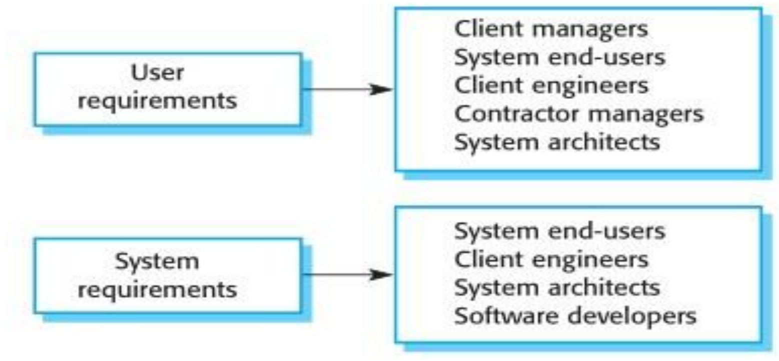
Unit-3 Requirements Engineering

The requirements for a system are the descriptions of the services that a system should provide and the constraints on its operation. These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information. The process of finding out, analyzing, documenting and checking these services and constraints is called **requirements engineering (RE).**

Some of the problems that arise during the requirements engineering process are a result of failing to make a clear separation between these different levels of description. We distinguish between them by using the term user requirements to mean the high-level abstract requirements and system requirements to mean the detailed description of what the system should do. User requirements and system requirements may be defined as follows:

1. **User requirements** are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate. The user requirements may vary from broad statements of the system features required to detailed, precise descriptions of the system functionality.
2. **System requirements** are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

You need to write requirements at different levels of detail because different types of readers use them in different ways. Figure below shows the types of readers of the user and system requirements.



The readers of the user requirements are not usually concerned with how the system will be implemented and may be managers who are not interested in the detailed facilities of the system. The readers of the system requirements need to know more precisely what the system will do because they are concerned with how it will support the business processes or because they are involved in the system implementation

Requirements engineering is usually presented as the first stage of the software engineering process. However, some understanding of the system requirements may have to be developed before a decision is made to go ahead with the procurement or development of a system. This early-stage RE establishes a high-level view of what the system might do and the benefits that it might provide. These may then be considered in a feasibility study, which tries to assess whether or not the system is technically and financially feasible. The results of that study help management decide whether or not to go ahead with the procurement or development of the system.

Functional and Non-Functional Requirements

Functional Requirements

Functional requirements define the basic system behavior. Essentially, they are **what** the system does or must not do, and can be thought of in terms of how the system responds to inputs. Functional requirements usually define if/then behaviors and include calculations, data input, and business processes.

Functional requirements are the statement of the services that the system must provide or descriptions of how some computation are carried out.

These are the statement of services the system should provide, how the system should react to particular inputs, how the system should behave in a particular situation. Functional requirements are features that allow the system to function as it was intended. If the functional requirements are not met, the system will not work. Functional requirements are product features and focus on user requirements.

Functional requirements are the main things that the user expects form the s/w for eg: if the application is banking application, that application should be able to create a new account. update account, delete an account etc.

The functional requirement for the system should be both complete and consistent. Completeness means that all services required by the user should be defined and consistency means that requirement should not have contradictory meaning.

Example of Functional Requirements

- The software automatically validates customers against the Contact Management system.
- The Sales systems should allow users to record customer's sales.
- The background color for all windows in the application will be blue and have a hexadecimal RGB color value of 0x0000FF.
- Only Managerial level employees have the right to view revenue data.
- The software system should be integrated with banking API

Non-Functional Requirements

Non-functional requirement are requirements that are the not directly concerned with specific functions delivered by the system but they concerned with emergent system properties, such as reliability, response time, security, safety etc, non-functional requirements specify how the system should do it. Non-functional requirements do not affect the basic functionality of the system (hence the name, non-functional requirements). Even if the non-functional requirements: are not met, the system will still perform its basic purpose.

Non-functional requirements place constraints on how the system will do so and elaborates a performance characteristic of the system. Non-functional requirements may also describe. aspects of the system that don't relate to its execution, but rather to it's evaluation over time. (eg maintainability, extensibility, reliability, etc.) For eg, for an banking application, a major non-functional requirement will be availability, the application should be available 24/7 with no down time if possible.

