SE - UNIT-2

UNIT II: Chapter 1: Agility, Agility and the Cost of Change, Agile Process, Extreme Programming (XP), Other Agile Process Models, A Tool Set for the Agile Process,
Chapter 2: Software Engineering Knowledge, Core Principles, Principles That Guide Each Framework Activity
Chapter 3: Requirements Engineering, Establishing the Groundwork, Eliciting Requirements, Developing Use Cases, Building the Requirements Model, Negotiating Requirements, Validating Requirements.

Chapter 1

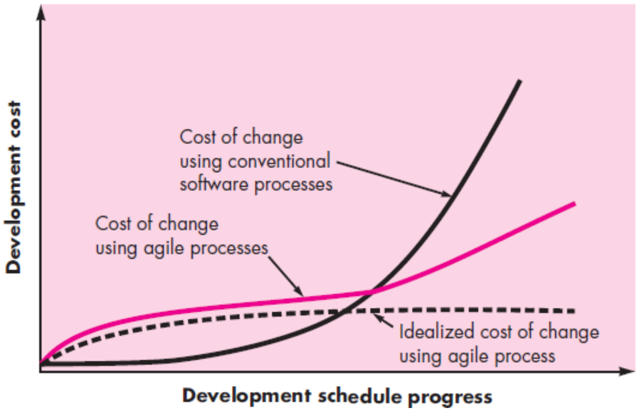
- Agile software engineering has a set of development guidelines. The mechanism encourages customer satisfaction and early incremental delivery of software.
- Software engineers and other project stakeholders (managers, customers, end users) work together on an agile team
- Agile software engineering represents a reasonable alternative to conventional software engineering .

Agility:

- Agility means effective (rapid and adaptive) response to change, effective communication among all stakekholder.
- Drawing the customer onto team and organizing a team so that it is in control of work performed. The Agile process is light-weight methods and People-based rather than plan-based methods.
- The agile process forces the development team to focus on software itself rather than design and documentation.
- The agile process believes in iterative method.
- The aim of agile process is to deliver the working model of software quickly to the customer For example: Extreme programming is the best known of agile process.

AGILITY AND THE COST OF CHANGE

The cost of change increases as the project progresses.(Figure 3.1, solid black curve). It is relatively easy to accommodate a change when a software team is gathering requirements (early in a project).



The team is in the middle of validation testing (something that occurs relatively
late in the project), and stakeholder is requesting a major functional change. The change requires a modification to the architectural design of the software, the design and construction of three new components, modifications to another five components, the design of new tests, and so on. Costs and time increases quickly.

A well-designed agile process "flattens" the cost of change curve (Figure 3.1, shaded, solid curve), allowing a software team to accommodate changes late in a software project without cost and time impact. the agile process encompasses incremental delivery.

What is an Agile Process:

Any agile software process is characterized in a manner that addresses a number of key assumptions about the majority of software projects:

1. It is difficult to predict(estimate for future change) in advance which software requirements will persist and which will change.
2. For many types of software, design and construction are performed simultaneously. It is difficult to predict how much design is necessary before construction is used to prove the design.
3. Analysis, design, construction, and testing are not expected. (from a planning point of view)

Agility Principles

12 agility principles for those who want to achieve agility:
1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.

3. Deliver working software frequently, from a couple of weeks to a couple of Months.
4. stake holders and software engineers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.
9. Continuous attention to technical excellence and good design enhances(increases)agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective and then adjusts its behavior accordingly.
Not every agile process model applies these 12 principles with equal weight, and some models choose to ignore (or at least downplay) the importance of one or more of the principles.

The Politics of Agile Development

- There is debate about the benefits and applicabilityof agile software development as opposed to more conventional software engineering processes(produces documents rather than working product).
- Even within the agile, there are many proposed process models each with a different approach to the agility .

Human Factors

Agile software development take the importance of "people factors". Number of different talents must exist among the people on an agile team and the team itself:
Competence:"competence" encompasses talent, specific software-related skills, and overall knowledge of the process .
Common focus: Members of the agile team may perform different tasks and all should be focused on one goal—to deliver a working software increment to the customer within the time promised.
Collaboration: Team members must collaborate with one another and all other

Stakeholders to complete the their task.

