Unit-4
Requirement analysis and specification

Requirements Engineering

- **Requirement**
  - A function that the system must provide to fill the needs of the system's intended user.
- **Engineering**
  - Implies that systematic and repeatable techniques should be used.
- **Requirement Engineering**
  - It is a systematic approach to define, manage and test requirements for a software.

Requirements Engineering Tasks

- Requirements Engineering encompasses seven distinct tasks:

1. **Inception (initiation)**

   Establish a basic understanding of the problem and the nature of the solution.

2. **Elicitation (to gather)**

   Draw out the requirements from stakeholders.

3. **Elaboration**

   Create ***an analysis model*** that represents information, functional, and behavioral aspects of the requirements.

4. **Negotiation**

   Agree on a deliverable system that is realistic for developers and customers.

Requirements Engineering Tasks

5. **Specification**

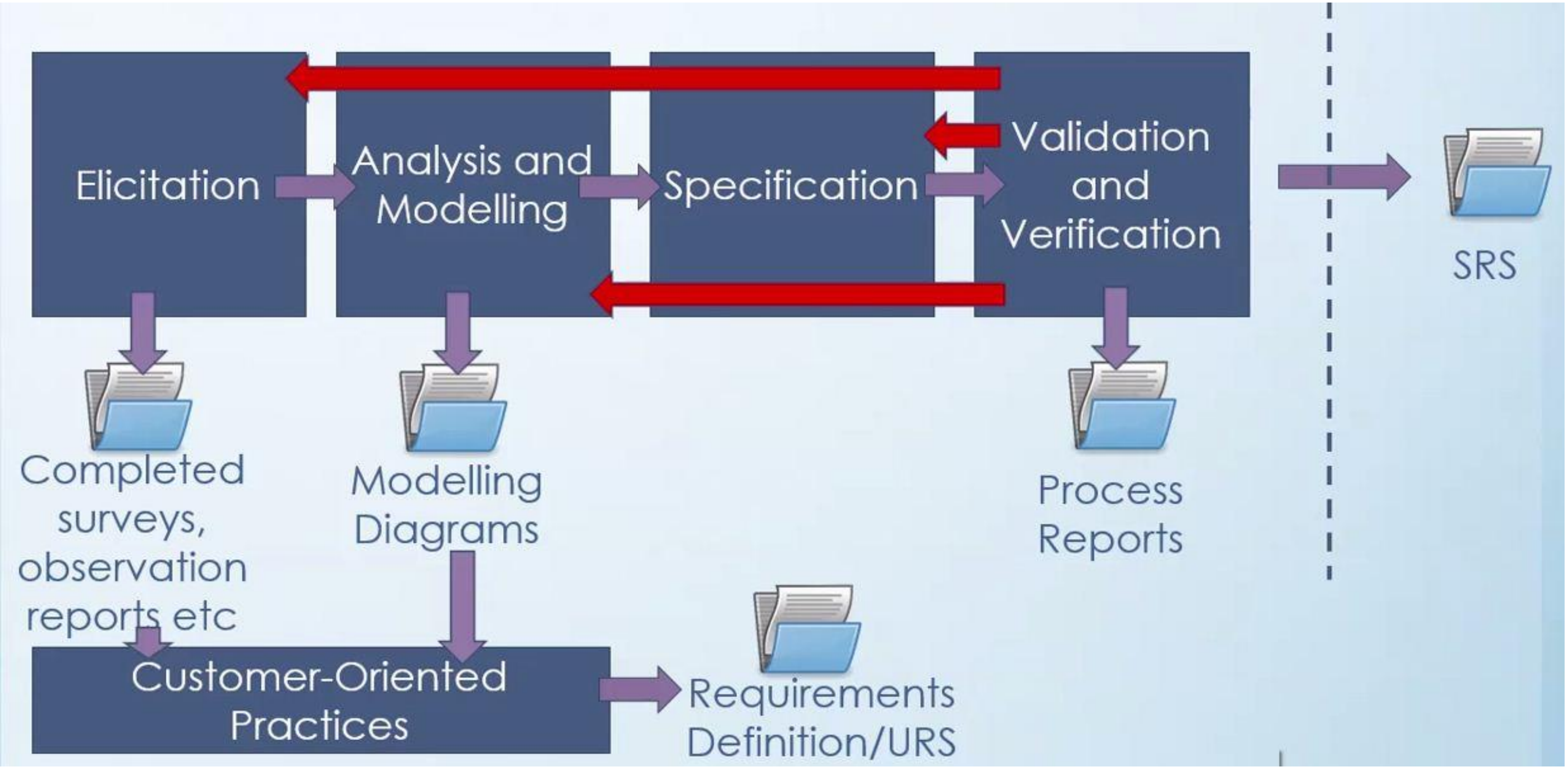
   Describe the requirements formally or informally.

6. **Validation**

   Review the requirement specification for errors, ambiguities, omissions and conflicts.

7. **Requirements Management**

   Manage changing requirements.


