Unit - II

Requirements Analysis and Specification

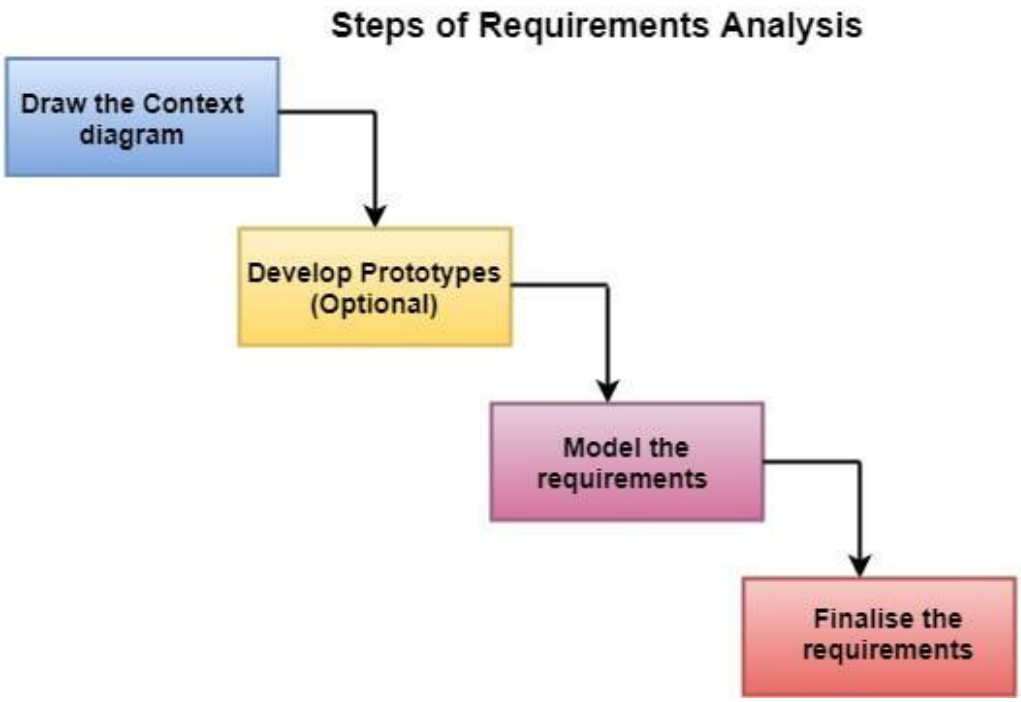
Syllabus:

Requirements Analysis and Specification: Requirements Analysis – Software Requirements – Requirements Engineering – Eliciting Requirements – Developing Use Cases – Building the Requirements Model – Negotiating and Validating Requirements.

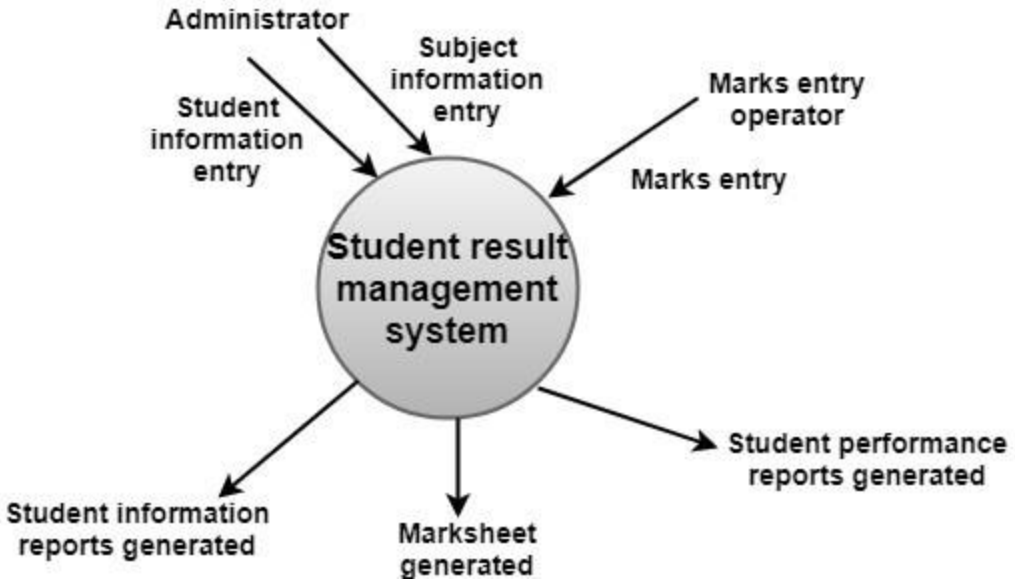
Requirements Analysis:

Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

The various steps of requirement analysis are shown in fig:



(i) Draw the context diagram: The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



(ii) Development of a Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

(iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

(iv) Finalise the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

Software Requirements

Software requirements are a crucial aspect of the software engineering process. They serve as the foundation for developing and testing a software system, providing a clear and detailed description of what the system should accomplish. Requirements help in understanding the needs of the end-users and stakeholders, guiding the development team throughout the software development life cycle.

Here are the key types of software requirements:

1. Functional Requirements:

- These describe the functions and features the software system should provide.
- Examples include user authentication, data processing, reporting, and any specific functionalities required by the users.

2. Non-Functional Requirements:

- These specify the attributes of the system, such as performance, reliability, scalability, and security.
- Examples include response time, system availability, security measures, and user interface design.

3. User Requirements:

- These represent statements in natural language that describe the functionality of the system from a user's perspective.
- User stories and use cases are common techniques to gather and document user requirements.

4. System Requirements:

- These describe the characteristics and constraints of the entire system.
- Examples include hardware and software compatibility, system interfaces, and external dependencies.

5. Performance Requirements:

- These detail the acceptable levels of performance, including response times, throughput, and resource usage.
- Performance requirements are critical for systems where speed and efficiency are crucial.

6. Security Requirements:

- These specify measures that must be taken to protect the system and its data from unauthorized access, data breaches, or other security threats.

7. Usability Requirements:

- These address the user interface and user experience aspects of the system, ensuring that it is user-friendly and meets the needs of its intended audience.

8. Reliability Requirements:

- These describe the system's ability to perform its functions without failures or errors over a specified period.

9. Maintainability Requirements:

- These address the ease with which the software can be modified or enhanced over time, including considerations for code readability, modularity, and documentation.

10. Compatibility Requirements:

- These specify the compatibility of the software with other systems, platforms, or software versions.

11. Legal and Regulatory Requirements:

- These include any legal or regulatory standards that the software must adhere to, such as data protection laws or industry-specific regulations.

12. Documentation Requirements:

- These outline the documentation that must be provided along with the software, including user manuals, system manuals, and technical documentation.

13. Testing Requirements:

- These specify the criteria and procedures for testing the software, including test cases, acceptance criteria, and performance testing parameters.

14. Interface Requirements:

- These describe how the software will interact with other software components, systems, or external services.

15. Data Requirements:

These detail the data needed by the system, including data formats, storage, retrieval, and processing requirements.