Decision-making ability: Any good software team (including agile teams) must be allowed the freedom to control its own destiny.

Fuzzy problem-solving ability: The agile team will continually have to deal with ambiguities(confusions or doubts).

Mutual trust and respect: The agile team exhibits the trust and respect .

Self-organization:

(1) the agile team organizes itself for the work to be done

(2) the team organizes the process to best accommodate its local environment

(3) the team organizes the work schedule to best achieve delivery of the software increment.

Extreme Programming:

Extreme Programming (XP), the most widely used approach to agile software development. XP proposed by kent beck during the late 1980's.

XP Values

Beck defines a set of five *values* —communication, simplicity, feedback, courage, and respect. Each of these values is used in XP activities, actions, and tasks.

- Effective ***communication*** between software engineers and

other stakeholders .

- . To achieve ***simplicity,*** XP restricts developers to design only for immediate needs, rather than future needs.
- ***Feedback*** is derived from three sources: the implemented software itself, the customer, and other software team members
- ***courage.(discipline)*** An agile XP team must have the discipline (courage) to design for today, recognizing that future requirements may change dramatically.
- the agile team inculcates ***respect*** among it members, between other stakeholders and team members.

The XP(Extreme Programming) Process

XP Process have four framework activities: planning, design, coding, and testing.

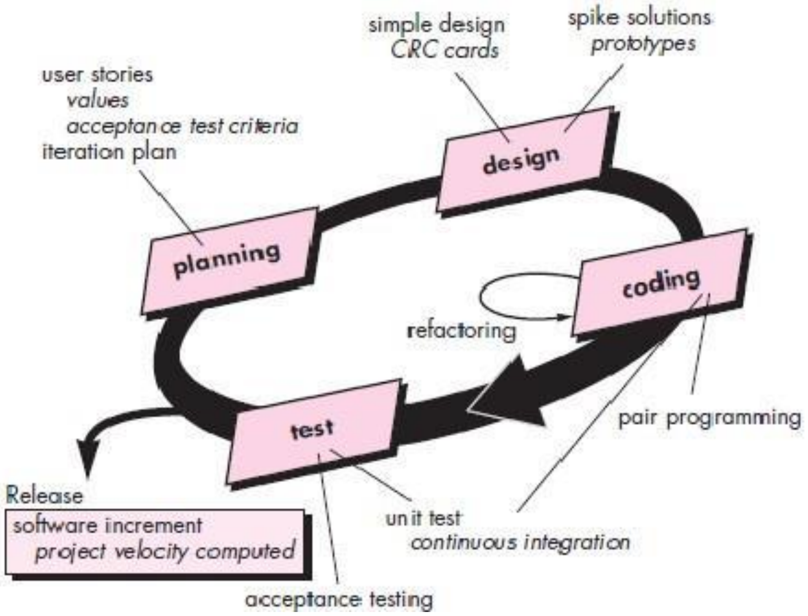
Planning. The planning activity begins with *listening*—a requirements gathering activity.

- Listening leads to the creation of a set of "stories" (also called *user stories*) that describe required output, features, and functionality for software to be built.

- Each *story* is written by the customer and is placed on an index card.The customer assigns a *value* (i.e., a priority) to the story based on the overall business value of the feature or function.
- Members of the XP team then assess each story and assign a *cost*—measured in development weeks—to it.
- If the story is estimated to require more than three development weeks,the story into smaller stories and the assignment of value and cost occurs again. It is important to note that new stories can be written at any time.
- the stories with highest value will be moved up in the schedule and implemented first.

FIGURE 3.2

The Extreme Programming process



Design: XP design follows the KIS (keep it simple) principle.

- If a difficult design problem is encountered as part of the design of a story, XP recommends the immediate creation of an operational prototype of that portion of the design. Called a *spike solution.*
- XP encourages *refactoring*—a construction technique that is also a method for design optimization.
- Refactoring is the process of changing a software system in a way that it does not change the external behavior of the code and improves the internal structure.

Coding:design work is done, the team does *not* move to code, develops a series of unit tests for each of the stories that is to be included in the current release (software increment).

- Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the test.
- Once the code is complete, it can be unit-tested immediately,and providing feedback to the developers.
- A key concept during the coding activity is *pair programming*. i.e..,two people work together at one computer workstation to create code for a story.
- As pair programmers complete their work, the code they develop is integrated with the work of others.

Testing:

- As the individual unit tests are organized into a “universal testing suite” integration and validation testing of the system can occur on a daily basis.
- XP *acceptance tests*, also called *customer tests,* are specified by the customer and focus on overall system features and functionality. Acceptance tests are derived from user stories that have been implemented as part of a software release.

Industrial XP

Joshua Kerievsky describes *Industrial Extreme Programming* (IXP)

IXP has six new practices that are designed to help XP process works successfully for projects within a large organization.

Readiness assessment. the organization should conduct a *readiness assessment.*

(1) Development environment exists to support IXP(Industrial Extreme Programming).

(2) the team will be populated by the proper set of stakeholders.

(3) the organization has a distinct quality program and supports continuous improvement.

(4) the organizational culture will support the new values of an agile Team.

Project community.

- Classic XP suggests that the right people be used to populate the agile team to ensure success.
- The people on the team must be well-trained, adaptable and skilled.
- A community may have a team members and customers who are central to the success of the project.

Project chartering.:

- Chartering examines the context of the project to determine how it complements, extends, or replaces existing systems or processes.

Test-driven management.

- Test-driven management establishes a series of measurable "destinations" and then defines mechanisms for determining whether or not these destinations have been reached.

Retrospectives.

- An IXP team conducts a specialized technical reviews after a software increment is delivered. Called a *retrospective.*
- The review examines "issues, events, and lessons-learned" across a software increment and/or the entire software release.

Continuous learning.

- learning is a vital part of continuous process improvement, members of the XP team are encouraged to learn new methods and techniques that can lead to a higher quality product.

The XP Debate

Requirements volatility. The customer is an active member of the XP team, changes to requirements are requested informally. As a consequence, the scope of the project can change and earlier work may have to be modified to accommodate current needs.

• ***Conflicting customer needs*.** Many projects have multiple customers, each with his own set of needs.

• ***Requirements are expressed informally*.** User stories and acceptance tests are the only explicit manifestation of requirements in XP. specification is often needed to remove inconsistencies, and errors before the system is built.

• ***Lack of formal design*:**when complex systems are built, design must have the overall structure of the software then it will exihibit quality.

Other Agile Process Models:

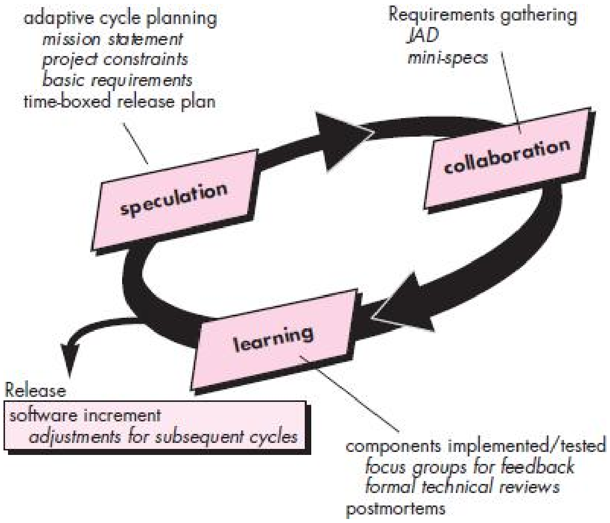
The most widely used of all agile process model is Extreme Programming (XP). But many other agile process models have been proposed and are in use across the industry. Among the most common are:

- Adaptive Software Development (ASD)
- Scrum
- Dynamic Systems Development Method (DSDM)
- Crystal
- Feature Drive Development (FDD)

- Lean Software Development (LSD)
- Agile Modeling (AM)
- Agile Unified Process (AUP)

Adaptive Software Development (ASD)*Adaptive Software Development* (ASD) has been proposed by Jim High smith as a technique for building complexsystems. ASD focus on human collaboration and team self-organization.

