.
- Dav90 Davis, A. M., *Software Requirements: Analysis & Specification*. Prentice-Hall, 1990.
- IEE90 IEEE, *IEEE Standard Glossary of Software Engineering Terminology*. Institute of Electrical and Electronics Engineers, 1990.
- IEE98 IEEE, *IEEE Recommended Practice for Software Requirements Specifications*. Institute of Electrical and Electronics Engineers, 1998.
- Jah70 Jahnukainen, M., Yrityksen informaatiojärjestelmän suunnittelun kehysmetodi (in finnish). Technical Report, The Finnish Economic Association, 1970. Kansantaloudellisia tutkimuksia XXX.
- JP94 Jarke, M. and Pohl, K., Requirements engineering in 2001: (virtually) managing a changing reality. *IEEE Software Engineering Journal*, 9,6(1994), pages 257–266.
- Ros77a Ross, D. D., Reflections on requirements, guest editorial. *TOSEM*, 13,1(1977), pages 2–5.
- Ros77b Ross, D. D., Structured analysis: A language for communicating definition. *IEEE Transactions on Software Engineering*, 13,1(1977), pages 16–34.
- Roy70 Royce, W. W., Managing the development of large software systems: Concepts and techniques. *WESCON Technical Papers, v. 14*, Los Angeles, August 1970, WESCON, pages 11–19. Reprinted in Proceedings of the Ninth International Conference on Software Engineering, 1987, pages 328–338.
- RS77 Ross, D. D. and Schoman, Jr, K. E., Structured analysis for requirements engineering. *IEEE Transactions on Software Engineering*, 13,1(1977), pages 6–15.
- SS96 Siddiqi, J. and Shekaran, M. C., Requirements engineering: The emerging wisdom. *IEEE Software*, 13,2(1996), pages 15–19.
- SS97 Sommerville, I. and Sawyer, P., *Requirements Engineering: A Good Practice Guide*. Wiley, 1997.