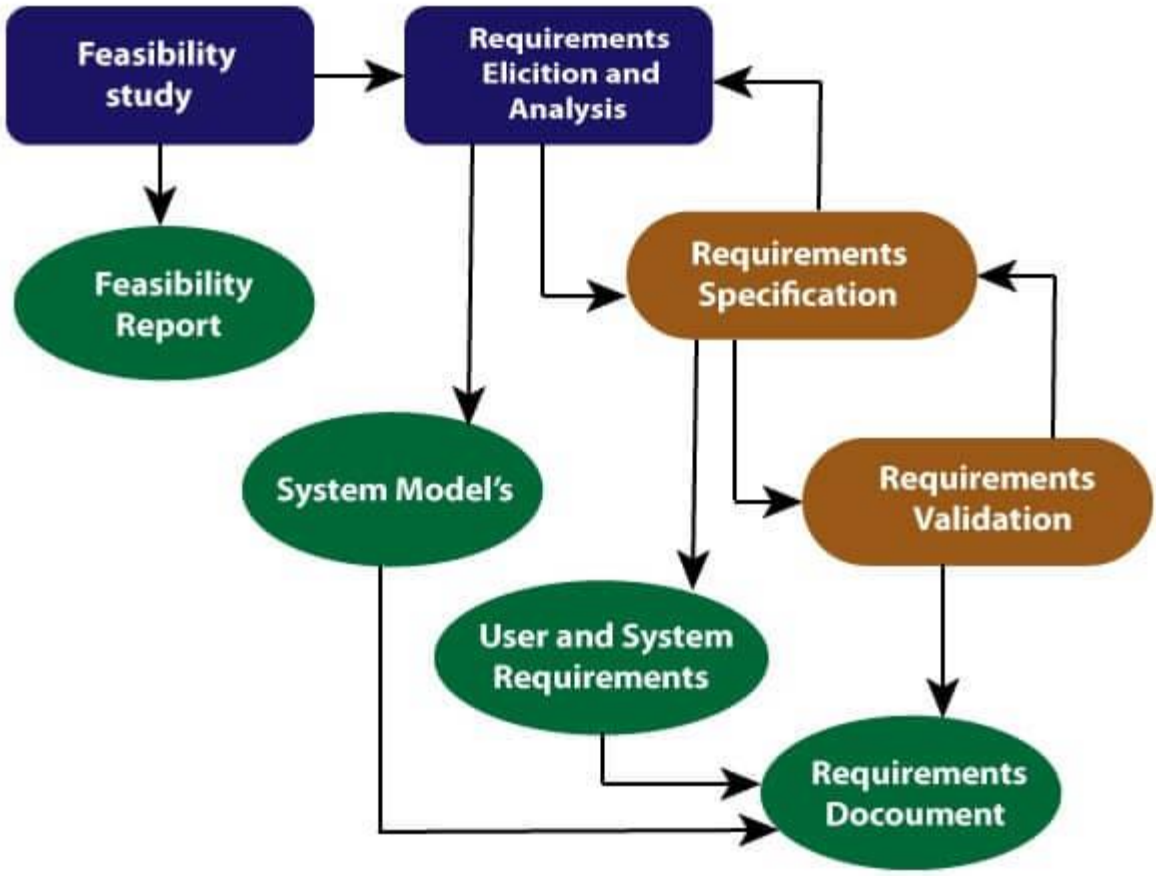
REQUIREMENTS ENGINEERING:

Requirements engineering (RE) refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

Requirement Engineering Process

It is a four-step process, which includes -

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management



Requirement Engineering Process

1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

Types of Feasibility:

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

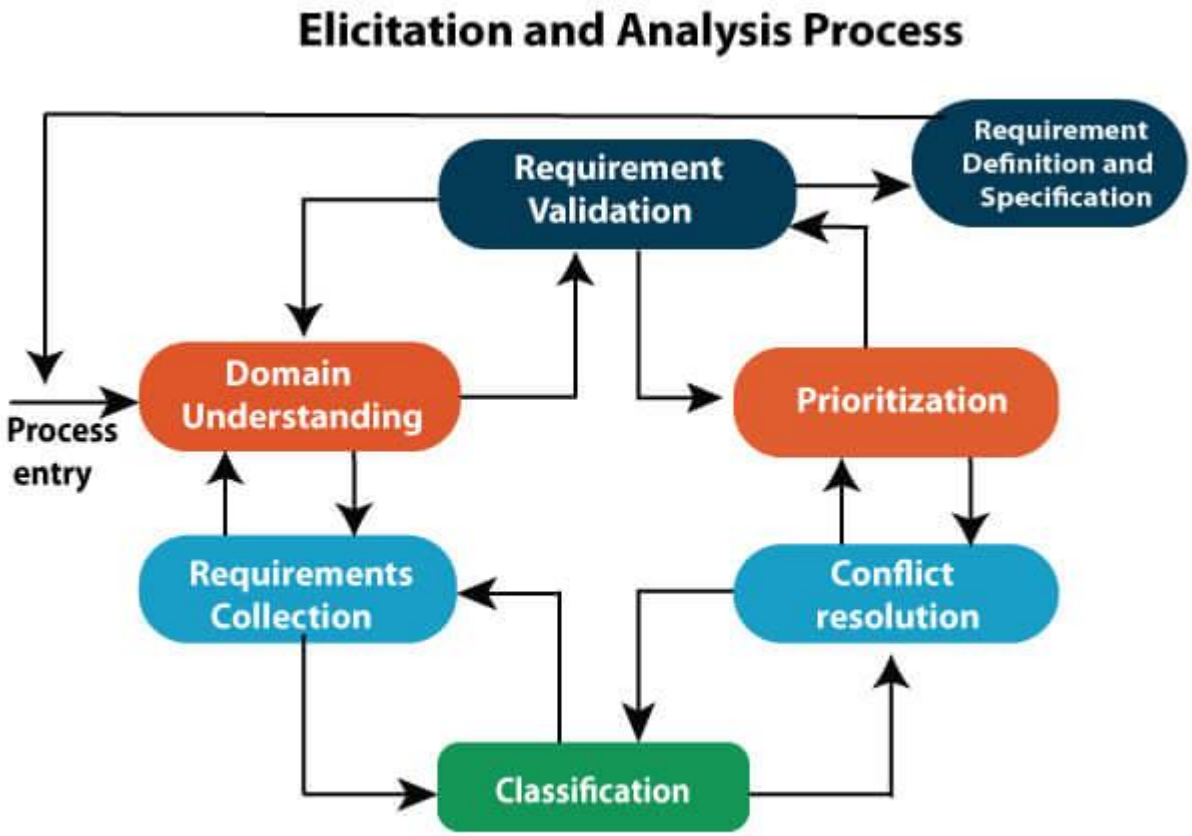
2. Requirement Elicitation and Analysis:

