**UNIT-II**

**Software Engineering Principles:** SE Principles, Communication Principles, Planning Principles, Modeling Principles, Construction Principles, Deployment.

**System Engineering:** Computer-based Systems, The System Engineering Hierarchy, Business Process Engineering, Product Engineering, System Modeling.

**Requirements Engineering:** A Bridge to Design and Construction, Requirements Engineering Tasks, Initiating Requirements Engineering Process, Eliciting Requirements, Developing Use-Cases, Building the Analysis Model, Negotiating Requirements, Validating Requirements.

.

**PART – I**

**Communication Principles**

Before customer requirements can be analyzed, modeled, or specified they must be gathered through the communication activity.

Principle 1. *Listen.*

Principle 2. *Prepare before you communicate.*

Principle 3. *Someone should facilitate the activity.*

Principle 4. *Face-to-face communication is best.*

Principle 5. *Take notes and document decisions*

Principle 6. *Strive for collaboration*

Principle 7. *Stay focused; modularize your discussion*

Principle 8. *If something is unclear, draw a picture.*

Principle 9. *(a) Once you agree to something, move on. (b) If you can’t*

*agree to something, move on. (c) If a feature or function is unclear*

*and cannot be clarified at the moment, move on.*

Principle 10. *Negotiation is not a contest or a game. It works best*

*when both parties win*

**Planning Principles**

The communication activity helps you to define your overall goals and objectives (subject, of course, to change as time passes). However, understanding these goals and objectives is not the same as defining a plan for getting there. The planning activity encompasses a set of management and technical practices that enable the software team to define a road map

Principle 1. Understand the scope of the project.

Principle 2. Involve stakeholders in the planning activity.

Principle 3. Recognize that planning is iterative.

Principle 4. Estimate based on what you know.

Principle 5. Consider risk as you define the plan.

Principle 6. Be realistic.

Principle 7. Adjust granularity as you define the plan.

Principle 8. Define how you intend to ensure quality.

Principle 9. Describe how you intend to accommodate change.

Principle 10. Track the plan frequently and make adjustments as required.

**Modeling Principles**

We create models to gain a better understanding of the actual entity to be built.

Principle 1. The primary goal of the software team is to build software, not create models.

Principle 2. Travel light—don’t create more models than you need.

Principle 3. Strive to produce the simplest model that will describe the problem or the software.

Principle 4. Build models in a way that makes them amenable to change.

Principle 5. Be able to state an explicit purpose for each model that is created.

Principle 6. Adapt the models you develop to the system at hand.

Principle 7. Try to build useful models, but forget about building perfect models.

Principle 8. Don’t become dogmatic about the syntax of the model. If it communicates content successfully, representation is secondary.

Principle 9. If your instincts tell you a model isn’t right even though it seems okay on paper, you probably have reason to be concerned

Principle 10. Get feedback as soon as you can.

**Requirements modeling principles.**

Over the past three decades, a large number of requirements modeling methods have been developed. Investigators have

Principle 1. The information domain of a problem must be represented and understood.

Principle 2. The functions that the software performs must be defined.

Principle 3. The behavior of the software (as a consequence of external events) must be represented.

Principle 4. The models that depict information, function, and behavior must be partitioned in a manner that uncovers detail in a layered (or hierarchical) fashion. .

Principle 5. The analysis task should move from essential information toward implementation detail.

**Design Modeling Principles.** The software begins by representing the totality of the thing to be built

Principle 1. Design should be traceable to the requirements model.

Principle 2. Always consider the architecture of the system to be built.

Principle 3. Design of data is as important as design of processing functions.

Principle 4. Interfaces (both internal and external) must be designed with care. .

Principle 5. User interface design should be tuned to the needs of the end user. .

Principle 6. Component-level design should be functionally independent.

Principle 7. Components should be loosely coupled to one another and to the external environment. .

Principle 8. Design representations (models) should be easily understandable.

Principle 9. The design should be developed iteratively.