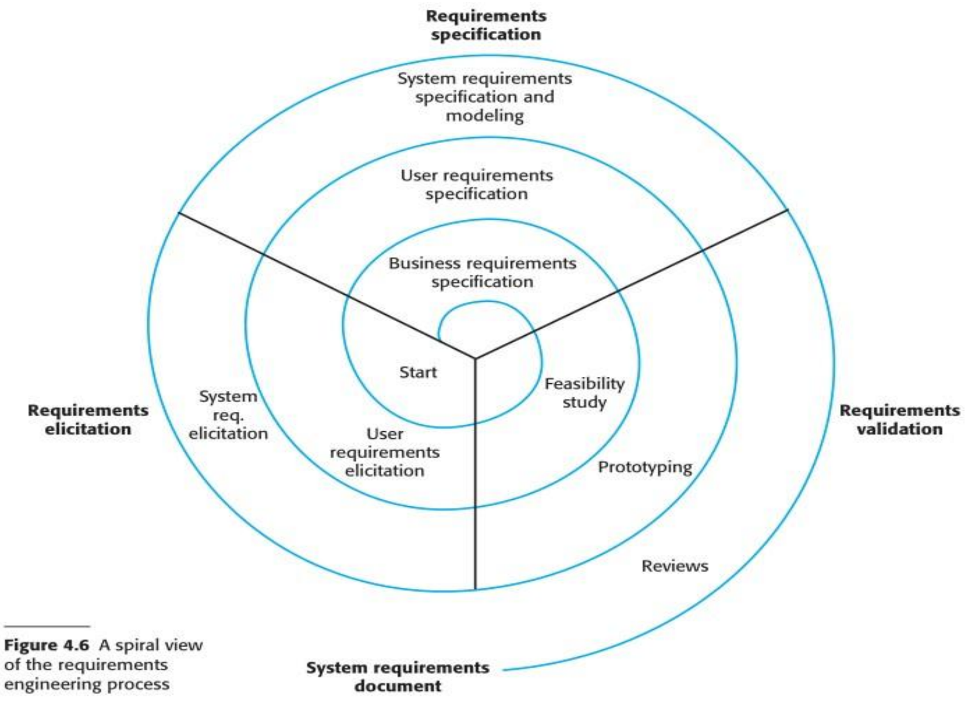
Non functional requirement may constraint the process that should be used to develop the system. These arise through user needs, because of budget constraints, organizational policies, external factors such as safety regulations of privacy legislation. These are often called qualities of system, constraints, quality goals, quality of service require.

Examples of Non-functional requirements

- Users must change the initially assigned login password immediately after the first successful login.
- Employees never allowed updating their salary information. Such attempt should be reported to the security administrator.
- Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
- The software should be portable. So moving from one OS to other OS does not create any problem.
- Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

Requirements Engineering Process

Requirements engineering involves three key activities. These are discovering requirements by interacting with stakeholders (**elicitation and analysis**); converting these requirements into a standard form (**specification**); and checking that the requirements actually define the system that the customer wants (**validation**). However, in practice, requirements engineering is an iterative process in which the activities are interleaved.



Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
Start
Feasibility study
Prototyping
Reviews
Requirements elicitation
System req. elicitation
User requirements elicitation
Requirements validation
System requirements document

Figure 4.6 A spiral view of the requirements engineering process

Above figure shows this interleaving. The activities are organized as an iterative process around a spiral. The output of the RE process is a system requirements document. The amount of time and effort devoted to each activity in iteration depends on the stage of the overall process, the type of system being developed, and the budget that is available.