- ASD "life cycle" (Figure 3.3) has three phases:- speculation, collaboration, and learning.
- ***speculation,*** the project is initiated and *adaptive cycle planning* is conducted.Adaptive cycle planning uses project initiation information—the customer's statement, project constraints (e.g., delivery dates or user descriptions), and basic requirements—to define the set of release cycles (software increments) that will be required for the project.



- Motivated people use ***collaboration***
- People working together must trust one another to (1) without criticize, (2) work as hard as or harder than they do, (3) have the skill set to contribute to the work and (4) communicate problems in away that leads to effective action.
- **learning** will help them to improve their level of real understanding.

Scrum

- Scrum is an agile software development method that was coined by Jeff Sutherland and his development team in the early 1990's.
- Scrum has the following framework activities: requirements, analysis, design, evolution, and delivery. Within each framework activity, actions and work tasks occur within a process called a *sprint.*

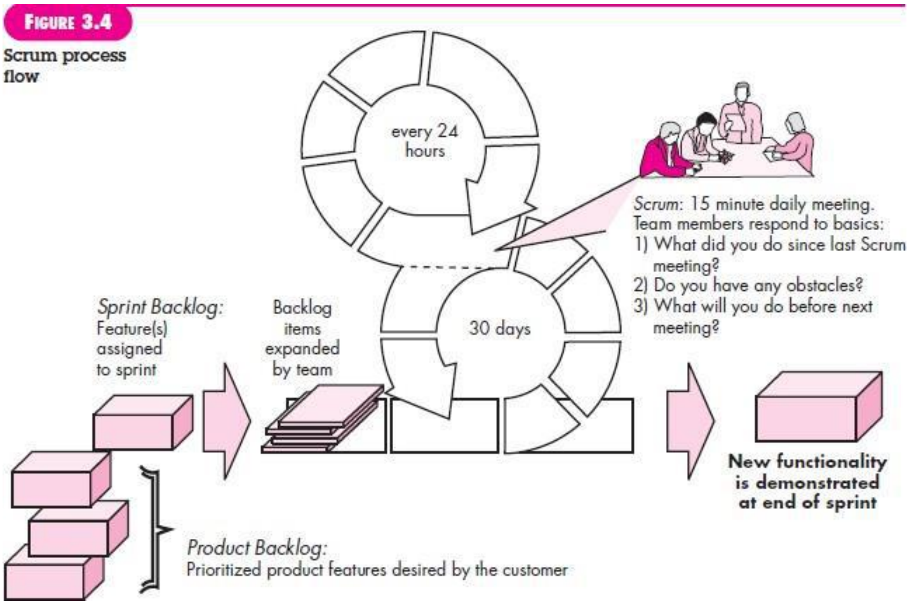
- scrum defines a set of development actions:

Backlog—a prioritized list of project requirements or features that provide business value for the customer. Items can be added to the backlog at any time (this is how changes are introduced). The product manager assesses the backlog and updates priorities as required.

Sprints—consist of work units that are required to achieve a requirement defined in the backlog .

FIGURE 3.4

Scrum process flow


every 24 hours
Scrum: 15 minute daily meeting.
Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog:
Feature(s) assigned to sprint
Backlog items expanded by team
30 days
New functionality is demonstrated at end of sprint
Product Backlog:
Prioritized product features desired by the customer

Scrum meetings—are short (typically 15 minutes) meetings held daily by the Scrum team. Three key questions are asked and answered by all team members.

• What did you do since the last team meeting?

• What obstacles are you encountering?

• What do you plan to accomplish by the next team meeting?

A team leader, called a *Scrum master,* leads the meeting and assesses the responses from each person.

Demos—deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.

Dynamic Systems Development Method (DSDM)

- The *Dynamic Systems Development Method* (DSDM) is an agile software development approach.
- The DSDM—80 percent of an application can be delivered in 20 percent of the time it would take to deliver the complete (100 percent) application.

- DSDM is an iterative software process in which each iteration follows the 80 percent rule. That is, only enough work is required for each increment to facilitate movement to the next increment.
- The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.
- The activities of DSDM are:

Feasibility study—establishes the basic business requirements and constraints associated with the application to be built and then assesses whether the application is a viable candidate for the DSDM process.

Business study—establishes the functional and information requirements that will allow the application to provide business value.

