SOFTWARE ENGINEERING

(CSE3005)

7/18/2019
Prof. Anand Motwani
Faculty, SCSE
VIT Bhopal University

Unit II

REQUIREMENT ENGINEERING

Understanding Requirements – Establish Ground Work – Eliciting Requirements - Developing Use case- Negotiating Requirements- Validating Requirements - Requirements Modeling – Requirement Analysis –Scenario Based Modeling - UML Supplements – Data Modeling Concepts- Class Based Modeling - Flow Oriented Modeling – Creating Behavioral Modeling - Patterns For Modeling.

Aims and Objectives

- To understand of software requirements specification.
- To understand Functional and Non-Functional requirements of Software.

1. Introduction

Software Development Process: A Brief Overview

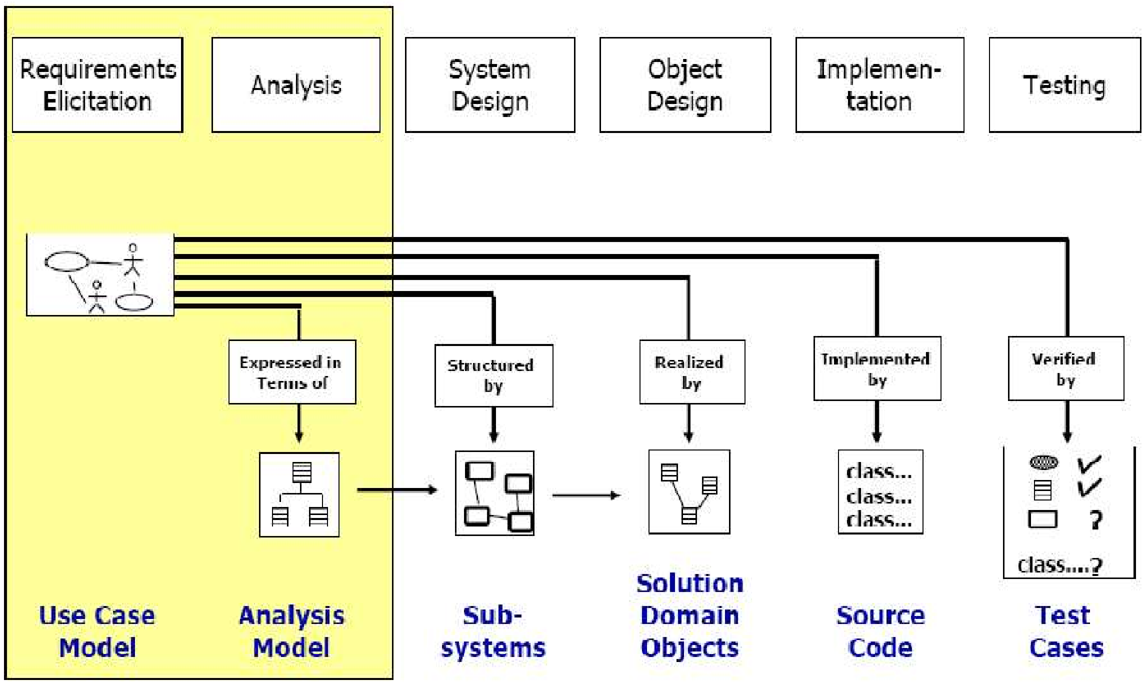


Fig. 1. Software Development Process

Understanding Requirements

"It Depends On Common Sense"

- *Software requirements* are documentation that completely describes the behavior that is required of the software-before the software is designed built and tested.
  - Requirements analysts (or business analysts) build software requirements specifications through *requirements elicitation*.
    - Interviews with the users, stakeholders and anyone else whose perspective needs to be taken into account during the design, development and testing of the software
    - Observation of the users at work
    - Distribution of discussion summaries to verify the data gathered in interviews

Discussion summary

- A requirements analyst can use a *discussion summary* to summarize information gathered during elicitation and validate it through a review.
- Notes gathered during the elicitation should fit into the discussion summary template
- The discussion summary outline can serve as a guide for a novice requirements analyst in leading interviews and meetings

Discussion Summary outline

- ✓ Project background
- ✓ Purpose of project
- ✓ Scope of project
- ✓ Other background information
- ✓ Perspectives
- ✓ Who will use the system?
- ✓ Who can provide input about the system?
- ✓ Project Objectives
- ✓ Known business rules
- ✓ System information and/or diagrams
- ✓ Assumptions and dependencies

- ✓ Design and implementation constraints
- ✓ Risks
- ✓ Known future enhancements
- ✓ References
- ✓ Open, unresolved or 'to be determined' (TBD) issues

Requirements Analysis and Specification

Analysis is the process of specifying *what* a system will do.

