UNIT II

REQUIREMENT ENGINEERING

Software Engineering Practice – communication Practice – Planning practice Modeling practice– Construction Practice –Deployment. Requirements Engineering - Requirements Engineering tasks– Initiating the requirements Engineering Process- Eliciting Requirements – Developing Use cases –Building the Analysis Models – Elements of the Analysis Model – Analysis pattern – Negotiating Requirements – Validating Requirements.

Software Engineering Practice

1. Definition

i. Practice is the collection of concepts, principles, methods, and tools that a software engineer calls upon a daily basis.

ii. Software engineering is the practice for conducting, implementing, and guidelines for the practice to be implemented.

2. Importance

i. Practice helps us to understand the concepts and the principles that must be followed for development of software engineering projects and project development.

ii. Practice instructs us to develop the model in more systematic form and at the pace it needs to be developed for deployment.

3. The Essence of Practice

The practice involves problem solving, modelling, designing, code generation, testing and quality assurance listed below in four steps.

i. Understand the problem (Communication and analysis)

a) Who has a stake in the solution to the problem?

b) What are the unknowns?

c) Can the problems be compartmentalized?

d) Can the problem be represented graphically?

ii. Plan a solution (modeling and software design)

a) Have you seen similar problems before?

b) Has a similar problem been solved?

c) Can sub problems be defined?

d) Can you represent a solution in a manner that leads to an effective implementation?

iii. Carry out the plan (code generation)

a) Does the solution conform?

b) Is each component part of the solution probably correct?

iv. Examine the result for accuracy (testing and quality assurance)

a) Is it possible to test each component part of the solution?

b) Does the solution produce results that conform to the data, function, features, and behavior that are required?

CORE PRINCIPLES

1. The reason it all exists. Provide value to the user

i. The software system exists to provide value for the user.

ii. Before specifying the problem the requirement and the specifications have to be laid down.

iii. The hardware and the software platform to be decided for implementation.

2. Keep it simple stupid

i. The terms and the design used for development of the project should be kept simple and easily understandable.

ii. All the terms used should be easy to facilitate the basic concept of the project.

3. Maintain the vision

i. A clear vision is important for the development of a software.

ii. Compromising the architectural vision of the project weakens the development of the software.

iii. The developer should hold the vision and ensure the successful development and deployment of the software.

4. What you reproduce, someone else will have to consume. (implement knowing someone else will have to understand what you are doing)

i. Always specify, design and implement knowing that someone else is going to understand what is being developed.

ii. Customers for the product development is very large.

iii. Design the data structure and the implementation keeping implementation in mind and the end user.

iv. Code with the concern that the product has to be implemented and maintained by the end user.

5. Be open to the future

i. The system designed today should be adaptable to the development and changes in the future at a low cost.

ii. There should not be much changes to the software to adopt to the new changes in the future development.

6. Plan ahead for reuse

i. The design and specifications should be developed in such a way that they can be reused for other implementations.

ii. The code and the design should be well documented for the use in future.

7. Think!

i. Before designing and implementation a proper thought should be to the end result.

ii. Proper data structure and the design and implementation strategy should be developed if the software needs modification in the future.

Communication Practices

Communication practice is two way communication between the client and the developer, hence it is also known as requirement elicitation.

1. Listen carefully

i. To collect lots of data from the client, the developer team has to listen carefully.

ii. Maximum information with respect to requirement and the specifications should be collected before the implementation and the designing of the software.

2. Prepare before you communicate

i. A proper agenda or the guidelines for the meetings should be prepared before the start of the meeting.

3. Have a facilitator for any communication meeting

i. The requirement gathering and the specification are important for any software development, hence communication should continue till the requirement gathering is over.

4. Face-to-face communication is best

i. It is always better to sit across the table and have discussion on the requirement on the software development by the client and the developer.

5. Take notes and document decisions

i. The important points discussed should also be recorded.

ii. Proper notes and the documentation is important for the successful completion and deployment of the project.

6. Strive for collaboration

i. Collaboration in terms of teamwork is required for the successful completion of software.

ii. The collective knowledge of the team members should be implemented in the development.