Elicitation
Analysis and Modelling
Specification
Validation and Verification
SRS
Completed surveys, observation reports etc
Modelling Diagrams
Process Reports
Customer-Oriented Practices
Requirements Definition/URS

Requirements Elicitation

1. Facility Application Specification Technique (FAST)

2. Quality Function Deployment (QFD)

Facility Application Specification Technique (FAST)

- It is an approach in which joint team of customers and developers work together to identify the requirements.
- Guideline for FAST approach:
  - 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 FAST team is composed of representatives from marketing, software and hardware engineering, and manufacturing.
- **Initial meetings** between the developer and customer occur and basic questions and answers help to establish the scope of the problem and the overall perception of a solution.
- Out of these initial meetings,
  - the developer and customer write a one- or two-page "product request.“
  - meeting place, time, and date for FAST are selected
  - a facilitator is chosen.
  - The product request is distributed to all attendees before the meeting date.

- In the days **before the meeting**, each FAST attendee is asked to make a list of

  1. *Objects*
     - that are part of the environment that ***surrounds the system***
     - that are ***to be produced*** by the system
     - that are ***used by the system*** to perform its functions
  2. *Services (processes or functions)*
     - *that manipulate or interact with the objects*
  3. *Constraints*

     e.g., cost, size, business rules
  4. *Performance Criteria*

     e.g., speed, accuracy

Example: Safe Home System

- **Objects:** smoke detectors, window and door sensors, motion detectors, an alarm, an event (a sensor has been activated), a control panel, a display, telephone numbers, a telephone call, and so on.
- **Services:** setting the alarm, monitoring the sensors, dialing the phone, programming the control panel, reading the display
- **Constraints:** the system must have a manufactured cost of less than $80, must be user-friendly, must interface directly to a standard phone line
- **Performance Criteria:** a sensor event should be recognized within one second, an event priority scheme should be implemented

- **During FAST meeting,**
- the first topic of discussion is the need and ***justification for the new product*****—** everyone should agree that the product is justified.
- Then each participant presents his ***lists*** for discussion.
- After individual lists are presented in one topic area, ***a combined list*** is created by the group. The combined list eliminates redundant entries, adds any new ideas that come up during the discussion, but does not delete anything.
- Then ***a consensus list*** *in each topic* area (objects, services, constraints, and performance) is developed.
- Then team is divided into smalle***r subteams;***

- Each subteam works to develop ***mini-specifications*** for one or more entries on each of the lists and then presents each of its mini-specs to all FAST attendees for discussion. Additions, deletions, and further elaboration are made.
- In some cases, the development of mini-specs will uncover new objects, services, constraints, or performance requirements that will be added to the original lists.
- After the mini-specs are completed, each FAST attendee makes a list of ***validation criteria*** for the product and presents his or her list to the team.
- ***A consensus list*** of validation criteria is then created.
- Finally, one or more participants is assigned the task of writing the complete draft specification using all inputs from the FAST meeting.

Example: Safe Home System

The mini-specification for the *SafeHome object control* panel might be

- mounted on wall
- size approximately contains standard 12-key pad and special keys
- contains LCD display
- all customer interaction occurs through keys
- used to enable and disable the system
- software provides interaction guidance
- connected to all sensors

Quality Function Deployment (QFD)

- This is a technique that **translates the needs of the customer into technical requirements** for software.

- It emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process through functions, information, and tasks.

- It identifies three types of requirements

1. **Normal requirements**
   - These requirements are the objectives and goals stated for a product during meetings with the customer.
   - Examples: types of graphical displays, specific system functions
2. **Expected requirements**
   - These requirements are implicit to the product and may be so fundamental that the customer does not explicitly state them.
   - Examples: ease of software installation

3. **Exciting requirements**

    - These requirements are for features that go beyond the customer's expectations and prove to be very satisfying when present.
    - **Example:** word processing software is requested with standard features. The delivered product contains a number of page layout capabilities that are quite pleasing and unexpected.

- QFD uses customer interviews and observation, surveys, and examination of historical data (e.g., problem reports) 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.
- A variety of diagrams, matrices, and evaluation methods are then used to extract expected requirements and to attempt to derive exciting requirements

Usage Scenarios

- As requirements are gathered as an part of informal meetings, software engineer can create a set of scenarios that identify a thread of usage for the system to be constructed. **The scenarios, often called use cases.**
- To create a use-case, the analyst must first identify the different types of people (or devices) that use the system or product. They are called **actors.**
- An actor is anything that communicates with the system and that is external to the system itself.
- Once actors have been identified, use-cases can be developed. The use-case describes the manner in which an actor interacts with the system.

Example: Safe Home System

We can define **three actors** for this system:

1. Homeowner (the user)
2. Sensors (devices attached to the system)
3. Monitoring & response subsystem (the central station that monitors *SafeHome*).