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Problems of Elicitation and Analysis

- Getting all, and only, the right people involved.
- Stakeholders often don't know what they want
- Stakeholders express requirements in their terms.
- Stakeholders may have conflicting requirements.
- Requirement change during the analysis process.
- Organizational and political factors may influence system requirements.



3. Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

- **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
- **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
- **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "***E-R diagram***." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

4. Software Requirement Validation:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

Requirements Validation Techniques

- **Requirements reviews/inspections:** systematic manual analysis of the requirements.
- **Prototyping:** Using an executable model of the system to check requirements.
- **Test-case generation:** Developing tests for requirements to check testability.
- **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

Eliciting Requirements:

Eliciting requirements is a critical phase in the software engineering process, as it involves gathering and understanding the needs and expectations of stakeholders to define what the software system should accomplish. Successful requirement elicitation ensures that the development team and stakeholders have a shared understanding of the project. Here are some techniques commonly used for eliciting requirements:

1. Interviews:

- Conduct one-on-one or group interviews with stakeholders, including end-users, managers, and subject matter experts. Structured and open-ended questions help gather valuable insights.

2. Surveys and Questionnaires:

- Distribute surveys or questionnaires to a wide audience to collect information about their needs, preferences, and expectations. This method is useful for obtaining feedback from a large number of stakeholders.

3. Workshops:

- Organize facilitated group sessions with key stakeholders to discuss and brainstorm requirements collaboratively. Workshops encourage active participation and allow for real-time clarification of doubts.

4. Brainstorming:

- Encourage stakeholders to freely share their ideas and requirements. This can be done in a structured session or informally. The goal is to generate a wide range of ideas that can be further refined.

5. Observation:

- Directly observe users and stakeholders in their work environment to understand their processes, challenges, and needs. Observations provide valuable insights into real-world scenarios.

6. Prototyping:

- Create low-fidelity or high-fidelity prototypes to help stakeholders visualize the proposed system. Prototypes can serve as a tangible reference point for discussions and refinement of requirements.

7. Use Cases and Scenarios:

- Develop use cases and scenarios that describe how users will interact with the system in various situations. This technique helps identify functional and non-functional requirements.

8. Document Analysis:

- Review existing documentation, such as business plans, user manuals, and process documents. Analyzing these documents can uncover requirements and provide context for the development team.

9. Focus Groups:

- Bring together a diverse group of stakeholders to discuss and provide feedback on specific aspects of the project. Focus groups can help identify common themes and priorities.

10. Job Shadowing:

- Spend time with end-users and stakeholders to understand their daily tasks and challenges. This technique provides a deeper understanding of the context in which the software will be used.

11. Storytelling:

- Encourage stakeholders to tell stories about their experiences, challenges, and aspirations related to the software. This technique can uncover hidden requirements and provide rich context.

