**UNIT-II**

U**nderstanding Requirements:** Requirements Engineering, Establishing the Groundwork, Eliciting Requirements, Developing Use Cases, Building the Requirements Model, Negotiating Requirements, Validating Requirements. (Text Book 1, Chapter-5) 6Hrs,Requirements Analysis (Text Book 1, Chapter-6) 2 Hrs

## REQUIREMENTS ENGINEERING

**Requirements** analysis, also called **requirements engineering**, is the process of determining user expectations for a new or modified product. Requirements engineering is a major software engineering action that begins during the **communication activity and continues into the modeling activity.** It must be adapted to the needs of the process, the project, the product, and the people doing the work. Requirements engineering builds a bridge to design and construction.

Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system. It encompasses **seven** distinct tasks: **inception, elicitation, elaboration, negotiation, specification, validation, and management**.

**Inception:** It establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.

**Elicitation:** In this stage, proper information is extracted to prepare to document the requirements. It certainly seems simple enough—ask the customer, the users, and others what the objectives for the system or product are, what is to be accomplished, how the system or product fits into the needs of the business, and finally, how the system or product is to be used on a day- to-day basis.

* + **Problems of scope.** The boundary of the system is ill-defined or the customers/users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.
  + **Problems of understanding.** The customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don’t have a full understanding of the problem domain, have trouble communicating needs to the system engineer, omit information that is believed to be “obvious,” specify requirements that conflict with the needs of other customers/users, or specify requirements that are ambiguous or un testable.
  + **Problems of volatility.** The requirements change over time.

**Elaboration:** The information obtained from the customer during inception and elicitation is expanded and refined during elaboration. This task focuses on developing a refined requirements model that identifies various aspects of software function, behavior, and information. Elaboration is driven by the creation and refinement of user scenarios that describe **how** the end user (and other actors) will interact with the system.

**Negotiation:** To negotiate the requirements of a system to be developed, it is necessary to identify conflicts and to resolve those conflicts. You have to reconcile these conflicts through a process of negotiation. Customers, users, and other stakeholders are asked to rank requirements and then discuss conflicts in priority. Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction.

**Specification:** The term *specification* means **different things to different people**. A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.

**Validation:** The work products produced as a consequence of requirements engineering are assessed for quality during a validation step. Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product.

The primary requirements validation mechanism is the **technical review**. The review team that validates requirements includes software engineers, customers, users, and other stakeholders who examine the specification looking for errors in content or interpretation, areas where clarification may be required, missing information, inconsistencies, conflicting requirements, or unrealistic requirements.

**Requirements management.** Requirements for computer-based systems change, and the desire to change requirements persists throughout the life of the system. Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds. Many of these activities are identical to the software configuration management (SCM) techniques.

### ESTABLISHING THE GROUNDWORK

**Identifying Stakeholders**

A ***stakeholder*** is anyone who has a direct interest in or benefits from the system that is to be developed. At inception, you should create a list of people who will contribute input as requirements are elicited..

### Recognizing Multiple Viewpoints

Because many different stakeholders exist, the requirements of the system will be explored from many different points of view. The information from multiple viewpoints is collected, emerging requirements may be inconsistent or may conflict with one another.

### Working toward Collaboration

The job of a requirements engineer is to identify areas of commonality and areas of conflict or inconsistency. It is, of course, the latter category that presents a challenge. Collaboration does not necessarily mean that requirements are defined by committee. In many cases, stakeholders collaborate by providing their view of requirements, but a strong “project champion” (e.g., a business manager or a senior technologist) may make the final decision about which requirements make the cut.

### Asking the First Questions

Questions asked at the inception of the project should be “**context free**” . The first set of context- free questions focuses on the customer and other stakeholders, the overall project goals and benefits. For example, you might ask:

* Who is behind the request for this work?
* Who will use the solution?
* What will be the economic benefit of a successful solution?
* Is there another source for the solution that you need?

These questions help to identify all stakeholders who will have interest in the software to be built. In addition, the questions identify the measurable benefit of a successful implementation and possible alternatives to custom software development.

The next set of questions enables you to gain a better understanding of the problem and allows the customer to voice his or her perceptions about a solution:

* How would you characterize “good” output that would be generated by a successful solution?
* What problem(s) will this solution address?
* Can you show me (or describe) the business environment in which the solution will be used?
* Will special performance issues or constraints affect the way the solution is approached?

The final set of questions focuses on the effectiveness of the communication activity itself. Gause and Weinberg call these “**meta-questions**” and propose the following list:

* Are you the right person to answer these questions? Are your answers “official”?
* Are my questions relevant to the problem that you have?
* Am I asking too many questions?
* Can anyone else provide additional information?
* Should I be asking you anything else?