- – The intention is to provide a clear understanding of what the system is about and what its underlying concepts are.

The result of analysis is a *specification document*.

Requirements Engineering

- Elicitation — determining what the customer requires
- Analysis & negotiation — understanding the relationships among various customer requirements and shaping those relationships to achieve a successful result
- Requirements specification — building a tangible model of requirements
- System Modeling — building a representation of requirements that can be assessed for correctness, completeness, and consistency
- Validation — reviewing the model
- Management — identify, control and track requirements and the changes that will be made to them

Functional and Non-functional requirements

- ***Functional requirements*** define the outward behavior required of the software project.
  - – The goal of the requirement is to communicate the needed behavior in as clear and unambiguous a manner as possible.
  - – The behavior in the requirement can contain lists, bullets, equations, pictures, references to external documents, and any other material that will help the reader understand what needs to be implemented.
- ***Nonfunctional requirements*** define characteristics of the software which do not change its behavior.
  - – Users have implicit expectations about how well the software will work.

- These characteristics include how easy the software is to use, how quickly it executes, how reliable it is, and how well it behaves when unexpected conditions arise.
- The nonfunctional requirements define these aspects about the system.
    - The nonfunctional requirements are sometimes referred to as "non-behavioral requirements" or "software quality attributes"

More Explanation on Functional and Non-functional requirements

These can take a number of forms such as traditional, basic form: textual, according to some template or standard form. It includes precise statement of one function the system carries out

Example: "The system must validate the patron ID number against the patron-database before allowing the patron to check out a book."

Product requirements specify the desired characteristics that a system or subsystem must possess.

☐ Requirements for critical systems are concerned with specifying constraints on the behaviour of the executing system. Such NFRs limit the freedom of the system designers and off-the-shelf-components. Some of the components can be formulated precisely and easily quantified with Performance and Reliability. Some are difficult to quantify in terms of Usability, Adaptability.

Non-functional (product) requirements play an important role for critical systems. Failure of such systems whose causes significant economic, physical or human damage to organizations or people. So here NFRs are discussed.

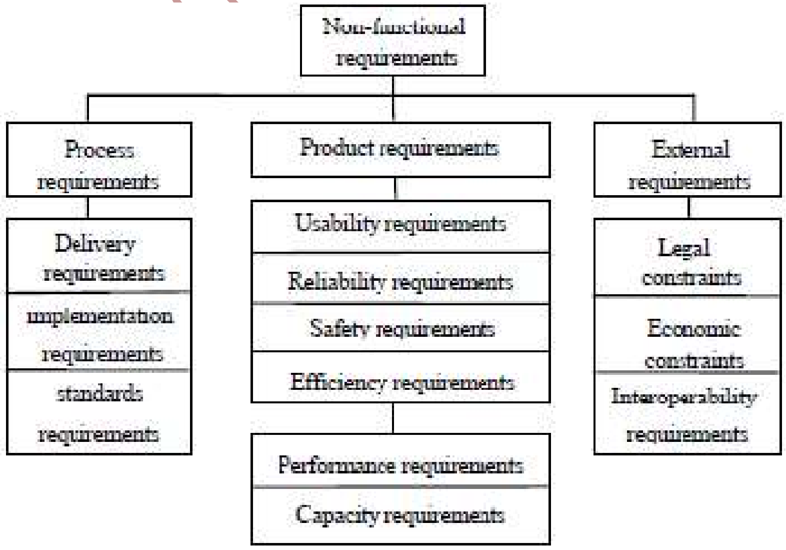


Fig. 2. Non-Functional Requirements

IEEE Definition:

"non functional requirement – in software system engineering, a software requirement that describes not what the software will do, but how the software will do it, for example, software performance requirements, software external interface requirements, design constraints, and software quality attributes. Nonfunctional requirements are difficult to test; therefore, they are usually evaluated subjectively."

There is no a clear distinction between functional and nonfunctional requirements. Whether or not a requirement is expressed as a functional or a non-functional requirement may depend: on the level of detail to be included in the requirements document and the comprehension of application domain and the desired system and also on experience of developers.