7. Stay focused and modularize your discussion

i. As the development is the working of many team members, so the possibility of the discussion going from one topic to the other topic is quite possible.

ii. As a good software developer it is required that the discussion remains focused on the specified area.

8. Draw a picture if something is unclear

i. Drawing flowcharts, E-R diagrams and other supporting graphical representations give clarity to the discussion and the documentation.

9. Move on once you agree, move on when you can't agree, move on if something unclear can't be clarified at the moment

i. Healthy discussion leads to the final conclusion of successful implementation of software

ii. Once reached to final statement recorded should move to the next step.

iii. If no conclusion is reached than that point should be left and move ahead with new implementation which is cost effective.

10. Negotiation is not a contest or game

i. Negotiation should be mutual not to put someone down or make them feel to be the loser.

Planning Practices

1. Understand the scope

i. To plan the development of the software after the communication principles are been laid down, analyze the scope of the software.

ii. The implementation strategy and its effectiveness is considered and scope is decided.

2. Involve the customer in planning

i. The client should be involved in the continuous development of the software.

ii. The requirement and the specifications should not change frequently.

3. Recognize that planning is iterative

i. Planning is iterative, but the specification should not change continuously.

ii. After the analysis phase the requirement should be fixed and should not change.

4. Estimate based on what you know

i. The knowledge of the developer is used for the final estimation of the planning phase of the software.

ii. New techniques if unknown should not be considered for planning.

5. Consider the risk

i. The risk in the development should be considered for the software to be with the designing standard and for future scope and development.

6. Be realistic

i. Considering the latest trends in the development and not choosing something which is not practically possible to implement.

7. Adjust granularity as you define the plan

i. The specification should be refined as the plan is defined at each stage of development.

8. Define how you intend to ensure quality

i. The quality of the software should be maintained by providing help at each level of the software.

ii. Proper documentation should be maintained and given at the time of deployment to the client.

iii. The end user license and agreement (EULA).should be provided

9. Describe how you intend to accommodate change

i. The up gradation option to the software should be provided to the client in the future with the regular maintenance policy.

10. Always keep track of the plan and make changes as required

i. The continuous communication with the client to be done to make the required changes to the planning and the development phase.

Modelling Practices

1. Meaning Of Software Modelling

Software modelling follows the seven W5HH principles for the software development.

i. Why is the system being developed?

ii. What will be done?

iii. When will it accomplished?

iv. Who is responsible for the function?

v. Where they are organizationally located?

vi. How will the job be done technically and managerially?

2. Analysis Modelling

i. The information domain for the problem must be represented and understood.

- The information that flows into the system and out of the system should be clearly laid down.

ii. Functions performed by the software must be defined

- The software developed should provide direct benefit to the user of the application software.
- Software functions can be described at the different levels of abstraction.
- Software functions transform data that flow into the system.

iii. Software behavior must be represented as consequences of external events

- The behavior of the software is driven by its interaction and nature with the external environment.
- Input data provided by the end user, control data provided by an external system.

iv. Models depicting the information, function, and behavior must be partitioned in manner that uncovers detail in a hierarchical fashion

- Analysis modelling help the user to understand the problem and have the analysis for the start of the solution.

- As large problems are difficult to solve, problems are divided into smaller sub problems using divide and conquer.
- It is always easy to find solutions to the smaller sub problems than the large systems.

v. The analysis task should move from essential information toward implementation detail

- Describe the problem with the perspective of the end user.
- The essence of the problem is explained without any consideration that how the solution can be developed.

3. Design Of Modelling

i. Design should be traceable to the analysis model

The analysis model provides the information domain of the problem.