12. Role Playing:

- Engage stakeholders in role-playing exercises where they act out scenarios related to the software. This can help uncover user needs and preferences in a dynamic way.

13. Competitor Analysis:

- Study similar systems or products in the market to identify features and functionalities that are valued by users. This can inform the development of requirements for your project.

14. Socialmedia and Online Forums:

- Monitor social media platforms and online forums relevant to the industry or user base. User discussions and feedback can provide valuable insights into their expectations and pain points.

Developing Use Cases:

A use case is a written description of how users will perform tasks on your website. It outlines, from a user's point of view, a system's behavior as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled.

Benefits of Use Cases

Use cases add value because they help explain how the system should behave and in the process, they also help brainstorm what could go wrong. They provide a list of goals and this list can be used to establish the cost and complexity of the system. Project teams can then negotiate which functions become requirements and are built.

Elements of a Use Case

Depending on how in depth and complex you want or need to get, use cases describe a combination of the following elements:

- **Actor** – anyone or anything that performs a behavior (who is using the system)
- **Stakeholder** – someone or something with vested interests in the behavior of the system under discussion (SUD)
- **Primary Actor** – stakeholder who initiates an interaction with the system to achieve a goal
- **Preconditions** – what must be true or happen before and after the use case runs.
- **Triggers** – this is the event that causes the use case to be initiated.
- **Main success scenarios** [Basic Flow] – use case in which nothing goes wrong.
- **Alternative paths [**Alternative Flow] – these paths are a variation on the main theme. These exceptions are what happen when things go wrong at the system level.

Building the Requirements Model:

The intent of the analysis model is to provide a description of the required informational, functional, and behavioural 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. For that reason, the analysis model is a snapshot of requirements at any given time.

Elements of the Requirements Model:

There are many different ways to look at the requirements for a computer-based system. Different modes of representation force you to consider requirements from different viewpoints—an approach that has a higher probability of uncovering omissions, inconsistencies, and ambiguity.

Scenario-based elements.

The system is described from the user's point of view using a scenario-based approach. For example, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases. Scenario-based elements of the requirements model are often the first part of the model that is developed. Three levels of elaboration are shown, culminating in a scenario-based representation.

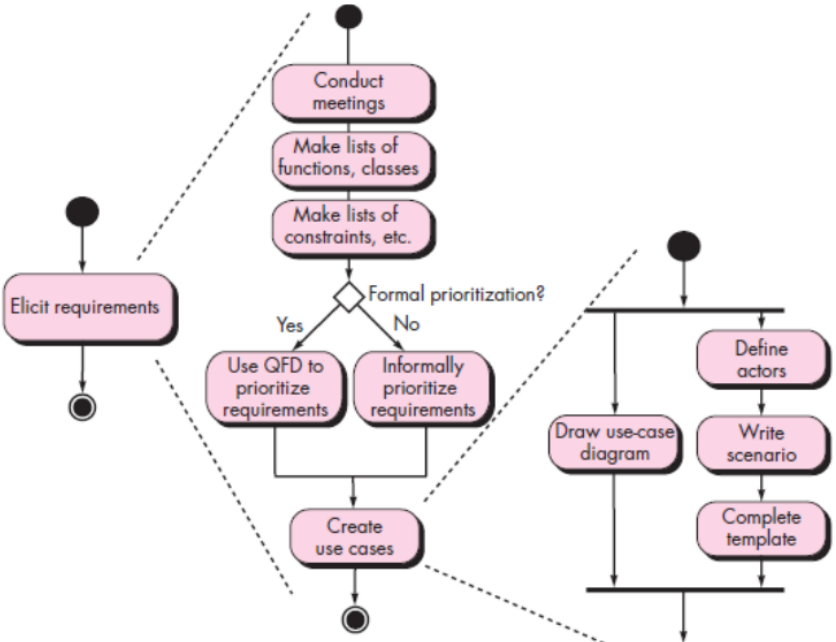


Fig 5.3: UML activity diagrams for Eliciting

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 behaviours.