**Construction Principles**

The construction activity encompasses a set of coding and testing tasks that lead to operational software that is ready for delivery to the customer or end user. In modern software engineering work, coding may be (1) the direct creation of programming language source code (e.g., Java), (2) the automatic generation of source code using an intermediate design-like representation of the component to be built, or (3) the automatic generation of executable code using a “fourth-generation programming language” (e.g., Visual C).

**Coding Principles.** The principles that guide the coding task are closely aligned

with programming style, programming languages, and programming methods.

However, there are a number of fundamental principles that can be stated:

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

• Understand of the problem you’re trying to solve.

• Understand basic design principles and concepts.

• Pick a programming language that meets the needs of the software to be built and the environment in which it will operate.

• Select a programming environment that provides tools that will make your work easier.

• Create a set of unit tests that will be applied once the component you code is completed.

**Programming 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 (e.g., indentation and blank lines) that aids understanding.

**Validation Principles: *After you’ve completed your first coding pass, be sure you***

• Conduct a code walkthrough when appropriate.

• Perform unit tests and correct errors you’ve uncovered.

• Refactor the code.

**Testing Principles.**

In a classic book on software testing, Glen Myers states a number of rules that can serve well as testing objectives:

• Testing is a process of executing a program with the intent of finding an error.

• A good test case is one that has a high probability of finding an as-yet undiscovered error.

• A successful test is one that uncovers an as-yet-undiscovered error.

**Principle 1. *All tests should be traceable to customer requirements.***7

The objective of software testing is to uncover errors. It follows that the most severe defects (from the customer’s point of view) are those that cause the program to fail to meet its requirements.

**Principle 2. *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. Therefore, all tests can be planned and designed before any code has been generated.

**Principle 3. *The Pareto principle applies to software testing.*** In this context the Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them.

**Principle 4. *Testing should begin “in the small” and progress toward 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.

**Principle 5. *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. It is possible, however, to adequately cover program logic and to ensure that all conditions in the component-level design have been exercised.

**Deployment Principles**

Principle 1. *Customer expectations for the software must be managed.*

Principle 2. *A complete delivery package should be assembled andtested.*

Principle 3. *A support regime must be established before the software is delivered.*

Principle 4. *Appropriate instructional materials must be provided toend users.*

Principle 5. *Buggy software should be fixed first, delivered later.*

**PART - II**

**System Engineering:** Computer-based Systems, The System Engineering Hierarchy, Business Process Engineering, Product Engineering, System Modeling.

**System Engineering**

System engineering focuses on a variety of elements, analyzing, designing, and organizing those elements into a system that can be a product, a service, or a technology for the transformation of information

**Computer-Based Systems:** All complex systems can be viewed as being composed of cooperating subsystems. A computer-based system makes use of a variety of system elements.

1. **Software**: programs, data structures, and related work products.

2. **Hardware**: electronic devices that provide computing capabilities.

3. **People**: Users and operators of hardware and software.

4. **Database**: A large, organized collection of information that is accessed via S/w and persists over time.

5. **Documentation**: manuals, on-line help files.

6. **Procedures**: the steps that define the specific use of each system element.

One complicating characteristic of computer-based system is that the elements constituting one system may also represent one macro element of a still large system. The *micro-element* is a computer-based system that is one part of a larger computer based system.

**The System Engineering Hierarchy:** The key to system engineering is a clear understanding of context. For software development this means creating a "world view" and progressively narrowing its focus until all technical detail is known.

In software engineering there is rarely one right way of doing something. Instead designers must consider the tradeoffs present in the feasible solutions and select one that seems advantageous for the current problem. This section lists several factors that need to be examined by software engineers when evaluating alternative solutions (assumptions, simplifications, limitations, constraints, and preferences).

Regardless of its domain of focus, system eng. Encompasses a collection of top-down and bottom-up methods to navigate the hierarchy illustrated below:



The system eng. process usually begins with a “world view.” The entire business or product domain is examined to ensure that the proper business or technology context can be established. The world view is refined to focus more fully on a specific domain of interest. Within a specific domain, the need for targeted system elements (data, S/W, H/W, and people) is analyzed. Finally, the analysis, design, and construction of a targeted system element are initiated.