- The design should be able to translate the information about the design, sub problems and the component level design.

ii. Always consider the architecture of the system to be built

- The software development is the skeleton to the product development.
- It effects the implementation with respect to design, data structure, and the manner in which testing can be performed.

iii. Data design is as important as algorithm design

- Data design is an important architectural strategy.
- The data should be designed as the flow of algorithm is designed for the implementation and deployment of the software.

iv. Internal and external interfaces must be designed with care

- Data flows between the modules should be designed with simplicity and processing efficiency.
- A well designed interface makes integration easier.

v. User interface design should be tuned to the needs of the end-user and must focus on use of user

- The user interface is the visible implementation of the software.
- The poor interface always leads to the perception that the software is bad.

vi. Component-level design should be functionally independent

- Each sub function or the module developed should focus on the perfect working functionality of that module.

vii. Components should be loosely coupled to one another and to the external environment

- Coupling is accomplished through many ways like message passing, component interface and global data.
- When the level of coupling increases, error propagation increases with overall maintainability of software decreases.
- Component coupling should be kept as low as possible.

viii. Design models should be easy to understand

- The design models are created for easy communication of information to the users about the code
- The design model should be easily understandable by the tester.
- If the software design model is difficult to understand and communicate than it is not the effective model.

ix. Design should be developed iteratively

CONSTRUCTION PRACTICES

Construction practices comprises of coding and testing principles. The initial focus is on the component level known as unit testing and the other testing are listed below:

i. Integration testing: Conducted as the system is constructed.

ii. Validation testing: That assesses whether requirements have been met for the complete system.

iii. Acceptance testing: That is conducted by the customer in an effort to exercise all required features and functions.

1. CODING PRINCIPLES

The principles which guide the coding tasks are programming languages, programming styles and programming methods

i. Preparation principles: Before you write one line of code, be sure you

- Understand of the problem you are trying to solve
- Understand basic design, principles & concepts.
- Pick a programming language that meets the needs of the software to be build and the environment in which the software will operate.
- Select a programming environment that provides tools that will make you work simpler and easier.
- Create a set of units that will be applied once the component you code is completed.

ii. Coding principles: As you begin writing code, be sure you:

- Constrain your algorithms by following structured programming practice.
- Consider the use of pair programming.
- Select data structures that will meet the needs of the design.
- Understand the software architecture and create interfaces that are consistent with it.
- Keep conditional logic as simple as possible.

- Create nested loops in a way that makes them easily testable.
- Select meaningful variable names and follow other local coding standards.
- Write code that is self-documenting.
- Create a visual layout that aids understanding.

iii. Validation Principles: After you have completed your first coding pass, be sure you:

- Conduct a code walkthrough when appropriate.
- Perform unit tests and correct errors when you have uncovered.
- Refactor the code.

2. TESTING PRINCIPLES

Testing is the process of executing programs with the intent of finding errors.

i. All tests should be traceable to customer requirements:

- The main objective of software testing is to uncover errors.
- It follows that the most server defects are those that cause the program to fail to meet its requirements.

ii. Tests should be planned long before testing begins

- Test planning can begin as soon as the requirements model is complete.
- Detailed definition of test cases can begin as soon as the design model has been solidified.
- For this reason all tests can be planned and designed before any code has been generated.

iii. The Pare to principle applies to software testing

- In this context the Pareto principle implies that 80% of all errors uncovered during testing will likely be traceable to 20% of all program components.

- The problem, of course, is to isolate these suspect components and to thoroughly test them.

iv. Testing should begin "in the small" and progress towards testing "in the large"

- The first tests planned and executed generally focus on individual components.
- As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

v. Exhaustive testing is not possible

- The number of path permutations for even a moderately sized program is exceptionally large.
- For this reason, it is impossible to execute every combination of paths during testing.

SOFTWARE DEPLOYMENT

1. Deployment Activities:

i. Delivery Cycle: Each delivery cycle provides the customer and end users with an operational software increment that provides usable functions and features.

ii. Support Cycle: Each support cycle provides documentation and human assistance for all functions and features introduced during all deployment cycles to date.

iii. Feedback Cycle: Each feedback cycle provides the software team with important guidance that results in modifications to the function, features and approach taken for the next increment.