Functional model iteration—produces a set of incremental prototypes that demonstrate functionality for the customer.

- The intent during this iterative cycle is to gather additional requirements by eliciting feedback from users as they exercise the prototype.

Design and build iteration—prototypes built during *functional model iteration* to ensure that each has been engineered in a manner that it will provide operational business value for end users.

Implementation—places the latest software increment (an "operationalized" prototype) into the operational environment. It should be noted that (1) the increment may not be 100 percent complete or (2) changes may be requested as the increment is put into place.

Crystal

- Alistair Cockburn and Jim Highsmith created the *Crystal family of agile methods* .
- Cockburn characterizes as "a resource limited, cooperative game of invention and communication, with a primary goal of delivering useful, working software and a secondary goal of setting up for the next game".
- Cockburn and Highsmith have defined a set of methodologies, each with core elements that are common to all, and roles,process patterns, work products, and practice that are unique to each.
- The Crystal family is actually a set of example agile processes that have been proven effective for different types of projects.

Feature Driven Development (FDD)

- *Feature Driven Development* (FDD) was originally coined by Peter Coad and his colleagues as a process model for object-oriented software engineering.

- A *feature* “is a client-valued function that can be implemented in two weeks or less”.
- the definition of features provides the following benefits:
  - features are small blocks of deliverable functionality, users can describe them more easily.
  - Since a feature is the FDD deliverable software increment, the team develops operational features every two weeks.
  - Because features are small, their design and code representations are easier.
- the following template for defining a feature:

<action> the **<result> <by for of to>** a(n) **<object>**

Where

<object> is “a person, place, or thing.”

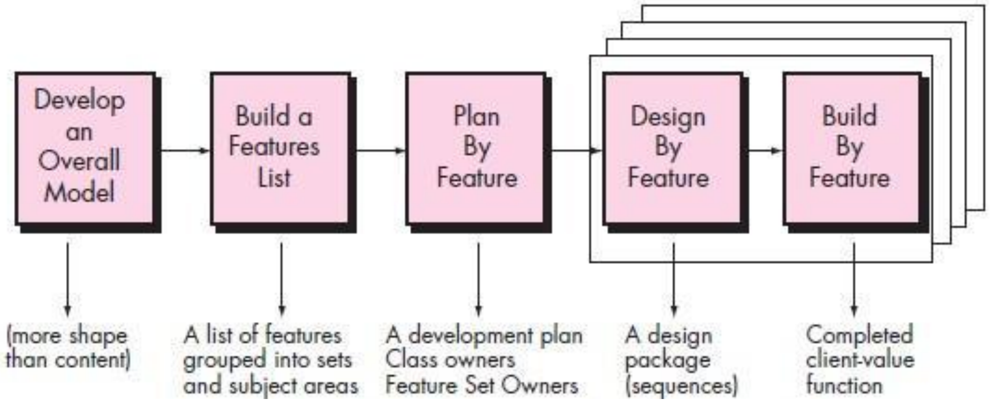
Examples of features for an **e-commerce application** might be:

1)Add the product to shopping cart

2)Store the shipping-information for the customer

FIGURE 3.5

Feature Driven Development [Coa99] (with permission)


Develop an Overall Model
Build a Features List
Plan By Feature
Design By Feature
Build By Feature
(more shape than content)
A list of features grouped into sets and subject areas
A development plan Class owners Feature Set Owners
A design package (sequences)
Completed client-value function

- A feature set groups related features into business-related categories and is defined as:

<action><-ing> a(n) **<object>**

For example: *Making a product sale* is a feature set that would encompass the features noted earlier and others.

- The FDD approach defines five “collaborating” framework activities as shown in **Figure 3.5.**
- It is essential for developers, their managers, and other stakeholders to understand project status.

- For that FDD defines six milestones during the design and implementation of a feature: "design walkthrough, design, design inspection, code, code inspection, promote to build".

