

Lecture No. 4

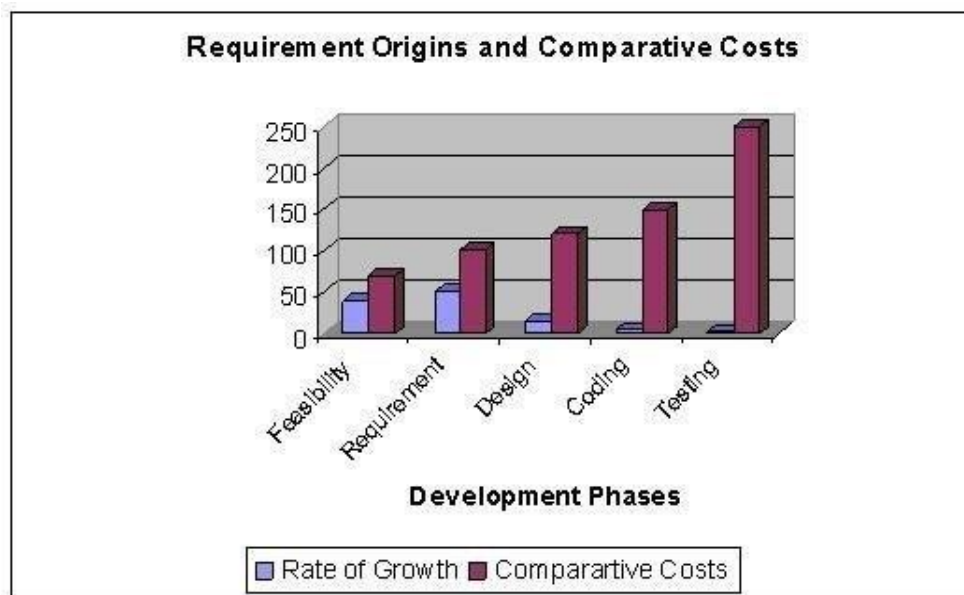
Requirement Engineering-2

3.1 Some Risks from Inadequate Requirement Process

From the above discussion, it should be clear that the requirements play the most significant role in the software development process and the success and failure of a system depends to a large extent upon the quality of the requirement documents. Following is a list of some of the risks of adopting an inadequate requirement process:

1. Insufficient user involvement leads to unacceptable products.
If input from different types of user is not taken, the output is bound to lack in key functional areas, resulting in an unacceptable product. Overlooking the needs of certain user classes (stake holders) leads to dissatisfaction of customers.
2. Creeping user requirements contribute to overruns and degrade product quality.
Requirement creep is one of the most significant factors in budget and time overruns.

It basically means identifying and adding new requirements to the list at some advanced stages of the software development process. The following figure shows the relative cost of adding requirements at different stages.



3. Ambiguous requirements lead to ill-spent time and rework.

Ambiguity means that two different readers of the same document interpret the requirement differently. Ambiguity arises from the use of natural language. Because of the imprecise nature of the language, different readers interpret the statements differently. As an example, consider the following Urdu Phrase: “*Rooko mut jane doo*”. Now, depending upon where a reader places the comma in this statement, two different readers may interpret it in totally different manner. If a comma is placed after “*Rooko*”, the sentence will become “*Rooko, mut jane doo*”, meaning “*don’t let him go*”. On the other hand if the comma is placed after “*mut*”, the sentence will become “*Rooko mut, jane doo*”, meaning “*let him go*”. Ambiguous requirements therefore result in misunderstandings and mismatched expectations, resulting in a wasted time and effort and an undesirable product.

Let us consider the following requirement statement:

The operator identity consists of the operator name and password; the password consists of six digits. It should be displayed on the security VDU and deposited in the login file when an operator logs into the system.

This is an example of ambiguous requirement as it is not clear what is meant by “*it*” in the second sentence and what should be displayed on the VDU. Does it refer to the operator identity as a whole, his name, or his password?

4. Gold-plating by developers and users adds unnecessary features.

Gold-plating refers to features are not present in the original requirement document and in fact are not important for the end-user but the developer adds them anyway thinking that they would add value to the product. Since these features are outside the initial scope of the product, adding them will result in schedule and budget overruns.

5. Minimal specifications lead to missing key requirements and hence result in an unacceptable product.

As an example, let us look at the following requirement. The requirement was stated as: “*We need a flow control and source control engineering tool.*” Based upon this requirement, system was built. It worked perfectly and had all the functionality needed for source control engineering tool and one could draw all kinds of maps and drawings. The system however could not be used because there was there was no print functionality.

Let us now look at the following set of requirement statements for another system:

- *The system should maintain the hourly level of reservoir from depth sensor situated in the reservoir. The values should be stored for the past six months.*
- *AVERAGE: Average command displays the average water level for a particular sensor between two times.*

This is another case of minimal requirements – it does not state how the system should respond if we try to calculate the average water level beyond the past six months.