2. Deployment Principles:

i. Manage customer's expectations or requirements

- In most cases customer wants more than he/she has stated earlier as his requirements or expectations.
- In many cases customer is disappointed, even after getting all his/her requirements satisfied.
- Hence, at the time of software delivery, developer must have skills to manage the customer's expectations and requirements.

ii. Record-keeping mechanism must be established for customer support

- 'Customer Support' is essential and important factor in deployment phase.
- The support should be well planned and with proper record keeping mechanism.

iii. Provide essential instructions, documentations and manual

- Actual software project delivery includes all documentations, help files and guidance for handling the software by user.

iv. Assemble and test complete delivery package

The customer side must get all supporting and essential help from developer's side. For this reason CD with complete assembled and tested delivery package with the following support should be delivered:

- Necessary supported operational features and help.
- Essential manuals required for software troubleshooting.
- All executable file of developed software.
- Necessary installation procedures.

v. Do not deliver any defective or buggy software to the customer

- The software should be tested before deployment to the customer.
- The specification should be with the requirement to the customer.

REQUIREMENTS ENGINEERING

- Requirement is a condition possessed by the software component in order to solve a real world problems.
- Requirement describe how a system should act, appear or perform.
- IEEE defines a requirement as: “A condition that must be possessed by a system to satisfy a contract specification, standard or other formally imposed document.

1. Principles of Requirement Engineering

i. Understand the problem before you start to create the analysis model

- There is a tendency to rush to a solution, even before the problem is understood.
- This often leads to elegant software that solves the wrong problem.

ii. Develop prototypes that enable a user to understand how human-machine interaction will occur

Since the perception of the quality of software is often is based on perception of time “friendliness” of the interface, prototyping (and the interaction that results) is highly recommended.

iii. Record the origin and the reason for every document

This is the step in establishing traceability back to the customer.

iv. Use multiple views of requirement

- Building data, functional and behavioral models provides software engineer three different views.
- This reduces the chances of missing errors.

v. Prioritize the requirements

Requirements should be followed for the tight implementation and delivery of the product.

vi. Work to eliminate ambiguity

The use of more technical reviews should be used for no ambiguity.

2. Requirement Engineering Task

The requirement engineering process tasks are achieved through seven distinct functions as shown in Figure 1