Non-functional requirements define the overall qualities or attributes of the resulting system

☐ Non-functional requirements place restrictions on the product being developed, the development process, and specify external constraints that the product must meet.

☐ Examples of NFR include safety, security, usability, reliability and performance requirements.

☐ Project management issues (costs, time, schedule) are often considered as non-functional requirements as well

Some properties of a system may be expressed either as a functional or non-functional property.

Example. The system shall ensure that data is protected from unauthorized access. Conventionally a non-functional requirement (security) because it does not specify specific system functionality.

When expressed as functional requirement: The system shall include a user authorization procedure where users must identify themselves using a login name and password. Only users who are authorized in this way may access the system data. Thus Non-functional requirements may result in new functional requirements statements.

Types of Non-functional requirements

The 'IEEE-Std 830 - 1993' lists 13 non-functional requirements to be included in a Software Requirements Document. Refer Figure 2.

Classification of Non-functional requirements (Jacobson, 1999)

The FURPS+ model is proposed for the Unified Process.

☐ Functional Requirements ☐ Usability ☐ Reliability ☐ Performance☐ Supportability

☐ The FURPS+ model provides additional requirements typically also included under the general label of non-functional requirements:

☐ Implementation requirements ☐ Interface requirements ☐ Operations requirements

☐ Packaging requirements ☐ Legal requirements

Requirements Elicitation

Requirements Elicitation is the process of discovering the requirements for a system by communication with customers, system users and others who have a stake in the system development. Development teams have to take the initiative.

Requirement Sources and Elicitation Techniques

Elicitation Techniques

➜ Traditional Approaches Introspection Interview/survey Group elicitation	➜ Observational approaches Protocol analysis Participant Observation (ethno methodology)
➜ Model-based approaches Goal-based: hierarchies of stakeholders' goals Scenarios: characterizations of the ways in which the system is used Use Cases: specific instances of interaction with the system	➜ Exploratory approaches Prototyping ("plan to throw one away")

Several Classifications are as Discussed in Lab and in Class.

For more details refer Power point presentation.

Use Cases

- A *use case* is a description of a specific interaction that a user may have with the system. A collection of scenarios that describe the thread of usage of a system.
- Use cases are deceptively simple tools for describing the functionality of the software.

- Use cases do not describe any internal workings of the software, nor do they explain how that software will be implemented.
- They simply show how the steps that the user follows to use the software to do his work.
- All of the ways that the users interact with the software can be described in this manner.

- Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
    - What are the main tasks of functions performed by the actor?
    - What system information will the actor acquire, produce or change?
    - Will the actor inform the system about environmental changes?
    - What information does the actor require of the system?
    - Does the actor wish to be informed about unexpected changes

Software Requirement Analysis

requirements modelling will begin with scenario-based modelling, which results in creating a use case

- identify the “customer” and work together to negotiate “product-level” requirements
- build an analysis model
    - focus on data
    - define function
    - represent behavior
- prototype areas of uncertainty
- develop a specification that will guide design
- conduct formal technical reviews.
- **Analysis Process is depicted in Figure 3.**

Davis Principle

- Understand the problem before you begin to create the analysis model.

- Develop prototypes that enable a user to understand how human-machine interaction will occur.
- Record the origin of and the reason for every requirement.
- Use multiple views of requirements.
- Prioritize requirements.
- Work to eliminate ambiguity.