6. Incompletely defined requirements make accurate project planning and tracking impossible.

Levels of Software Requirements

Software requirements are defined at various levels of detail and granularity. Requirements at different level of detail also mean to serve different purposes. We first look at these different levels and then will try to elaborate the difference between these with the help of different examples.

1. Business Requirements:

These are used to state the high-level business objective of the organization or customer requesting the system or product. They are used to document main system features and functionalities without going into their nitty-gritty details. They are captured in a document describing the project vision and scope.

2. User Requirements:

User requirements add further detail to the business requirements. They are called user requirements because they are written from a user's perspective and the focus of user requirements describe tasks the user must be able to accomplish in order to fulfill the above stated business requirements. They are captured in the requirement definition document.

3. Functional Requirements:

The next level of detail comes in the form of what is called functional requirements. They bring-in the system's view and define from the system's perspective the software functionality the developers must build into the product to enable users to accomplish their tasks stated in the user requirements - thereby satisfying the business requirements.

4. Non-Functional Requirements

In the last section we defined a software requirement as a document that describes all the services provided by the system along with the constraints under which it must operate. That is, the requirement document should not only describe the functionality needed and provided by the system, but it must also specify the constraints under which it must operate. Constraints are restrictions that are placed on the choices available to the developer for design and construction of the software product. These kinds of requirements are called Non-Functional Requirements. These are used to describe external system interfaces, design and implementation constraints, quality and performance attributes. These also include regulations, standards, and contracts to which the product must conform.

Non-functional requirements play a significant role in the development of the system. If not captured properly, the system may not fulfill some of the basic business needs. If proper care is not taken, the system may collapse. They dictate how the system architecture and framework. As an example of non-functional requirements, we can require software to run on Sun Solaris Platform. Now it is clear that if this requirement was not captured initially and the entire set of functionality was built to run on Windows, the system would be useless for the client. It can also be easily seen that this requirement would have an impact on the basic system architecture while the functionality does not change.

While writing these requirements, it must always be kept in mind that all functional requirements must derive from user requirements, which must themselves be aligned with business requirements. It must also be remembered that during the requirement engineering process we are in the definition phase of the software development where the focus is on what and not how. Therefore, requirements must not include design or implementation details and the focus should always remain on what to build and not how to build.

Let us now look at an example to understand the difference between these different types of requirements.

Let us assume that we have a word-processing system that does not have a spell checker. In order to be able to sell the product, it is determined that it must have a spell checker. Hence the business requirement could be stated as: *user will be able to correct spelling errors in a document efficiently*. Hence, the Spell checker will be included as a feature in the product.

In the next step we need to describe what tasks must be included to accomplish the above-mentioned business requirement. The resulting user requirement could be as follows: *finding spelling errors in the document and decide whether to replace each misspelled word with the one chosen from a list of suggested words*. It is important to note that this requirement is written from a user's perspective.

After documenting the user's perspective in the form of user requirements, the system's perspective: what is the functionality provided by the system and how will it help the user to accomplish these tasks. Viewed from this angle, the functional requirement for the same user requirement could be written as follows: *the spell checker will find and highlight misspelled words. It will then display a dialog box with suggested replacements. The user will be allowed to select from the list of suggested replacements. Upon selection it will replace the misspelled word with the selected word. It will also allow the user to make global replacements*.

Finally, a non-functional requirement of the system could require that *it must be integrated into the existing word-processor that runs on windows platform*.