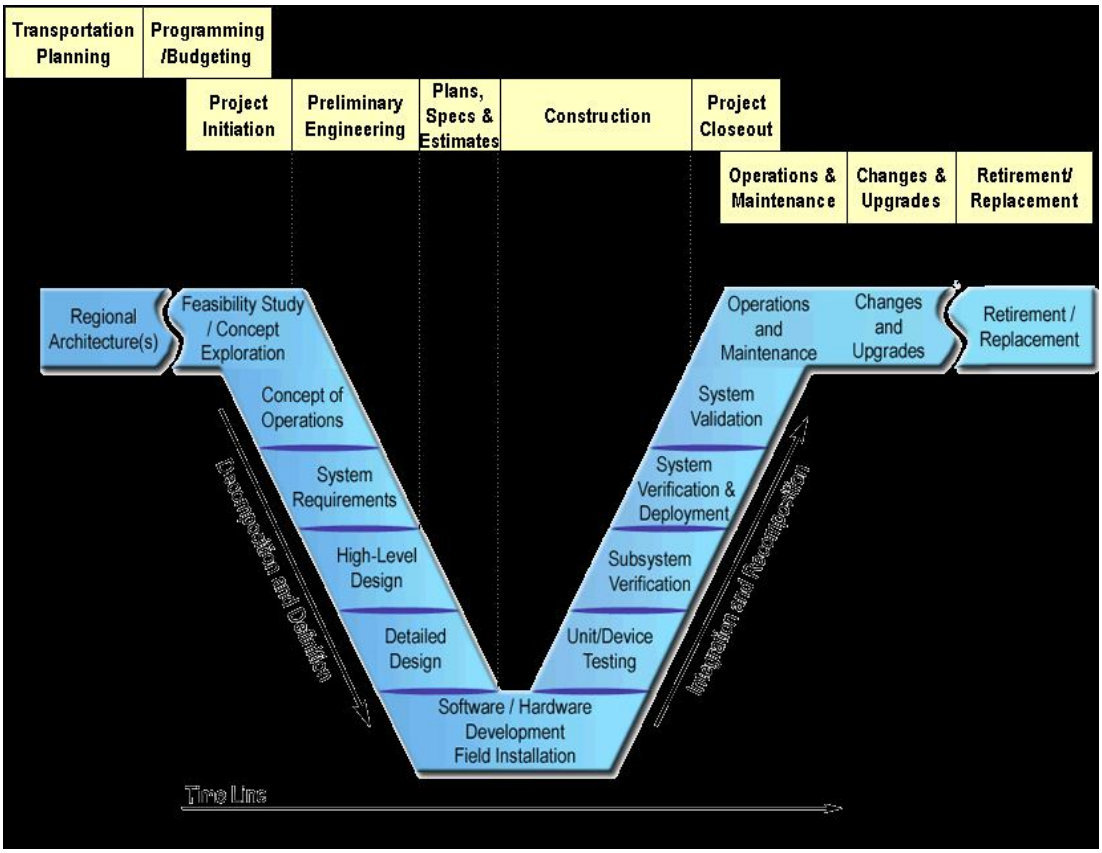

Figure 1: Requirement Engineering

i. Inception

- Inception is also known as the beginning.
- Inception needs various questions to be answered.
- How does a software project get started?

ii. Elicitation

- This is the process by which users of the system are interviewed in order to reveal and understand their requirements.
- This sub-phase involves a high level of client input.

iii. Elaboration

- The most relevant, mission critical requirements are emphasized and developed first.
- This ensures the final product satisfies all of the important requirements, and does so in the most time and cost efficient manner possible.
- Other “should-have” and “nice-to-have” requirements can be relegated to future phases, should they be necessary.

iv. Negotiation

- A milestone within that phase may consist of implementing Use Case x, y and z .
- By scheduling the project in this iterative way, our client has a good understanding of when certain things will be delivered and when their input will be required.
- Additionally, features are generally developed in a “vertical” fashion; once Use Case x has been developed, its unabridged functionality can be demonstrated to the client.
- In this way, our client has a clear understanding of where development time is being spent, and potential issues can be caught early and dealt with efficiently.

v. Specification

- This is the process by which requirements and their analyses are committed to some formal media.
- For this project, we would generate four documents during this sub-phase:
- **Use Case document**: a use case is a description of the system’s response as a result of a specific request from a user. Each use case will cover certain requirements discovered during elicitation and analysis. For example, a use case may cover “Adding a New Employee”, or “Generating Site Report”.
- **Interface prototypes**: an interface prototype is a rough representation of certain critical functionality, recorded as anything from a hand-drawn image to an elaborate HTML mock-up. The prototypes are completely horizontal in nature; they roughly illustrate how the interface for a certain feature will look and what information is accessible but they will have no back end (vertical) functionality.
- **Architecture Diagram**: an architecture diagram will be generated for developer use as it is a high level view of how the software will be structured. Only if we feel it will help in our understanding of the system will this document be created.

- **Database Diagram**: as with the architecture diagram, a database diagram is generally created for developer use. Through these final two media, we are beginning the shift from words (requirements) to code (software).

vi. Validation

- This is the final phase of the requirements engineering process.
- It involves scrutinizing the documents generated during the specification sub-phase and ensuring their relevance and validity.
- This must be done by all interested parties so they are in agreement with the proposed system's behavior, especially in the case of the Use Case document and prototypes.

vii. Management

- As requirements are elicited, analyzed, specified and validated, we will be estimating most features.
- This is an on-going process that will culminate in the delivery of the Project Estimation and Timeline document.
- This will outline the duration of work required to develop and deliver Phase 2.
- As indicated in the Requirements Analysis sub-phase, Phase 2 will likely consist of the majority of the "must-have" features; however it is ultimately up to the client to decide what is most important (even within the "must-have" category), and this will be developed first.

Initiating Requirements Engineering Process

• Identify stakeholders

• Recognize the existence of multiple stakeholder viewpoints

• Work toward collaboration among stakeholders

• These context-free questions focus on customer, stakeholders, overall goals, and benefits of the system

o Who is behind the request for work?

- Who will use the solution?
- What will be the economic benefit of a successful solution?
- Is there another source for the solution needed?