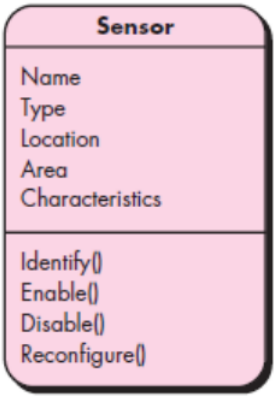


Fig 5.4: Class diagram for sensor

Behavioural 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. The state diagram is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state. A state is any externally observable mode of behavior. In addition, the state diagram indicates actions taken as a consequence of a particular event.

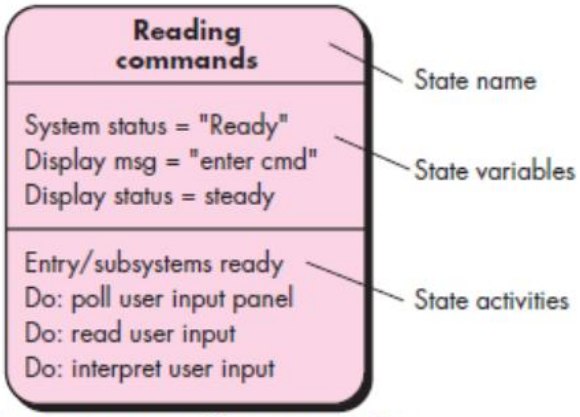


Fig 5.5: UML State diagram notation

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. Input may be a control signal transmitted by a transducer, a series of numbers typed by a human operator, a packet of information transmitted on a network link, or a voluminous data file retrieved from secondary storage. The transform(s) may comprise a single logical comparison, a complex numerical algorithm, or a rule-inference approach of an expert system.

Analysis Patterns:

Anyone who has done requirements engineering on more than a few software projects begins to notice that certain problems reoccur across all projects within a specific application domain. These analysis patterns suggest solutions (e.g., a class, a function, a behavior) within the application domain that can be reused when modeling many applications. Analysis patterns are integrated into the analysis model by reference to the pattern name. They are also stored in a repository so that requirements engineers can use search facilities to find and apply them. Information about an analysis pattern (and other types of patterns) is presented in a standard template.

Negotiating Requirements:

The inception, elicitation, and elaboration tasks in an ideal requirement engineering setting determine customer requirements in sufficient depth to proceed to later software engineering activities. You might have to negotiate with one or more stakeholders. Most of the time, stakeholders are expected to balance functionality, performance, and other product or system attributes against cost and time-to-market.

The goal of this discussion is to create a project plan that meets the objectives of stakeholders while also reflecting the real-world restrictions (e.g., time, personnel, and budget) imposed on the software team. The successful negotiations aim for a "win-win" outcome. That is, stakeholders, benefit from a system or product that meets the majority of their needs, while you benefit from working within realistic and reasonable budgets and schedules.

At the start of each software process iteration, *Boehm* defines a series of negotiating actions. Rather than defining a single customer communication activity, the following are defined:

1. Identifying the major stakeholders in the system or subsystem.
2. Establishing the stakeholders' "win conditions."
3. Negotiation of the win conditions of the stakeholders in order to reconcile them into a set of win-win conditions for all people involved.

Validating Requirements:

Each aspect of the requirements model is checked for consistency, omissions, and ambiguity as it is developed. The model's requirements are prioritised by stakeholders and bundled into requirements packages that will be implemented as software increments.

The following questions are addressed by an examination of the requirements model:

- Is each requirement aligned with the overall system/product objectives?
- Were all requirements expressed at the appropriate level of abstraction? Do some criteria, in other words, give a level of technical information that is inappropriate at this stage?
- Is the requirement truly necessary, or is it an optional feature that may or may not be critical to the system's goal?
- Is each requirement well defined and unambiguous?

- Is each requirement attributed? Is there a source noted for each requirement?
- Are there any requirements that conflict with others?
- Is each requirement attainable in the technical environment in which the system or product will be housed?
- Is each requirement, once implemented, testable?
- Does the requirements model accurately represent the information, functionality, and behaviour of the system to be built?
- Has the requirements model been “partitioned” in such a way that progressively more detailed information about the system is exposed?
- Have requirements patterns been used to reduce the complexity of the requirements model?
- Have all patterns been validated properly? Are all patterns in accordance with the requirements of the customers?