These questions will help to “**break the ice**” and initiate the communication that is essential to successful elicitation. But a question-and-answer meeting format is not an approach that has been overwhelmingly successful.

### ELICITING REQUIREMENTS

Requirements elicitation (also called *requirements gathering*) combines elements of problem solving, elaboration, negotiation, and specification

### Collaborative Requirements Gathering

Many different approaches to collaborative requirements gathering have been proposed. Each makes use of a slightly different scenario, but all apply some variation on the following basic guidelines:

* Meetings are conducted and attended by both software engineers and other stakeholders.
* Rules for preparation and participation are established.
* An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
* A “facilitator” (can be a customer, a developer, or an outsider) controls the meeting.
* A “definition mechanism” (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere that is conducive to the accomplishment of the goal.

During inception basic questions and answers establish the scope of the problem and the overall perception of a solution. Out of these initial meetings, the developer and customers write a **one- or two-page “product request**.”

A meeting place, time, and date are selected; a facilitator is chosen; and attendees from the software team and other stakeholder organizations are invited to participate. The product request is distributed to all attendees before the meeting date.

While reviewing the product request in the days before the meeting, each attendee is asked to make a list of objects that are part of the environment that surrounds the system, other objects that are to be produced by the system, and objects that are used by the system to perform its functions. In addition, each attendee is asked to make another list of services that manipulate or interact with the objects. Finally, lists of constraints (e.g., cost, size, business rules) and performance criteria (e.g., speed, accuracy) are also developed. The attendees are informed that the lists are not expected to be exhaustive but are expected to reflect each person’s perception of the system.

The lists of objects can be pinned to the walls of the room using large sheets of paper, stuck to the walls using adhesive-backed sheets, or written on a wall board. After individual lists are presented in one topic area, the group creates a combined list by eliminating redundant entries, adding any new ideas that come up during the discussion, but not deleting anything.

### Quality Function Deployment

*Quality function deployment* (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software. QFD “**concentrates on maximizing customer satisfaction from the software engineering process**”. To accomplish this, QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.

QFD identifies **three** types of requirements:

* **Normal requirements.** The objectives and goals that are stated for a product or system during meetings with the customer. If these requirements are present, the customer is satisfied. Examples of normal requirements might be requested types of graphical displays, specific system functions, and defined levels of performance.
* **Expected requirements.** These requirements are implicit to the product or system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction.
* **Exciting requirements.** These features go beyond the customer’s expectations and prove to be very satisfying when present.

Although QFD concepts can be applied across the entire software process, QFD uses customer interviews and observation, surveys, and examination of historical data as raw data for the requirements gathering activity. These data are then translated into a table of requirements— called the ***customer voice table***—that is reviewed with the customer and other stakeholders.

### Usage Scenarios

As requirements are gathered, an overall vision of system functions and features begins to materialize. However, it is difficult to move into more technical software engineering activities until you understand how these functions and features will be used by different classes of end users. To accomplish this, developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed. The scenarios, often called ***use cases***, provide a description of how the system will be used.

### Elicitation Work Products

The work products produced as a consequence of requirements elicitation will vary depending on the size of the system or product to be built. For most systems, the work products include

* A statement of need and feasibility.
* A bounded statement of scope for the system or product.
* A list of customers, users, and other stakeholders who participated in requirements elicitation.
* A description of the system’s technical environment.
* A list of requirements and the domain constraints that apply to each.
* A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
* Any prototypes developed to better define requirements.

Each of these work products is reviewed by all people who have participated in requirements elicitation.

### DEVELOPING USE CASES

Use cases are defined from an actor’s point of view. An actor is a role that people (users) or devices play as they interact with the software.

The first step in writing a use case is to define the set of “actors” that will be involved in the story. *Actors* are the different people (or devices) that use the system or product within the context of the function and behavior that is to be described.

Actors represent the roles that people (or devices) play as the system operates. Defined somewhat more formally, an actor is anything that communicates with the system or product and that is external to the system itself. Every actor has one or more goals when using the system. It is important to note that an actor and an end user are not necessarily the same thing. A typical user may play a number of different roles when using a system, whereas an actor represents a class of external entities (often, but not always, people) that play just one role in the context of the use case. Different people may play the role of each actor.

Because requirements elicitation is an evolutionary activity, not all actors are identified during the first iteration. It is possible to identify **primary actors** during the first iteration and **secondary actors** as more is learned about the system.