Lean Software Development (LSD)

- *Lean Software Development* (LSD) has adapted the principles of lean manufacturing to the world of software engineering.
- LSD process can be summarized as *eliminate waste, build quality in, create knowledge, defer commitment, deliver fast, respect people,* and *optimize the whole.*
- For example,

eliminate waste within the context of an agile software project as
1)adding no extraneous features or functions
(2) assessing the cost and schedule impact of any newly requested requirement,
(3) removing any superfluous process steps,
(4) establishing mechanisms to improve the way team members find information,
(5) ensuring the testing finds as many errors as possible,

Agile Modeling (AM)

Agile Modeling(AM) suggests a wide array of "core" and "supplementary" modeling principles,

Model with a purpose. A developer who uses AM should have a specific goal (e.g., to communicate information to the customer or to help better understand aspect of the software) in mind before creating the model.

Use multiple models. There are many different models and notations that can be used to describe software.

- Each model should present a different aspect of the system and only those models that provide value to their intended audience should be used.

Travel light. As software engineering work proceeds, keep only those models that will provide long-term value .

Content is more important than representation. Modeling should impart information to its intended audience.

Know the models and the tools you use to create them. Understand the strengths and weaknesses of each model and the tools that are used to create it.

Adapt locally. The modeling approach should be adapted to the needs of the agile team.

Agile Unified Process (AUP)

The *Agile Unified Process* (AUP) adopts a "serial in the large" and "iterative in the small" philosophy for building computer-based systems.

Each AUP iteration addresses the following activities:

• ***Modeling.*** UML representations of the business and problem domains are created.

• ***Implementation.*** Models are translated into source code.

• ***Testing.*** Like XP, the team designs and executes a series of tests to uncover errors and ensure that the source code meets its requirements.

• ***Deployment.*** focuses on the delivery of a software increment and the acquisition of feedback from end users.

• ***Configuration and project management.***configuration management addresses change management, risk management,and the control of any persistent work products that are produced by the team.

Project management tracks and controls the progress of the team and coordinates team activities.

• ***Environment management.*** Environment management coordinates a process infrastructure that includes standards, tools, and other support technology available to the team.

A tool set for agile development

- Automated software tools (e.g., design tools) should be viewed as a minor supplement to the team's activities, and not to the success of the team.
- Collaborative and communication "tools" are generally low technology and incorporate any mechanism(, whiteboards, poster sheets) that provides information and coordination among agile developers.
- other agile tools are used to optimize the environment in which the agile team works ,improve the team culture by social interactions, physical devices (e.g., electronic whiteboards), and process enhancement (e.g., pair programming or time-boxing)"

Chapter 2

- *practice* is a collection of concepts, principles, methods, and tools that a software engineer calls upon on a daily basis.
- Practice allows managers to manage software projects and software engineers to build computer programs.
- Practice populates a software process model with the necessary technical and management how-to's to get the job done.

Software Engineering knowledge:

Steve McConnell made the following comment:

- Many software practitioners(developers) think of software engineering knowledge as knowledge of specific technologies: Java, Perl, html, C__, and so on. Knowledge of specific technology details is necessary to perform computer programming.
- software engineering knowledge had evolved to a “stable core”and represented as “75 percent of the knowledge needed to develop a complex system.”
- As McConnell indicates, core principles are the elemental ideas that guide the software engineers in the work that they do.It provide a foundation from which software engineering models, methods, and tools can be applied and evaluated.

Core principles

- Software engineering is guided by a collection of core principles that help in the application of a meaningful software process and the execution of effective software engineering methods.
- At the process level, core principles establish a foundation that guides a software team as it performs framework and umbrella activities,navigates the process flow, and produces a set of software engineering work products.
- At the level of practice, core principles establish a collection of values and rules that serve as a guide to gather the requirements, analyze a problem, design a solution, implement and test the solution, and deploy the software in the user community.

Principles That Guide Process

The following set of core principles can be applied to the framework, and by extension, to every software process.

Principle 1. ***Be agile.*** Whether the process model you choose is prescriptive or agile.

- keep your technical approach as simple as possible
- keep the work products you produce as concise(short) as possible
- Make decisions locally whenever possible.

Principle 2. ***Focus on quality at every step.*** For every process activity, action, and task should focus on the quality of the work product that has been produced.

Principle 3. ***Be ready to adapt.*** adapt your approach to conditions imposed by the problem, the people, and the project itself.