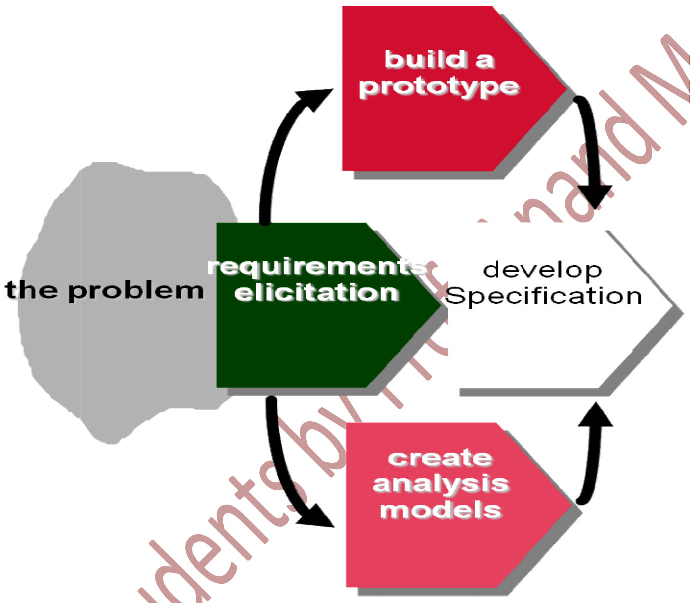


Fig. 3. Analysis Process

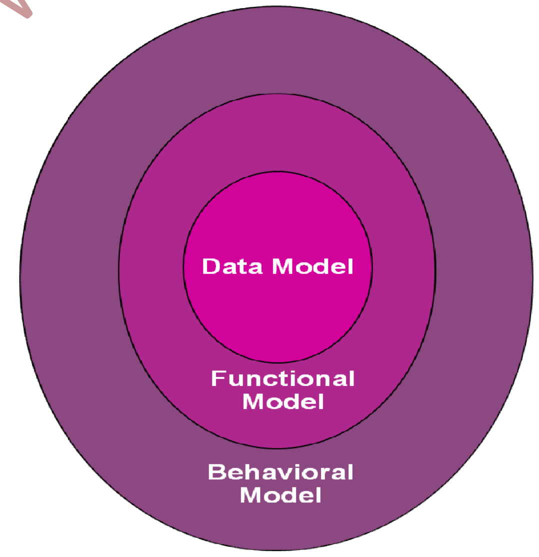


Fig. 4. Analysis Model

Analysis Principle

I. Model the Data Domain:

- define data objects
- describe data attributes
- establish data relationships

II. Model Function

- identify functions that transform data objects
- indicate how data flow through the system
- represent producers and consumers of data

III. Model Function

- indicate different states of the system
- specify events that cause the system to change state

IV. Partition the Models

- refine each model to represent lower levels of abstraction
  - refine data objects
  - create a functional hierarchy
  - represent behavior at different levels of detail

V. Essence

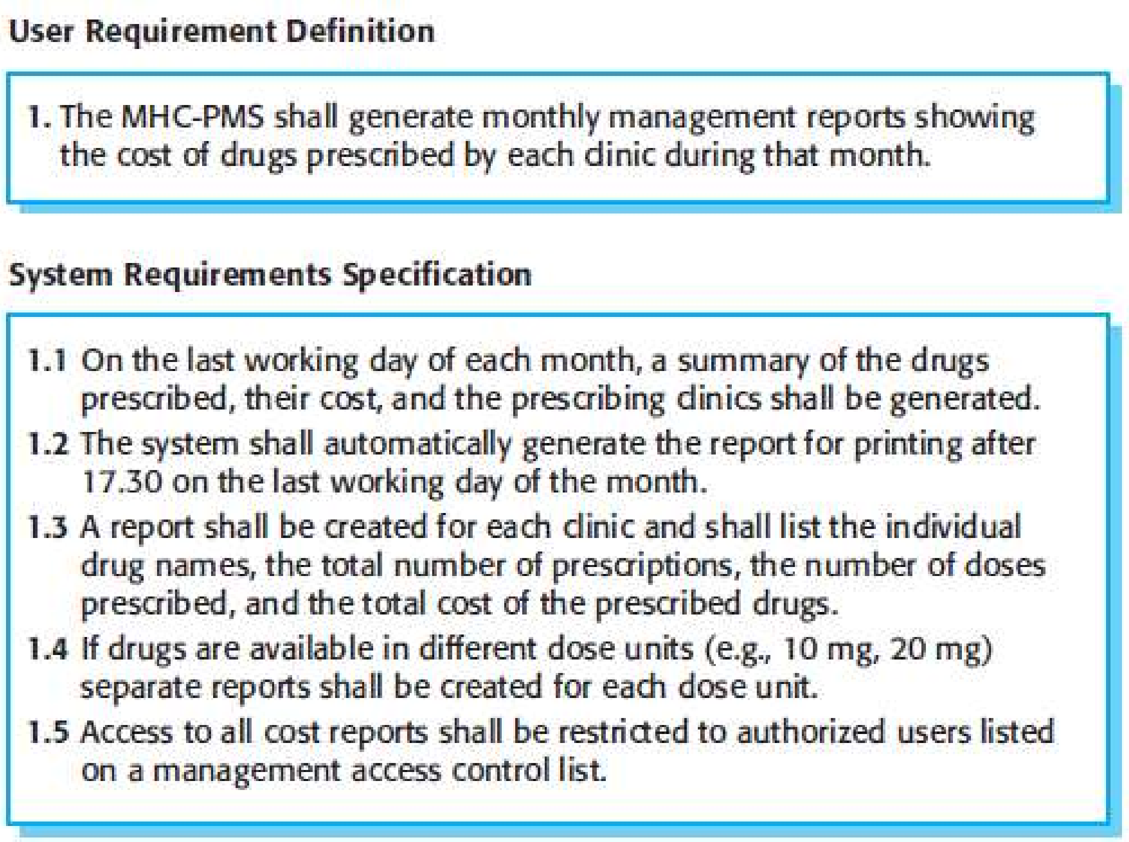
- begin by focusing on the essence of the problem without regard to implementation details

