# USER REQUIREMENTS

*Gottfried Kellner*
CERN, Geneva, Switzerland

**Abstract**
The requirements are a collection of statements that should describe in a clear, concise, consistent and unambiguous manner all significant aspects of a proposed system. They should describe what the computer system should be capable to do and describe the environmental need which the proposed software is to satisfy. User requirements definition is an iterative process to find the widest possible agreement through interviews and surveys. The overriding objective of requirements analysis is to provide necessary and sufficient information for subsequent design, implementation, and validation & verification to be successful. Requirements form the baseline for system development, when the system is initially built, but also through subsequent maintenance and enhancement. We will discuss the problems involved in requirements gathering and analysis, the content and property of a good requirements specification, and a number of general and specific approaches to requirements analysis.

## 1. INTRODUCTION

Already since the mid-70s it is known that requirements errors are the most numerous and, more significantly, that they also are the most costly and time-consuming to correct. The recognition of the critical nature of requirements established Requirements Engineering as an important sub-field of Software Engineering. A number of research activities and conferences are devoted to this field since many years [1].

Quoting from Fred Brooks [2]:

- The hardest single part of building a software system is deciding precisely what to build.
- No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems.
- No part of the work so cripples the resulting system if done wrong.
- No other part is more difficult to rectify later.

Results of industry studies in the 1970s described by Boehm [3], and since replicated many times, showed that requirements errors are the most costly. These studies all produced the same basic result: the earlier in the development process an error occurs and the later the error is detected, the more expensive it is to correct. Moreover, the relative cost rises quickly – an error that costs one dollar to fix in the requirements phase may cost $ 20 to fix during unit tests, and $ 100-200 to fix if not corrected until the system is put into operation or in the maintenance phase.

## 2. IMPORTANCE OF (USER) REQUIREMENTS

Deciding precisely what to build and documenting the results is the goal of the requirements phase of software development. The traditional “waterfall model” [4] of the software life-cycle assumes that at the end of the user requirements phase a requirements specification exists which is unambiguous, complete, consistent, verifiable and validatable. This specification should establish and specify precisely what the software must do without describing how to do it. This rather simple-minded approach might be applicable for rather small and well-defined software projects. In most cases requirements are not fully known, or are not clearly understood. An iterative approach, including prototyping or partial design may be needed to clarify issues. A variety of software life-cycle models have been evolved which better address the inherent difficulty to arrive at requirements specifications – incremental, prototyping, RAD, spiral, 4GL, and others [4]. Even if the specification document would be perfect to start with there will always be changes due to planned upgrades or unanticipated changes to functionality or environment of the system.

Good requirements are essential if we are to be sure that we are building the system the user wants, and that we are not doing more than is needed. They should play this role not only during initial development, but also during subsequent maintenance and enhancements. In practice there are rather few systems where the requirements are good enough to be useful throughout the system's life. Requirement management is one of the key process areas to achieve this goal.

## 3. USER REQUIREMENTS

User Requirements are the primary communication vehicle between users (and customers) of a desired software system and the computer specialists to document the user's perceived needs. They describe both the system and the environment in which it will operate. They must provide necessary and sufficient information for subsequent design and implementation to be useful.

A requirement is a 'condition or capability needed by a user to solve a problem or achieve an objective' [5]. This definition leads to two principal categories of requirements: 'capability requirements' and 'constraint requirements' [6]. Capability requirements describe the process to be supported by software. They should define operations, or a sequence of related operations, that the software will be able to perform. Capability requirements should be qualified with attributes, where feasible (e.g. capacity, speed, accuracy). Constraint requirements place restrictions on how the user requirements are to be met. The user may place constraints on the software related to interface requirements (e.g. communication, hardware, software, human-computer interaction), to quality requirements (e.g. adaptability, availability, portability, security, safety, standards), or requirements on resources and time-scales for producing and operating the software.

## 4. USER REQUIREMENTS DEFINITION

A variety of methods and tools are used to elicit (i.e. capture), specify and document the user requirements. Considering the importance of user requirements rather 'low-tech' techniques are used. Requirements elicitation is still a very active field of research in computer science, cognitive psychology, social science, Artificial Intelligence, amongst others. Many papers are presented at conferences, but few of these ideas are used in real-life projects outside the study cases.