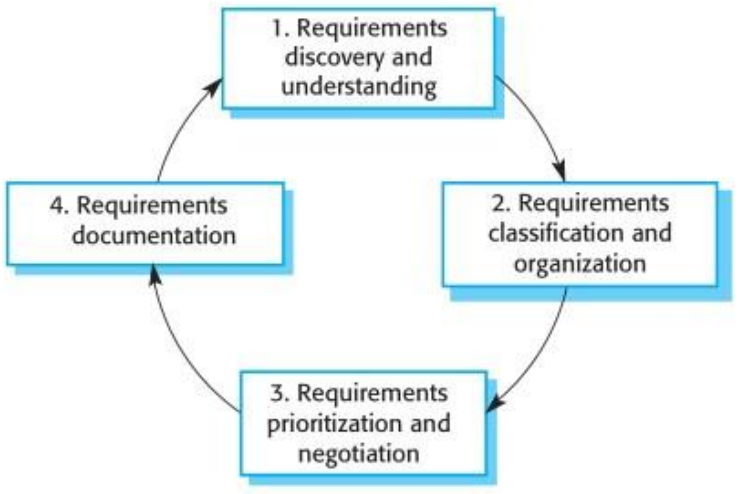
Early in the process, most effort will be spent on understanding high-level business and non-functional requirements, and the user requirements for the system. Later in the process, in the outer rings of the spiral, more effort will be devoted to eliciting and understanding the non-functional requirements and more detailed system requirements.

This spiral model accommodates approaches to development where the requirements are developed to different levels of detail. The number of iterations around the spiral can vary so that the spiral can be exited after some or all of the user requirements have been elicited. Agile development can be used instead of prototyping so that the requirements and the system implementation are developed together.

In virtually all systems, requirements change. The people involved develop a better understanding of what they want the software to do; the organization buying the system changes; and modifications are made to the system's hardware, software, and organizational environment. Changes have to be managed to understand the impact on other requirements and the cost and system implications of making the change.

Requirements Elicitation

The aims of the requirements elicitation process are to understand the work that stakeholders do and how they might use a new system to help support that work. During requirements elicitation, software engineers work with stakeholders to find out about the application domain, work activities, the services and system features that stakeholders want, the required performance of the system, hardware constraints, and so on.


1. Requirements discovery and understanding
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements documentation

A process model of the elicitation and analysis process is shown in Figure above. Each organization will have its own version or instantiation of this general model, depending on local factors such as the expertise of the staff, the type of system being developed, and the standards used. The process activities are:

1. **Requirements discovery and understanding** This is the process of interacting with stakeholders of the system to discover their requirements. Domain requirements from stakeholders and documentation are also discovered during this activity.
2. **Requirements classification and organization** This activity takes the unstructured collection of requirements, groups related requirements and organizes them into coherent clusters.
3. **Requirements prioritization and negotiation** Inevitably, when multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation. Usually, stakeholders have to meet to resolve differences and agree on compromise requirements.
4. **Requirements documentation** The requirements are documented and input into the next round of the spiral. An early draft of the software requirements documents may be produced at this stage, or the requirements may simply be maintained informally on whiteboards, wikis, or other shared spaces.

Requirements elicitation techniques

Requirements elicitation involves meeting with stakeholders of different kinds to discover information about the proposed system. You may supplement this information with knowledge of existing systems and their usage and information from documents of various kinds. You need to spend time understanding how people work, what they produce, how they use other systems, and how they may need to change to accommodate a new system.

There are two fundamental approaches to requirements elicitation:

1. Interviewing, where you talk to people about what they do.
2. Observation or ethnography, where you watch people doing their job to see what artifacts they use, how they use them, and so on.

You should use a mix of interviewing and observation to collect information and, from that, you derive the requirements, which are then the basis for further discussions.

Interviewing

Formal or informal interviews with system stakeholders are part of most requirements engineering processes. In these interviews, the requirements engineering team puts questions to stakeholders about the system that they currently use and the system to be developed. Requirements are derived from the answers to these questions.

Interviews may be of two types:

1. Closed interviews, where the stakeholder answers a predefined set of questions.
2. Open interviews, in which there is no predefined agenda.

Ethnography

Ethnography is an observational technique that can be used to understand operational processes and help derive requirements for software to support these processes. An analyst immerses himself or herself in the working environment where the system will be used. The day-to-day work is observed, and notes are made of the actual tasks in which participants are involved. The value of ethnography is that it helps discover implicit system requirements that reflect the actual ways that people work, rather than the formal processes defined by the organization.

Requirements Validation

Requirements validation is the process of checking that requirements define the system that the customer really wants. It overlaps with elicitation and analysis, as it is concerned with finding problems with the requirements. Requirements validation is critically important because errors in a requirements document can lead to extensive rework costs when these problems are discovered during development or after the system is in service.