Software Requirements Specification (SRS)

The *software requirements specification* (SRS) represents a complete description of the behavior of the software to be developed.

- The SRS includes:
  - A set of use cases that describe all of the interactions that the users will have with the software.
  - All of the functional requirements necessary to define the internal workings of the software: calculations, technical details, data manipulation and processing, and other specific functionality that shows how the use cases are to be satisfied
  - Nonfunctional requirements, which impose constraints on the design or implementation (such as performance requirements, quality standards or design constraints).

Example of System Requirement Specification



User Requirement Definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System Requirements Specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.

1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.

1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.

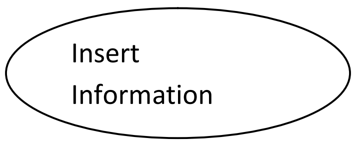
1.4 If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.

1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

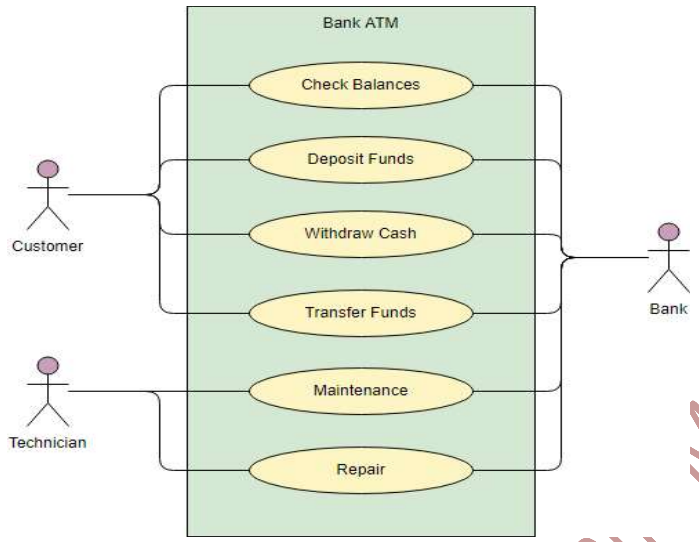
Use case Modeling

Ivar Jackobson has been credited with inventing use cases which appeared in object oriented community somewhere in 1992.

A Use Case describes a function (behaviour) provided by the system in terms of flowing events, including interaction with actors. It is initiated by an actor. Each use case has a name with entry and termination condition. Graphical Notation: An oval with the name of the use case



Use Case Model: The set of all use cases specifying the complete functionality of the system.



Users, Roles and which user can invoke which use case are shown in figure.

A use case model consists of use cases and use case associations

☐ A use case association is a relationship between use cases.

The important types of use case associations: Include, Extends, Generalization

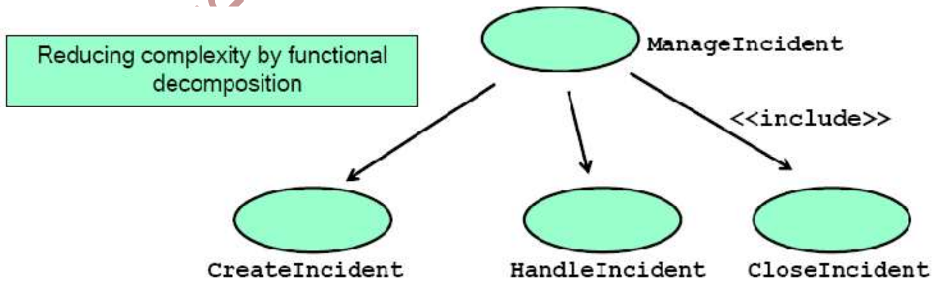
♦ Include: A use case uses another use case ("functional decomposition").