**System Modeling:** System modeling is an important element of the system eng. Process. The Engineer creates models that:

1. Define the processes that serve the needs of the view under consideration.

2. Represent the behavior of the processes and the assumptions on which the behavior is based.

3. Explicitly define both exogenous and endogenous input to the model. Exogenous inputs link one constituent of a given view with other constituents at the same level of other levels; endogenous input links individual components of a constituent at a particular view.

4. Represent all linkages (including output) that will enable the engineer to better understand the view.

To construct a system model, the engineers should consider a number of restraining factors:

1. *Assumptions* that reduce the number of possible permutations and variations, thus enabling a model reflect the problem in a reasonable manner.

2. *Simplifications* that enable the model to be created in a timely manner.

3. *Limitations* that help to bound the system.

4. *Constraints* that will guide the manner in which the model is created and the approach taken when the model is implemented.

5. *Preferences* that indicate the preferred architecture for all data, functions, and technology.

***Business Process Engineering***

* The goal of *Business* P*rocess Engineering (BPE)* is to define architectures that will enable a business to use information effectively.
* BPE is one process for creating an overall plan for implementing the computing architecture.
* Three different architectures must be analyzed and designed within the context of business objectives and goals:
	+ - Data architecture
		- Application architecture
		- Technology infrastructure
* **The *data architecture*** provides a framework for the information needs of a business. The building blocks of the architecture are the data objects that are used by the business. Once a set of data objects is defined, their relationships are identified. A *relationship* indicates how objects are connected to one another.
* **The *application architecture*** encompasses those elements of a system that transform objects within the data architecture for some business purpose.
* **The *technology infrastructure*** provides the foundation for the data and application architectures. The infrastructureencompasses the h/w and s/w that are used to support the applications and data.



**BPE Hierarchy**

Information Strategy Planning:

 Management Issues:

* + define strategic business goals/objectives
	+ isolate critical success factors
	+ conduct analysis of technology impact
	+ perform analysis of strategic systems

 Technical Issues:

* + create a top-level data model
	+ cluster by business/organizational area
	+ refine model and clustering

Business Area Analysis: It defines “naturally cohesive groupings of business functions and data”

 Perform many of the same activities as ISP, but narrow scope to individual business area

 Identify existing (old) information systems / determine compatibility with new ISP model

* + define systems that are problematic
	+ defining systems that are incompatible with new information model
	+ begin to establish re-engineering priorities

The Business Area Analysis Process:



**Product Engineering:**

* Emphasize that software engineers participate in all levels of the product engineering process that begins with requirements engineering. The analysis step maps requirements into representations of data, function, and behavior. The design step maps the analysis model into data, architectural, interface, and software component designs.



Product Architecture Template: Proposed by Hatley and Pirbhai, also known as Hatley-Pirbhai modeling



**System Modeling with UML**

* In terms of the data that describe the element and the operations that manipulate the data Deployment diagrams
	+ Each 3-D box depicts a hardware element that is part of the physical architecture of the system



* Activity diagrams
	+ Represent procedural aspects of a system element



* Class diagrams
	+ Represent system level elements



4. Use-case diagrams (Modeling people): Illustrate the manner in which an actor interacts with the system



**PART - III**

**REQUIREMENTS ENGINEERING**

**A BRIDGE TO DESIGN AND CONSTRUCTION:**

Requirement engineering, like all other Software Engineering activities, must be adapted to the process, project, product and the people doing the work. Requirement Engineering begins during the communication activity and continues into the modeling activity. It is essential that the software team make a real effort to understand the requirements of a problem before the team atoms to solve the problem.

Requirement Engineering builds a bridge to design and construction. It allows a software team, to examine,

1) About the **context** of the software work to be performed.

2) The **specific needs** that design the construction must address.

3) The **priorities** that guide the order in which work is to be completed.

4) The **information, functions and behaviors** that will have a profound impact on the

 resultant design.