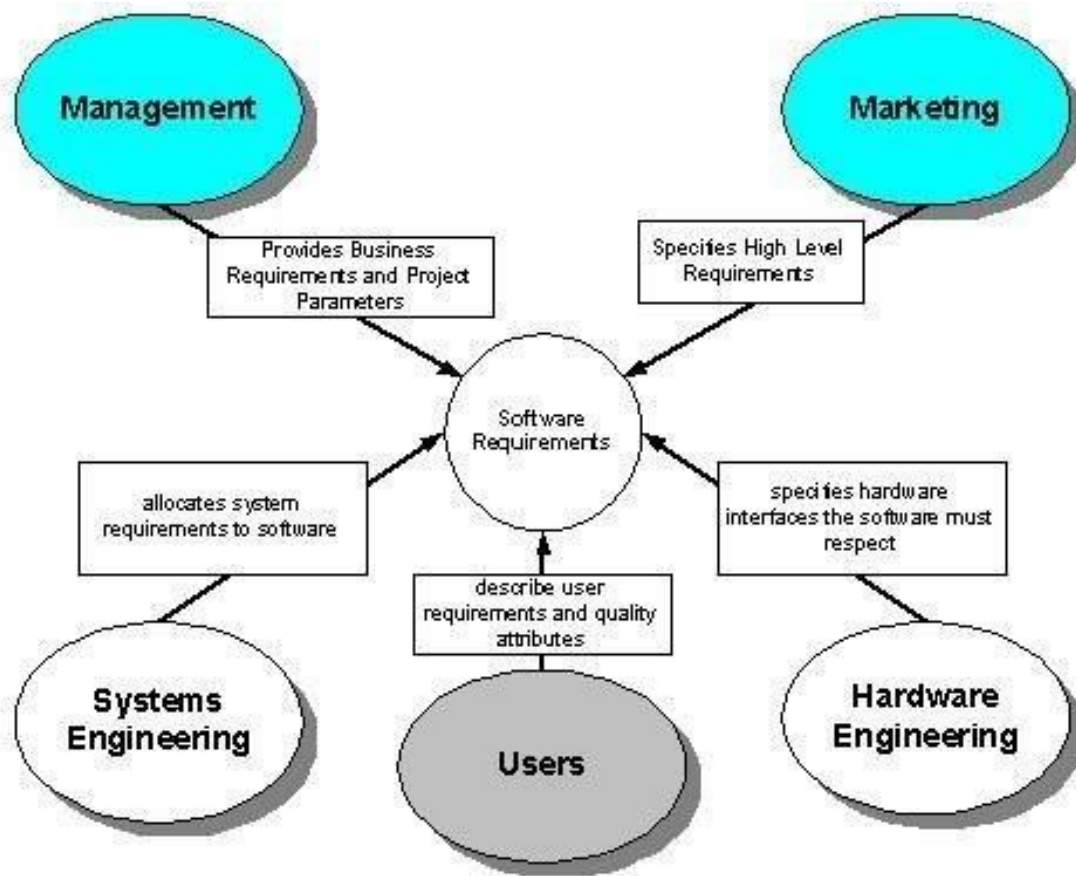
Stakeholders

As mentioned earlier, in order to develop a good requirement document, it is imperative to involve all kinds of user in the requirement engineering process. The first step in fulfillment of this need is the identification of all the stakeholders in the system. Stakeholders are different people who would be interested in the software. It is important to recognize that management carries a lot of weight, but they usually are not the actual users of the system. We need to understand that it is the actual user who will eventually use the system and hence accept or reject the product. Therefore, ignoring the needs of any user class may result in the system failure.

A requirement engineer should be cognizant of the fact that stakeholders have a tendency to state requirements in very general and vague terms. Some times they trivialize things. Different stakeholders have different requirements – sometimes even conflicting. On top of that internal politics may influence requirements.

The role of stakeholders cannot be overemphasized. A study of over 8300 projects revealed that the top two reasons for any project failure are lack of user input and incomplete requirements.

The following diagram shows the role of different stakeholders in the setting the system requirements.



Requirement Statement and Requirement Specification Documents

Different levels of software requirements are documented in different documents. The two main documents produced during this phase are Requirement Statement and Requirement Specification. They are also called Requirement Definition and Functional Specification and are used to document user requirements and functional requirements respectively.

Requirement Statement Characteristics

A good Requirements statement document must possess the following characteristics.

- **Complete** - Each requirement must fully describe the functionality to be delivered.
- **Correct** - Each requirement must accurately describe the functionality to be built.
- **Feasible** - It must be possible to implement each requirement within the known capabilities and limitations of the system and its environment.

- **Necessary** -Each requirement should document something that the customer really need or something that is required for conformance to an external system requirement or standard.
- **Prioritized** - An implementation priority must be assigned to each requirement, feature or use case to indicate how essential it is to a particular product release.
- **Unambiguous** - All readers of a requirement statement should arrive at a single, consistent interpretation of it.
- **Verifiable** – User should be able to devise a small number of tests or use other verification approaches, such as inspection or demonstration, to determine whether the requirement was properly implemented.

Requirement Specification Characteristics

A good Requirements specification document should possess the following characteristics.

- **Complete** - No requirement or necessary information should be missing.
- **Consistent** – No requirement should conflict with other software or higher-level system or business requirements.

Let us try to understand this with the help of some examples. The following set of (non-functional) requirements was stated for a particular embedded system.

- *All programs must be written in Ada*
- *The program must fit in the memory of the embedded micro-controller*

These requirements conflicted with one another because the code generated by the Ada compiler was of a large footprint that could not fit into the micro-controller memory.

Following is another set of (functional) requirements that conflicted with one another:

- *System must monitor all temperatures in a chemical reactor.*

- *System should only monitor and log temperatures below - 20° C and above 400° C.* In this case the two requirements clearly conflict with each other in stating what information needs to be monitored and stored.
- **Modifiable** - One must be able to revise the Software Requirement Specification when necessary and maintain a history of changes made to each requirement.
- **Traceable** - One should be able to link each requirement to its origin and to the design elements, source code, and test cases that implement and verify the correct implementation of the requirement.

Mixed level of Abstraction

It is important to recognize that all requirements in a requirement document are stated at a uniform level of abstraction. This difference in detail falsely implies the relative importance of these requirements and hence misguides all involved in the development process. The following set of requirements clearly demonstrates violation of this principle:

- *The purpose of the system is to track the stock in a warehouse.*
- *When a loading clerk types in the withdraw command he or she will communicate the order number, the identity of the item to be removed, and the quantity removed. The system will respond with a confirmation that the removal is allowable.*