♦ Extends: A use case extends another use case.

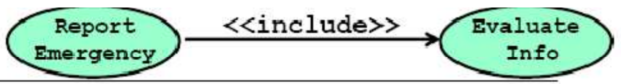
♦ Generalization: An abstract use case has different specializations.

Problem: A function in the original problem statement is too complex to be solvable immediately or the function already exists.

♦ Solution: Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases.



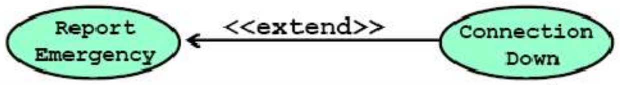
- <<include>> relationship



Flow of Events ...

3. **FieldOfficer** completes and submitts the form
    4. SYSTEM receives the form, evaluates the information (by using Use Case *EvaluateInfo)* being received and sends an acknowledge

- <<extend>> relationship



Entry Condition The *ConnectionDown* use case extends *any use* case in which the connection between the **FieldOfficer** and the SYSTEM can be lost

Flow of Events 1. The FieldOfficer is notified that the connection is broken

Dynamic Modeling with UML

Modeling of behavioral aspects among the participating objects is Dynamic Modeling. It shows how the behavior of a use case (or scenario) is distributed among its participating objects. In it we Tie use cases with objects. Identify new objects and classes. It helps to determine how decentralized the system is. Its purpose is to Detect and supply methods for the object model.

♦ How do we do this?

☐ Start with use case or scenario

☐ Model interaction between objects: (three types of objects i.e. Entity, boundary and control) and draw sequence diagram.

☐ Model dynamic behavior of a single object: State chart diagram.

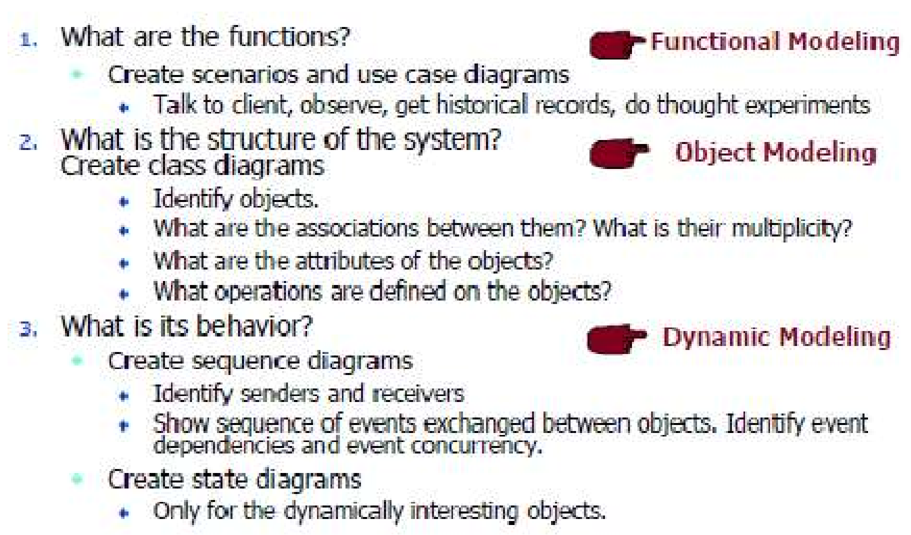
Sequence Diagram can be read out of use cases. It is Dynamic behavior of a set of objects arranged in time sequence and good for real-time specifications and complex scenarios. It tells the Information of the life time of objects and Relationships between objects.

♦ State Chart Diagram describes the dynamic behavior of a single object. A state machine that describes the response of an object of a given class to the receipt of outside stimuli (Events).

2.5 System and Software Requirement Specifications

The goal of analysis is to discover problems, incompleteness and inconsistencies in the elicited requirements. These are then fed back to the stakeholders to resolve them through the negotiation process. Analysis is interleaved with elicitation as problems are discovered when the requirements are elicited. A problem checklist may be used to support analysis. Each requirement may be assessed against the checklist.