The cost of fixing a requirements problem by making a system change is usually much greater than repairing design or coding errors. A change to the requirements usually means that the system design and implementation must also be changed. Furthermore, the system must then be retested. During the requirements validation process, different types of checks should be carried out on the requirements in the requirements document. These checks include:

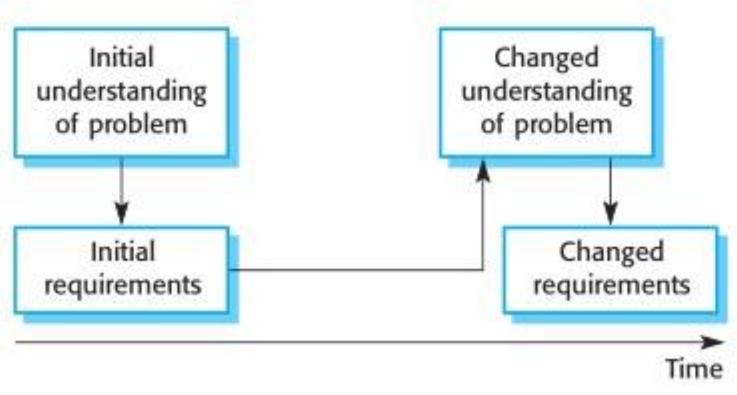
1. **Validity checks** These check that the requirements reflect the real needs of system users. Because of changing circumstances, the user requirements may have changed since they were originally elicited.
2. **Consistency checks** Requirements in the document should not conflict. That is, there should not be contradictory constraints or different descriptions of the same system function.
3. **Completeness checks** The requirements document should include requirements that define all functions and the constraints intended by the system user.
4. **Realism checks** By using knowledge of existing technologies, the requirements should be checked to ensure that they can be implemented within the proposed budget for the system. These checks should also take account of the budget and schedule for the system development.
5. **Verifiability** To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable. This means that you should be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement.

A number of requirements validation techniques can be used individually or in conjunction with one another:

1. **Requirements reviews** The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.
2. **Prototyping** This involves developing an executable model of a system and using this with end-users and customers to see if it meets their needs and expectations. Stakeholders experiment with the system and feedback requirements changes to the development team.
3. **Test-case generation** Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered. Developing tests from the user requirements before any code is written is an integral part of test-driven development

Requirements Change

The requirements for large software systems are always changing. Because the problem cannot be fully defined, the software requirements are bound to be incomplete. During the software development process, the stakeholders' understanding of the problem is constantly changing.


Initial understanding of problem
Changed understanding of problem
Initial requirements
Changed requirements
Time

The system requirements must then evolve to reflect this changed problem understanding. Once a system has been installed and is regularly used, new requirements inevitably emerge. This is partly a consequence of errors and omissions in the original requirements that have to be corrected. However, most changes to system requirements arise because of changes to the business environment of the system:

1. The business and technical environment of the system always changes after installation. New hardware may be introduced and existing hardware updated. It may be necessary to

interface the system with other systems. Business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that require system compliance.

2. The people who pay for a system and the users of that system are rarely the same people. System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements, and, after delivery, new features may have to be added for user support if the system is to meet its goals.
3. Large systems usually have a diverse stakeholder community, with stakeholders having different requirements. Their priorities may be conflicting or contradictory. The final system requirements are inevitably a compromise, and some stakeholders have to be given priority. With experience, it is often discovered that the balance of support given to different stakeholders has to be changed and the requirements re-prioritized.

As requirements are evolving, you need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You therefore need a formal process for making change proposals and linking these to system requirements. This process of “requirements management” should start as soon as a draft version of the requirements document is available.

Agile development processes have been designed to cope with requirements that change during the development process. In these processes, when a user proposes a requirements change, this change does not go through a formal change management process. Rather, the user has to prioritize that change and, if it is high priority, decide what system features that were planned for the next iteration should be dropped for the change to be implemented.

The problem with this approach is that users are not necessarily the best people to decide on whether or not a requirements change is cost-effective. In systems with multiple stakeholders, changes will benefit some stakeholders and not others. It is often better for an independent authority, who can balance the needs of all stakeholders, to decide on the changes that should be accepted.