**REQUIREMENTS ENGINEERING TASKS**

Requirement Engineering Provides an 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 requirements. Requirement engineering tasks are classified as:

**Requirements engineering tasks**

**Seven distinct tasks**

* 1. **Inception**
	2. **Elicitation**
	3. **Elaboration**
	4. **Negotiation**
	5. **Specification**
	6. **Validation**
	7. **Requirements management**
* Some of these tasks may occur in parallel and all are adapted to the needs of the project
* All strive to define what the customer wants
* All serve to establish a solid foundation for the design and construction of the software

**1) Inception:**

 In some cases, for casual conversation all that is needed is to precipitate in a major software engineering effort. At project Inception, Software Engineers ask a set of questions, which establish:

* Basic understanding of the problem
* People who want a solution.
* Nature of solution that is desired.
* Effectiveness of preliminary communication and collaboration between customer and developer.

2) Elicitation: Elicitation is about asking the customers, uses and others “what they want”, i.e., what the objectives of the product are, what is to be accomplished, how the product/ system fits into business needs, and finally how product is to be used on day-to-day basis. **But elicitation is difficult because:**

**a. Problems of Scope**: Boundary of the system is ill-defined, or customers/ users specify unnecessary technical detail that make confuse rather than clarify, overall system objectives.

**b. Problems of Understanding**: Customers or users have a poor understanding of their computing environment, the problem domain etc., specify requirements that conflict with other users needs or specify requirements that are ambiguous or untestable.

**c. Problems of Volatility**: Requirements change over time.

Software engineers overcome these three problems by gathering requirements in an organised manner.

3) **Elaboration:** Information obtained from the customer during Inception and elicitation is expanded and refined during elaboration. It focuses on developing a refined technical model of software functions, features and constraints.

Elaboration is driven by creation and reinforcement of user scenarios that describe how end-user will interact with the system. Each user scenario is parsed to extract analysis classes (business entities) that are visible to end user. The relationships and collaboration between classes are identified and UML diagrams are produced. End-result of elaboration is an analysis model that defines informational, functional and behavioral domain of the problem.

**4) Negotiation:** In this task, customers, users and other stakeholders are asked to rank requirements and then discuss conflicts in priority. Risks associated with each requirement are identified and analysed. Rough “guest mates” of development are made and used to assess impact of each requirement on project cost and delivery time. Measures are taken in negotiations (discussions) so that each party achieves some satisfaction.

**5) Specification:** A specification can be written document, a set of graphical models, a formal mathematical model (algorithm), a collection of usage scenario or a prototype or a combination of these. It is necessary to remain flexible when specification is to be developed. For large systems, a written document is the best approach, whereas for smaller products, usage scenarios are best.

1. Specification is final work product by requirements engineer.

2. It serves as foundation for subsequent Software Engineering activities and describes function, performance and constraints in the product.

**6) Validation:** Work products produced as a result of Requirement Engineering are assessed for quality during validation step. It examines,

1. Specification to ensure all software requirements are stated unambiguously.

2. That inconsistencies, omissions, errors have been detected and corrected.

3. The work products conform to standards established for the process, project and product.

Primary requirement validation mechanism is the Formal Technical Review. The review team consists of Software Engineers, customers, users other stakeholders. Review team examines specification for errors, missing information, inconsistencies, conflicting requirements or unrealistic requirements.

**7) Requirements Management**: “It is a set of activities that help the project team identify, control and track requirements and changes two requirements at any time as the project proceeds.”

It begins with identification; each requirement is assigned a unique identifier. Once requirement have been identified, traceability/ feasibility tables are developed. Each traceability table relates requirements to one or more aspects of the system or its environment.

Possible traceability tables are:

1. Features traceability table: shows how requirements relate to important customer observable system/ product features.

2. Source traceability table: Identifies source of each requirement.

3. Dependency traceability table: Indicates how requirements are related to one another.

4. Subsystem traceability table: Categorises requirements by the subsystem(s) that they govern.

5. Interface traceability table: Shows how requirements relate to both internal and external system interfaces.

These traceability tables are maintained in Requirements Database to understand how a change in requirements will affect different aspects of the system to be built.

**INCEPTION (OR) INITIATING REQUIREMENT ENGINEERING PROCESS**:

To get the project started and forward towards a successful solution, we need the following steps to initiate Requirement Engineering:

**1. Identifying the Stakeholders**: stakeholder is defined as “anyone who benefits in a direct or indirect way from the system which is being developed.” Each stakeholder has a different view of the system, achieves different benefits when system is successfully developed and is open to different risks.