Software (System) Requirement Analysis (Summary)

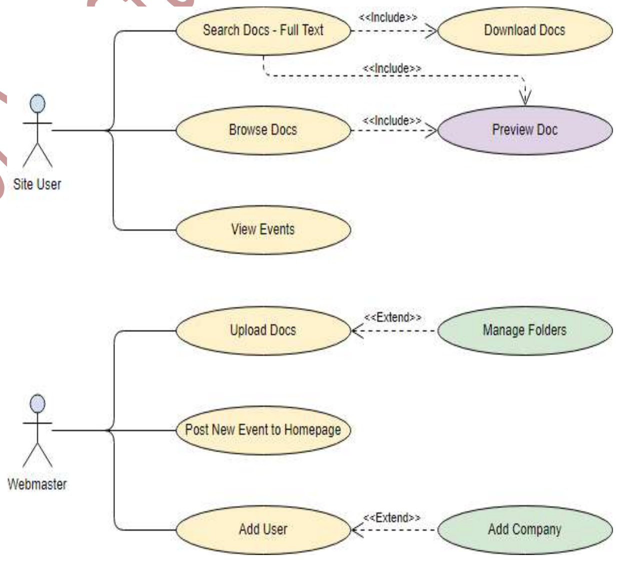


1. What are the functions? — Functional Modeling
   - Create scenarios and use case diagrams
     - Talk to client, observe, get historical records, do thought experiments
2. What is the structure of the system? — Object Modeling
   Create class diagrams
     - Identify objects.
     - What are the associations between them? What is their multiplicity?
     - What are the attributes of the objects?
     - What operations are defined on the objects?
3. What is its behavior? — Dynamic Modeling
   - Create sequence diagrams
     - Identify senders and receivers
     - Show sequence of events exchanged between objects. Identify event dependencies and event concurrency.
   - Create state diagrams
     - Only for the dynamically interesting objects.

The above activities results in the Requirement Specification document.

Example Use Case:

The **Document Management System (DMS)** use case diagram example below shows the actors and use cases of the system. In particular, there are include and extend relationships among use cases.



Must refer to learn more about writing effective use-cases:
https://www.visual-paradigm.com/tutorials/writingeffectiveusecase.jsp

Requirement Validation

Requirements validation is a critical step in the development process, usually during requirements engineering or requirements analysis and also at delivery (client acceptance test) of software.

♦ **Requirements validation criteria:**

- ✓ Complete: All possible scenarios, in which the system can be used, are described, including exceptional behavior by the user or the system.
- ✓ Consistent: There are no two functional or nonfunctional requirements that contradict each other.
- ✓ Unambiguous: Requirements can not be interpreted in mutually exclusive ways
- ✓ Correct: The requirements represent the client's view
- ✓ More Requirements validation criteria
- ✓ Realistic: Requirements can be implemented and delivered
- ✓ Verifiable: Requirements can be checked. It needs an exact description of the requirements

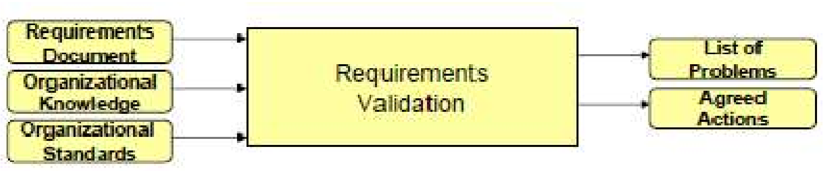
Problem with requirements validation: Requirements change very fast during requirements elicitation.

♦ Tool support for managing requirements:

- Store requirements in a shared repository
- Provide multi-user access
- Automatically create a system specification document from the repository
- Allow for change management
- Provide traceability throughout the project lifecycle

Analysis vs. Validation Differentiation	
Analysis (Verification) works with raw requirements as elicited from the system stakeholders	Validation works with a final draft of the requirements document i.e. with negotiated and agreed requirements
Usually incomplete and expressed in an informal and unstructured way	All known incompleteness and inconsistency are removed
Incorporation of stakeholder is important	Validation process is more concerned with the way in which the requirements are described
This tests "Have we got the right requirements"	This tests "Have we got the requirements right"

Validation Inputs and Outputs



Notes on following topics will be provided into Part – II of this UNIT.

Refer PPT.

- *Data Modeling Concepts - Class Based Modeling - Flow Oriented Modeling – Creating Behavioral Modeling - Patterns For Modeling*.