*Primary actors* interact to achieve required system function and derive the intended benefit from the system. *Secondary actors* support the system so that primary actors can do their work. Once actors have been identified, use cases can be developed.

Jacobson suggests a number of questions that should be answered by a use case:

* Who is the primary actor, the secondary actor(s)?
* What are the actor’s goals?
* What preconditions should exist before the story begins?
* What main tasks or functions are performed by the actor?
* What exceptions might be considered as the story is described?
* What variations in the actor’s interaction are possible?
* What system information will the actor acquire, produce, or change?
* Will the actor have to inform the system about changes in the external environment?
* What information does the actor desire from the system?
* Does the actor wish to be informed about unexpected changes?

The basic use case presents a high-level story that describes the interaction between the actor and the system.

### BUILDING THE REQUIREMENTS MODEL

The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system. The model changes dynamically as you learn more about the system to be built, and other stakeholders understand more about what they really require.

### Elements of the Requirements Model

The specific elements of the requirements model are dictated by the analysis modeling method that is to be used. However, a set of generic elements is common to most requirements models.

* **Scenario-based elements.** The system is described from the user’s point of view using a scenario-based approach.
* **Class-based elements.** Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system. These objects are categorized into classes a collection of things that have similar attributes and common behaviors.
* **Behavioral elements.** The behavior of a computer-based system can have a profound effect on the design that is chosen and the implementation approach that is applied. Therefore, the requirements model must provide modeling elements that depict behavior.
* **Flow-oriented elements.** Information is transformed as it flows through a computer- based system. The system accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms.

1. Identification of the system or subsystem’s key stakeholders.

### Determination of the stakeholders’ “win conditions.”

1. Negotiation of the stakeholders’ win conditions to reconcile them into a set of win-win conditions for all concerned.

### Analysis Patterns

*Analysis patterns* suggest solutions (e.g., a class, a function, a behavior) within the application domain that can be reused when modeling many applications.

Geyer-Schulz and Hahsler suggest two benefits that can be associated with the use of analysis patterns:

**First**, analysis patterns speed up the development of abstract analysis models that capture the main requirements of the concrete problem by providing reusable analysis models with examples as well as a description of advantages and limitations.

**Second**, analysis patterns facilitate the transformation of the analysis model into a design model by suggesting design patterns and reliable solutions for common problems.

Analysis patterns are integrated into the analysis model by reference to the pattern name.

### NEGOTIATING REQUIREMENTS

The intent of negotiation is to develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team. The best negotiations strive for a “**win-win**” result. That is, stakeholders win by getting the system or product that satisfies the majority of their needs and you win by working to realistic and achievable budgets and deadlines.

Boehm defines a set of negotiation activities at the beginning of each software process iteration. Rather than a single customer communication activity, the following activities are defined: Successful completion of these initial steps achieves a win-win result, which becomes the key criterion for proceeding to subsequent software engineering activities.

### VALIDATING REQUIREMENTS

As each element of the requirements model is created, it is examined for inconsistency, omissions, and ambiguity. The requirements represented by the model are prioritized by the stakeholders and grouped within requirements packages that will be implemented as software increments.

A review of the requirements model addresses the following questions:

* Is each requirement consistent with the overall objectives for the system/product?
* Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
* Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
* Is each requirement bounded and unambiguous?
* Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
* Do any requirements conflict with other requirements?
* Is each requirement achievable in the technical environment that will house the system or product?
* Is each requirement testable, once implemented?
* Does the requirements model properly reflect the information, function, and behavior of the system to be built?
* Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system?
* Have requirements patterns been used to simplify the requirements model?
* Have all patterns been properly validated? Are all patterns consistent with customer requirements?

These and other questions should be asked and answered to ensure that the requirements model is an accurate reflection of stakeholder needs and that it provides a solid foundation for design.

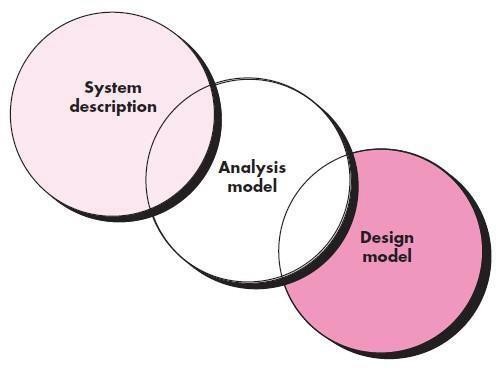
### REQUIREMENTS ANALYSIS

Requirements analysis results in the specification of software’s operational characteristics, indicates software’s interface with other system elements, and establishes constraints that software must meet. Requirements analysis allows you to elaborate on basic requirements established during the inception, elicitation, and negotiation tasks that are part of requirements engineering.