- The homeowner interacts with the product in a number of different ways:
    - enters a password to allow all other interactions
    - inquires about the status of a sensor
    - presses the panic button in an emergency
    - activates/deactivates the security system

Requirement Elaboration

Requirements Analysis Model

- The intent of this model is to provide a description of the required informational, functional, and behavioral domains for a system.
- ***The analysis model is a snapshot of requirements at any given time.***
- Elements of the Requirements Model:
  - **Scenario-based elements**
    - Describe the system from the user's point of view using scenarios that are stated in use cases and activity diagrams.
  - **Class-based elements**
    - Identify the domain classes for the objects manipulated by the actors, the attributes of these classes, and how they interact with one another; which utilize class diagrams to do this.

Requirements Analysis Model (Cont…)

- **Behavioural elements**
  - Use state diagrams to represent the state of the system, the events that cause the system to change state, and the actions that are taken as a result of a particular event.
- **Flow-oriented elements**
  - Use data flow diagrams to show the input data that comes into a system, what functions are applied to that data to do transformations, and what resulting output data are produced.

Requirement Negotiation

Negotiating Requirements

- Agree on a deliverable system that is realistic for developers and customers.

- Activities in negotiation are:
  - ***Identify the key stakeholders***

    These are the people who will be involved in the negotiation.
  - ***Determine each of the stakeholders "win conditions"***

    Win conditions are not always obvious.
  - ***Negotiate***

    Work toward a set of requirements that lead to "win-win".

Requirement Specification

Functional and non-functional requirements

- **Functional Requirements**
  - Statements of services the system should provide.
  - How the system should react to particular inputs

- **Non-functional Requirements**
  - Constraints on the services or functions offered by the system, such as timing constraints, constraints on the development process, standards, etc.

- **Domain Requirements**
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain.

Non-functional Requirements

- **Security**

  Ex– Log in, OTP, Authenticator etc.

- **Performance Requirements**

  Ex- Response Time, User interface, capacity

- **Maintainability**

  Ex- Back up, errors/crash reports

- **Reliability**

  Ex- **Availability**

Software Requirements Specification

- Software Requirement Specification (SRS) is a document that completely describes ***what the proposed software should do without describing how software will do it.***
- **It contains:**
  - a complete information description
  - a detailed functional description
  - a representation of system behaviour
  - an indication of performance requirements and design constraints
  - appropriate validation criteria
  - other information suitable to requirements
- SRS is also helping the clients to understand their own needs.

Characteristics of a Good SRS

- SRS should be accurate, complete, efficient, and of high quality
- An SRS is said to be of high quality when the developer and user easily understand the prepared document.
- Characteristics of a Good SRS:
  - **Correct**
    - SRS is correct when *all user requirements are stated* in the SRS.
    - Note that there is no specified tool or procedure to assure the correctness of SRS.
  - **Unambiguous**
    - SRS is unambiguous when every stated requirement has *only one interpretation.*

Characteristics of a Good SRS (Cont…

- **Complete**
  - SRS is complete when the *requirements clearly define what the software is required to do.*
- **Ranked for Importance & Stability**
  - All requirements are not equally important, hence each requirement is identified to make differences among other requirements.
  - SRS should be stable. Stability implies the *probability of changes in the requirement* in future.
- **Modifiable**
  - The requirements of the user can change, hence requirements document should be created in such a manner that those *changes can be modified easily.*

Characteristics of a Good SRS (Cont…)

- **Traceable**
  - SRS is traceable when the *source of each requirement is clear*
- **Verifiable**
  - SRS is verifiable when the *specified requirements can be verified with a cost-effective process* to check whether the final software meets those requirements.
- **Consistent**
  - SRS is consistent when the subsets of individual requirements defined *do not conflict with each other.*

Standard Template for writing SRS

- **Front Page**

  Software Requirements Specification

  for

  <Project>

  Version <no.>

  Prepared by <author>

  <organization>

  <date created>

- **Table of Contents**
- **Revision History**

Standard Template for writing SRS (Cont…

1. Introduction

1.1 Purpose

1.2 Document Conventions

1.3 Intended Audience and Reading Suggestions

1.4 Project Scope

1.5 References

2. Overall Description

2.1 Product Perspective

2.2 Product Features

2.3 User Classes and Characteristics

2.4 Operating Environment

2.5 Design and Implementation Constraints

Standard Template for writing SRS (Cont…

2.6 User Documentation

2.7 Assumptions and Dependencies

3. System Features

3.1 System Feature 1

3.2 System Feature 2 (and so on)

4. External Interface Requirements

4.1 User Interfaces

4.2 Hardware Interfaces

4.3 Software Interfaces

4.4 Communications Interfaces

Standard Template for writing SRS (Cont…

5. Other Nonfunctional Requirements

- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements
- 5.4 Software Quality Attributes

6. Other Requirements

Appendix A: Glossary

Appendix B: Analysis Models

Appendix C: Issues List

Problems Without SRS

- Without developing the SRS document,
  - the system would not be properly implemented according to ***customer needs***.
  - ***Software developers*** would not know whether what they are developing is what exactly is required by the customer.
  - ***Maintenance engineers*** would find it difficult to understand the functionality of the system.
  - ***User document writers*** would find it difficult to write the users' manuals properly without understanding the SRS.