At inception, requirement engineer should create list of people who will input as requirements are elicited. The initial list will grow as stakeholders are contacted further.

**2. Recognizing Multiple Viewpoints**: As different stakeholders exist; requirements of the system will be explored from many different points of the view. Each of various constituencies such as marketing groups, business managers, end users, Software Engineers will contribute information to the Requirements Engineering process. For ex. support engineer may focus on the software maintainability.

The job of requirements engineer is to categorize all stakeholder information including inconsistent and conflicting requirements in a way that will allow decision makers to choose a consistent set of requirements for the system.

**3. Working towards Collaboration**: Customers should collaborate among themselves and with Software Engineers to result a successful system. The job of requirement engineer is to identify areas of commonality and areas of conflict or inconsistency.

In many cases, stakeholders collaborate by providing their view of requirements, but a strong “project champion” (Ex: business manager) may make the final decision about which requirements make the cut.

**4. Asking the first question**: The questions asked at the Inception of the project should be “context free”. The first set of questions focus on customer and other stakeholders, overall goals, and benefits.

For ex: requirements engineering might ask:

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

These questions help to identify all stakeholders who will have interest in software to be built. These also identify measurable benefits of successful implementation and alternatives for software development.

* Next set of questions include:
* What problems will this solution address?
* How would you characterize “good output”?
* Can you show me the environment where solution will be used?
* Will special performance issues or constraints affect the way the solution is approached?

These questions enable software team to gain better understanding of the problems and allows customer to say his/her perceptions.

Final set of questions are:

* Are you the right person and are your answers “official”?
* 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?

These questions focus on effectiveness of communication. These are also called as meta- questions. All these questions will help to “break the ice” and initiate the communication that is essential for successful elicitation.

**ELICITING REQUIREMENTS**

The Q&A session should be used for the first encounter only and then replaced by a requirements elicitation format.

**Collaborative Requirements Gathering:** Many different approaches to collaborative requirements gathering have been proposed and each follows the basic guidelines:

1. Meetings are conducted and attended by both Software Engineers, customers along with stakeholders.

2. Rules for preparation and participation are established.

3. An agenda is suggested that is formal enough to cover all important points but informal enough to encourage free flow of ideas.

4. A “facilitator” (customer/ developer/ outsider) controls the meeting.

5. A “definition mechanism” (worksheets etc) can be used.

6. The goal is to

a. Identify the problem.

b. Propose elements of the solution

c. Negotiate different approaches

d. Specify preliminary set of solution requirements.

During Inception the stakeholders write a “one or two page request”. A meeting place, time, date and a facilitator are selected. Then the product request is distributed to all attendees before the meeting date, and ask to go through the product request and make suggestions in the meeting.

**Quality Function Deployment:** QFD is a technique that translates the needs of customer into the technical requirements for software. QFD “concentrates on maximizing customer satisfaction from the Software Engineering process”. QFD identifies three types of requirements:

**1. Normal Requirements:** These reflect objectives and goals stated for a product during meetings with the customer. If these requirements are present then the customer is satisfied.

**2. Expected Requirements:** These are implicit to the product and customer does not explicitly state them. Their absence will cause significance dissatisfaction.

**3. Exciting Requirements:** These reflect features that go beyond customer’s expectations and prove to be very satisfying when present.

*  In meetings with the customer, Function Deployment determines value of each function that is required for the system.
*  Information Deployment identifies both data objects and events that system must consumer and produce.
*  Task Deployment examines behaviour of the system within context of its environment.
*  Value Analysis is conducted to determine relative priority of the requirements determined during each of three deployments.

 QFD uses customer interviews, observations, surveys and examination of historical data as raw data for requirements gathering activity. These data are then translated into a table of requirements, called as the Customer voice table that is reviewed with customer.

**User Scenarios:** Developers and users create a set of scenarios that identify a trend of usage for the system to be constructed. These scenarios often called use cases provide a description of how the system will be used.