• The next set of questions enable developer to better understand the problem and the customer's perceptions of the solution

- How would you characterize good output form a successful solution?
- What problem(s) will this solution address?
- Can you describe the business environment in which the solution will be used?
- Will special performance constraints affect the way the solution is approached?

• The final set of questions focuses on communication effectiveness

- Are you the best person to give "official" answers to these questions?
- Are my questions relevant to your problem?
- Am I asking too many questions?
- Can anyone else provide additional information?
- Should I be asking you anything else?

Eliciting Requirements

• Collaborative requirements gathering

- Meetings attended by both developers and customers
- Rules for preparation and participation are established
- Flexible agenda is used
- Facilitator controls the meeting
- Definition mechanism (e.g., stickers, flip sheets, electronic bulletin board) used to gauge group consensus

- Goal is to identify the problem, propose solution elements, negotiate approaches, and specify preliminary set of solutions requirements

• Quality function deployment (QFD)

- Identifies three types of requirements (normal, expected, exciting)
- In customer meetings **function deployment** is used to determine value of each function that is required for the system
- **Information deployment** identifies both data objects and events that the system must consume or produce (these are linked to functions)
- **Task deployment** examines the system behavior in the context of its environment
- **Value analysis** is conducted to determine relative priority of each requirement generated by the deployment activities

• User-scenarios

- Also known as use-cases, describe how the system will be used
- Developers and users create a set of usage threads for the system to be constructed

Developing Use-Cases

• Each use-case tells stylized story about how end-users interact with the system under a specific set of circumstances

• First step is to identify **actors** (people or devices) that use the system in the context of the function and behavior of the system to be described

- Who are the primary or secondary actors?
- What preconditions must exist before story begins?
- What are the main tasks or functions performed by each actor?
- What extensions might be considered as the story is described?
- What variations in actor interactions are possible?
- What system information will the actor acquire, produce, or change?

- Will the actor need to inform the system about external environment changes?
- What information does the actor desire from the system?
- Does the actor need to be informed about unexpected changes?

- **Use-case template**
  - Name
  - Primary actor
  - Goal in context
  - Preconditions
  - Trigger
  - Scenario details
  - Extensions
  - Priority
  - When available
  - Frequency of use
  - Channels to secondary actors
  - Open issues

- **Use-case Drawbacks**
  - Lack of formality in use-case descriptions
  - Not all systems have explicitly defined actors
  - Use-cases are not inherently object-oriented
  - Developers have a tendency to functionally decompose use-cases.

Analysis Model

• Intent is to provide descriptions of required information, functional, and behavioral domains for computer-based systems

• Analysis Model Elements

o Scenario-based elements (describe system from user perspective)

o Class-based elements (relationships among objects manipulated by actors and their attributes)

o Behavioral elements (depict system and class behavior as states and transitions between states)

o Flow-oriented elements (shows how information flows through the system and is transformed by the system functions)

• Many different representations can be used to depict the analysis model

o Use-case diagrams

o Activity diagrams

o Class diagrams

o State diagram

o Data flow diagram (DFD)

• Analysis Pattern Template

o Name

o Intent

o Motivation

o Forces and context

o Solution

o Consequences

o Design

o Known use examples

Negotiating Requirements

• Negotiation activities

o Identification of system key stakeholders

o Determination of stakeholders' "win conditions"

o Negotiate to reconcile stakeholders' win conditions into "win-win" result for all stakeholders (including developers)

• Key points

o It's not a competition

o Map out a strategy

o Listen actively

o Focus on other party's interests

o Don't let it get personal

o Be creative

o Be ready to commit

Requirement Validation

• Is each requirement consistent with overall project or system objective?

• Are all requirements specified at the appropriate level off abstraction?

• Is each requirement essential to system objective or is it an add-on feature?

• Is each requirement bounded and unambiguous?

• Do you know the source for each requirement?

• Do requirements conflict with one another?

• Is the requirement achievable in the proposed technical environment for the system or product?

• Is each requirement testable?

• Does the requirements model reflect the information, function, and behavior of the system to be built?

• Has the requirements model been partitioned in a way that exposes more detailed system information progressively?

• Have all the requirements patterns been properly validated and are they consistent with customer requirements?