The requirements modeling action results in one or more of the following types of models:

* ***Scenario-based models*** of requirements from the point of view of various system “**actors**”
* ***Data models*** that depict the information domain for the problem
* ***Class-oriented models*** that represent object-oriented classes (attributes and operations) and the manner in which classes collaborate to achieve system requirements
* ***Flow-oriented models*** that represent the functional elements of the system and how they transform data as it moves through the system
* ***Behavioral models*** that depict how the software behaves as a consequence of external “events” These models provide a software designer with information that can be translated to architectural, interface, and component-level designs. Finally, the requirements model provides the developer and the customer with the means to assess quality once software is built.

Throughout requirements modeling, primary focus is on ***what,* not *how****.* What user interaction occurs in a particular circumstance, what objects does the system manipulate, what functions must the system perform, what behaviors does the system exhibit, what interfaces are defined, and what constraints apply?



### The requirements model as a bridge between the system description and the design model Overall Objectives and Philosophy

The requirements model must achieve three primary objectives:

1. To describe what the customer requires,
2. To establish a basis for the creation of a software design, and
3. To define a set of requirements that can be validated once the software is built.

The analysis model bridges the gap between a system-level description that describes overall system or business functionality as it is achieved by applying software, hardware, data, human, and other system elements and a software design that describes the software’s application architecture, user interface, and component-level structure.

### Analysis Rules of Thumb

Arlow and Neustadt suggest a number of worthwhile rules of thumb that should be followed when creating the analysis model:

* ***The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.***
* ***Each element of the requirements model should add to an overall understanding of software requirements and provide insight into the information domain, function, and behavior of the system.***
* ***Delay consideration of infrastructure and other nonfunctional models until design****.* That is, a database may be required, but the classes necessary to implement it, the functions required to access it, and the behavior that will be exhibited as it is used should be considered only after problem domain analysis has been completed.
* ***Minimize coupling throughout the system****.* It is important to represent relationships between classes and functions. However, if the level of “interconnectedness” is extremely high, effort should be made to reduce it.
* ***Be certain that the requirements model provides value to all stakeholders****.* Each constituency has its own use for the model
* ***Keep the model as simple as it can be****.* Don’t create additional diagrams when they add no new information. Don’t use complex notational forms, when a simple list will do.

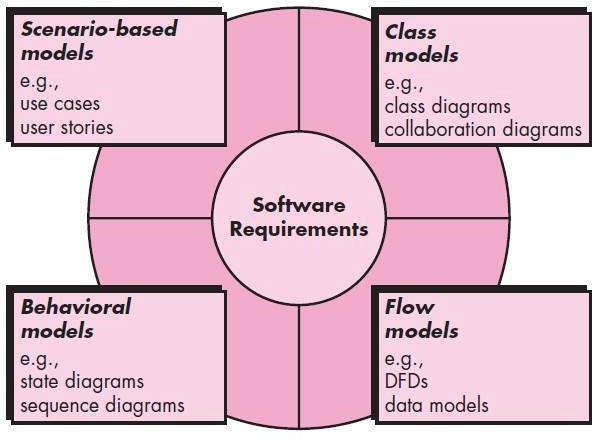
### Domain Analysis

Domain analysis doesn’t look at a specific application, but rather at the domain in which the application resides.

The “specific application domain” can range from avionics to banking, from multimedia video games to software embedded within medical devices. The goal of domain analysis is straightforward: to identify common problem solving elements that are applicable to all applications within the domain, to find or create those analysis classes and/or analysis patterns that are broadly applicable so that they may be reused.

### Requirements Modeling Approaches

One view of requirements modeling, called ***structured analysis****,* considers data and the processes that transform the data as separate entities. Data objects are modeled in a way that defines their *attributes and relationships*.

A second approach to analysis modeling, called ***object-oriented analysis****,* focuses on the definition of classes and the manner in which they collaborate with one another to effect customer requirements. UML and the Unified Process are predominantly object oriented.

### Elements of the analysis model

Each element of the requirements model is represented in following figure presents the problem from a different point of view.

**Scenario-based elements** depict how the user interacts with the system and the specific sequence of activities that occur as the software is used.

**Class-based elements model** the objects that the system will manipulate, the operations that will be applied to the objects to effect the manipulation, relationships between the objects, and the collaborations that occur between the classes that are defined.

**Behavioral elements** depict how external events change the state of the system or the classes that reside within it. Finally,

**Flow-oriented elements** represent the system as an information transform, depicting how data objects are transformed as they flow through various system functions.