Elicitation Work Products: These depend on the size of the system/product to be built. These include:

*  A statement of need or feasibility.
*  A bounded statement of scope for system or product.
*  A list of customers, users and stakeholders.
*  A description of system’s technical environment.
*  A list of requirements and constraints that apply.
*  A set of usage scenarios that provide insight into use of the system.
*  Any prototypes developed to better define requirements.

ELABORATION

Developing Use Cases: “Use case is defined as set of sequence of actions performed by an actor to achieve a specific result”. An actor refers to various people that use system or product within context of the function.



Use Case is a collection of user scenarios that describe the thread of usage of a system. 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:

* 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 extensions 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?

**Building the Analysis Model:** The intent of the analysis model is to provide a description of the functional, informational and behavioural requirements for a computer-based system.

 Analysis model is a snapshot of requirements at any given time. We expect it to change. As the analysis model evolves, certain elements will become relatively stable and other elements may be more volatile, indicating customer does not yet understand requirements for system.

**Elements of Analysis Model:** The specific elements of the analysis model are dictated by analysis modeling method. These elements include:

**1. Scenario-based elements:** These are often the first part of analysis model that is developed. They serve as input (or) informational requirements for creation of other modeling elements.

 A variation in scenario based modeling depicts activities (functions/operations) that have been defined as a part of requirement elicitation task, i.e., sequence of activities is defined as part of analysis model. Activities can be represented iteratively at different levels of abstraction by using activity diagrams (or) use case diagrams.

As an example, consider UML diagrams for eliciting requirements.

 

2. Class-based elements:

Each usage scenario implies a set of “Objects” that are manipulated as an actor interacts with the system. These objects are categorised into “Classes”- a collection of things that have similar attributes and common behaviour.

 **A class diagram** usually represents the functional requirements in analysis model. Analysis model may also depict the manner in which classes collaborate with one another and relationships between them. An example for class diagram is,

 

**3. Behavioral Elements**: The behaviour of a product can have a profound effect on design and implementation approach. [i.e., static or dynamic].

 State diagram is used to represent behaviour of a system buy depicting its states and events that cause the system to change state. A state is any observable mode of behaviour.

 A state diagram indicates what actions are taken as a consequence of a particular event. A state diagram is given as,

 

4. Flow oriented elements: Information/Data is transformed as it flows through a computer-based system. System accepts input in a variety of forms, applies functions to transform it, and produces output in a variety of forms. This data flow is depicted using DFDs (Data Flow Diagrams).



Analysis Patterns: Analysis patterns represent the things (class, function or behaviour) that can be reused when modeling many applications. Analysis patterns are integrated into analysis model by reference to the pattern name. They are also stored in a repository so that Requirement Engineers can reuse them.

Analysis pattern template includes:

1. Pattern name: A descriptive that captures essence of pattern.

2. Intent: Describes what pattern accomplishes or represents and/or what problem is addressed.

3. Motivation: A scenario that illustrates how pattern can be used to address the problem.

4. Forces and context: Description of external issues that can affect how pattern is used and how they will be resolved.

5. Solution: Description of how pattern is applied to solve problem.

6. Consequences: Address what happens when pattern is applied.

7. Design: Discusses how analysis pattern can be achieved through use of known design patterns.

8. Known uses: Examples of uses within actual systems.

9. Related patterns: One or more analysis patterns that are related to named pattern because the analysis pattern,

a. is commonly used with named pattern

b. is structurally similar to the named pattern

c. is a variation of named pattern.

**NEGOTIATING REQUIREMENTS**

Customer and developer enter into a process of negotiation, where they will have a discussion about balancing functionality, performance, and other product or system characteristics against cost and time to market.

The best negotiations strive for a **“win-win”** result. i.e., customer wins by getting system/ product that satisfies the needs, and software team wins by working to realistic and achievable budget and deadlines.

Boehm defines a set of negotiation activities:

1. Identification of system/ subsystem key stakeholders.

2. Determination of stakeholders “Win conditions”.

3. Negotiate stakeholders win conditions to reconcile them into a set of **win-win** conditions for all concerned (including software team).

**VALIDATING REQUIREMENTS**

Requirements are validated in this task by a review of customer requirements. A review of analysis model addresses the following questions:

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