People involved in the elicitation process are users, customers, designers and software engineers, project managers, and facilitators with extensive knowledge in techniques for elicitation, analysis and design of software. The purpose is to gather a maximum of information about user needs, to capture the user's aims and objectives, and to clarify requests or statements which are ambiguous, incomplete, too detailed, etc. Methods used include interviews, surveys, studies of existing systems, feasibility studies, exploratory prototyping, joint or rapid application development procedures, identification of scenarios, use cases or patterns. Usually a combination of these techniques is applied, depending on the project, the expertise of the users to identify and express their needs, and the skills of the facilitator.

Most user requirements are specified in simple natural language. These can be complemented by diagrams, tables, mathematical formalisms, context diagrams for describing the environment, indeed anything that can help in clarification of issues. Word processors, spread-sheet and diagramming tools are mostly used. For larger software systems database management systems and requirements engineering tools will help to manage the complexity of a large number of requirements and their interdependency. These tools are particularly useful to cope with problems of requirements management over long periods of time for development and operation of software systems.

Documentation of user requirements, the User Requirements Document (URD), usually applies standard templates provided by a variety of sources. We have adopted the layout proposed by the ESA PSS-05 documents. ESA PSS-05 provides standards and guidelines for the whole software life-cycle and is widely used in industry in Europe [6]. A standard URD template using Adobe FrameMaker [7], as well as several examples of URD for HEP applications have been made available on the web [8]. A URD will normally pass many cycles of internal reviews for updates and clarifications. Acceptance tests will be identified for validation of the user requirements in the transfer phase at the end of the software development cycle. A formal review of the URD and the Acceptance Test Plan, involving users, developers, management and QA staff, will conclude by a formal statement that the project is ready to proceed. Formally, the software life-cycle begins with the acceptance of the User Requirements Document.

## 5. SUMMARY

Requirements are intrinsically hard to do well. Beyond the need for discipline, there are a number of difficulties that attend both the understanding of requirements and their specification. Technical and human concerns have to be addressed to manage complexity or communicate to different audiences. All of the approaches outlined very briefly above have significant weaknesses, but experience confirms that the use of any careful and systematic approach is preferable to an ad-hoc and chaotic one. Non-existent or inconsistent requirements will end up in poor quality software and expensive rework. Benefits of good requirements come at a cost. It needs people with adequate experience, training and resources at the begin of a project, not after disaster has struck. The bulk of software problems arise from inadequate specifications, not from errors in implementation.

## REFERENCES


[1] Software Requirements Engineering, 2nd edition, Edited by R.H.Thayer and M.Dorfman, IEEE Computer Society Press, ISBN 0-8186-7738-4, 1997.

Proceedings of the Third IEEE International Symposium on Requirements Engineering, January 6-10, 1997, Annapolis, IEEE Computer Society Press, ISBN 0-8186-7740-6, 1997.

Requirements engineering : a good practice guide, I.Sommerville, P.Sawyer, John Wiley & Sons, ISBN 0-471-97444-7, 1997.

Software Engineering (5th edition), I.Sommerville, Addison Wesley, ISBN 0-201-42765-6, 1995.

[2] “No Silver Bullet: Essence and Accidents of SoftwareEngineering”, F.Brooks, Computer, April 1987, pp. 10-19

The Mythical Man-Month, F.Brooks, Addison-Wesley, 1975

[3] Software Engineering Economics, B.Boehm, Prentice Hall, 1981

[4] Software Engineering, A Practioner's Approach, R.S.Pressman, 4th edition, McGraw-Hill, ISBN 0-070-52182-4, 1996

[5] IEEE Standard Glossary for Software Engineering Terminology, ANSI/IEEE Std 610.12-1990

[6] Software Engineering Standards, C.Mazza et al., Prentice Hall, ISBN 0-13-106568-8, 1994

Software Engineering Guides, Edited by J.Fairclough, Prentice Hall, ISBN 0-13-449281-1, 1996

[7] Learning Adobe Framemaker: The Official Guide to Adobe Framemaker. Paperback, Mac Millan, May 1 1996, ISBN: 1568302908

[8] see http://www.cern.ch/FrameMaker/#templates