Principle 4. ***Build an effective team.*** Build a self-organizing team that has mutual trust and respect.
Principle 5. ***Establish mechanisms for communication and coordination.***

Projects fail because stakeholders fail to coordinate their efforts to create a successful end product.
Principle 6. ***Manage change.*** The methods must be established to manage the way changes are requested, approved, and implemented.
Principle 7. ***Assess risk.*** Lots of things can go wrong as software is being developed.
Principle 8. ***Create work products that provide value for others.***
Create only those work products that provide value for other process activities, actions and tasks.

Principles That Guide Practice

- Software engineering practice has a single goal i.e..,to deliver on-time, high quality, operational software that contains functions and features that meet the needs of all stakeholders.
- To achieve this goal, should adopt a set of core principles that guide the technical work.
- The following set of **core principles** are fundamental to the practice of software engineering:

Principle 1. ***Divide and conquer.*** A large problem is easier to solve if it is subdivided into a collection of elements(or *modules or components*).

- Ideally, each element delivers distinct functionality that can be developed.

Principle 2. ***Understand the use of abstraction.*** At its core, an abstraction(overview) is a simplification of some complex element of a system used to communicate meaning in a single phrase.
Principle 3. Strive for consistency. Whether it's creating a requirements model, developing a software design, generating source code, or creating test cases. All these are consistent so that the software is easier to develope.
.**Principle 4.** ***Focus on the transfer of information.*** Software is about information transfer—from a database to an end user, from an operating system to an application,

Principle 5. ***Build software that exhibits effective modularity.***
Modularity provides a mechanism for any complex system can be divided into modules (components).
Principle 6. ***Look for patterns.*** Brad Appleton suggests that:

The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development process.

Principle 7. ***When possible, represent the problem and its solution from a number of different perspectives.*** When a problem and its solution are examined from a number of different perspectives(ways).

Principle 8. ***Remember that someone will maintain the software.*** software will be corrected as defects are remove, adapted as its environment changes, and enhanced as stakeholders request more capabilities.

Principles That Guide The Each Framework Activity:

Communication Principles

customer requirements must be gathered through the communication activity. Communication has begun.

Principle 1. ***Listen.*** Try to focus on the speaker's(stake holder) words, rather than response to those words. Ask for clarification if something is unclear, but avoid constant interruptions(disturbances).

Principle 2. ***Prepare before you communicate.*** Spend the time to understand the problem before meet with others.

Principle 3. ***Someone should facilitate the activity.***

- Every meeting should have a leader (a facilitator) to keep the conversation moving in a productive direction.
- to mediate any conflict(disturbances) that does occur.

Principle 4. ***Face-to-face communication is best.*** it usually works better when some other representation of the relevant information is present. For example,A document having all the details..

Principle 5. ***Take notes and document decisions.*** Someone participating in the communication should serve as a "recorder" and write down all important points and decisions.

Principle 6. ***Strive for collaboration.*** Each small collaboration serves to build trust among team members and creates a common goal for the team.

Principle 7. ***Stay focused; modularize your discussion.*** The more people involved in any communication, the more likely that discussion will move from one topic to the next. The facilitator should keep the conversation modular, leaving one topic only after it has been resolved.

Principle 8. ***If something is unclear, draw a picture.*** A sketch or drawing can often provide clarity when words fail to do the job.

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.***

If the team has collaborated well, all parties have a common goal. Still, negotiation will demand compromise from all parties.

Planning Principles

- The communication activity helps you to define your overall goals and objectives.
- The planning activity encompasses a set of management and technical practices that enable the software team to define a road map as it travels toward the objectives.the following principles always apply:

Principle 1. ***Understand the scope of the project***:Scope provides the software team with a destination.

Principle 2. ***Involve stakeholders in the planning activity.*** Stakeholders define priorities for which feature to develop first and establish project constraints(conditions).

Principle 3. ***Recognize that planning is iterative.***

- As work begins, it is very likely that things will change.
- The plan must be adjusted to accommodate these changes.

Principle 4. ***Estimate based on what you know.*** The estimation is to provide an indication of effort, cost, and task duration, based on the team's current understanding of the work to be done.

Principle 5. ***Consider risk as you define the plan.*** If you have identified risks that have high impact and high probability, planning is necessary.

Principle 6. ***Be realistic.*** People don't work 100 percent of every day.

- Noise always enters into any human communication.
- Change will occur. Even the best software engineers make mistakes.
- These should be considered as a project plan is established.

Principle 7. ***Adjust granularity as you define the plan.*** *Granularity* refers to the level of detail that is introduced as a project plan is developed.

Principle 8. ***Define how you intend to ensure quality.*** The plan should identify how the software team intends to ensure quality.

Principle 9. ***Describe how you intend to accommodate change.*** Even the best planning can be obviated by uncontrolled change. You should identify how changes are to be accommodated as software engineering work proceeds.

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

- Software projects fall behind schedule one day at a time.
- To track progress on a daily basis, looking for problem areas and situations in which scheduled work does not conform to actual work conducted.

Modeling Principles

Create models to gain a better understanding of the actual entity to be built.
The modeling principles are:

Principle 1. ***The primary goal of the software team is to build software, not create models.*** Agility means getting software to the customer in the fastest possible time.

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

- Every model that is created must be kept up-to-date as changes occur.
- Create only those models that make it easier and faster to construct the software.

Principle 3. ***Strive to produce the simplest model that will describe the problem or the software.***
software that is easier to integrate, easier to test, and easier to maintain.

Principle 4. ***Build models in a way that makes them amenable to change.***
The problem with this attitude is that without a reasonably complete requirements model, you'll create a design (design model) that will invariably miss important functions and features.

Principle 5. ***Be able to state an explicit purpose for each model that is created.*** Every time you create a model, ask yourself why you're doing so. If you can't provide solid justification for the existence of the model, don't spend time on it.

Principle 6. ***Adapt the models you develop to the system at hand.*** It may be necessary to adapt model notation .

Principle 7. ***Try to build useful models, but forget about building perfect models.***
When building requirements and design models, that the models should be complete and consistent .

Principle 8. ***Don't become dogmatic about the syntax of the model. If it communicates content successfully, representation is secondary.***
Although everyone on a software team should try to use consistent notation during modeling.

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. you are an experienced software engineer, trust your instincts(friends).If something tells you that a design model is to fail (even though you can't prove it explicitly), then to spend additional time examining the model or developing a different one.

Principle 10. ***Get feedback as soon as you can.*** Every model should be reviewed by members of the software team. These reviews is to provide feedback that can be used to correct modeling mistakes.

Requirements modeling principles. Investigators have been identified requirements analysis problems and their causes and have developed a variety ofmodeling notations and corresponding sets of methods to overcome them.Each analysis method has a unique point of view. All analysis methods are related by aset of operational principles:

Principle 1. ***The information domain of a problem must be represented and understood.*** The *information domain* encompasses the data that flow into the system, the data that flow out of the system, and the data stores that collect and organize persistent data objects

Principle 2. ***The functions that the software performs must be defined.*** Software functions provide direct benefit to end users

Principle 3. ***The behavior of the software (as a consequence of external events) must be represented.*** The behavior of computer software is driven by its interaction with the external environment.

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.***

Develope the user defined functions using programming languages.

Design Modeling Principles.

- The design model that is created for software provides a variety of different views of the system.
- Set of design principles that can be applied regardless of the method that is used:

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

- The requirements model describes the information domain of the problem,user-visible functions, system behavior.

- The design model translates this information into an architecture.(diagrams)
- The elements of the design model should be traceable to the requirements model.

Principle 2. ***Always consider the architecture of the system to be built.*** Software architecture is the skeleton of the system to be built. It affects interfaces, data structures, program control flow and behavior.

Principle 3. ***Design of data is as important as design of processing functions.*** Data design is an essential element of architectural design. A well-structured data design helps to simplify program flow, makes the design and implementation of software components easier.

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

A well-designed interface (communication between components)makes integration easier and assists the tester in validating component functions.

Principle 5. ***User interface design should be tuned to the needs of the end user. However, in every case, it should stress ease of use.*** The user interface is the visible manifestation of the software.

Principle 6. ***Component-level design should be functionally independent.*** Functional independence is a measure of the "single-mindedness" of a software component. The functionality that is delivered by a component should be cohesive—that is, it should focus on one and only one function .

Principle 7. ***Components should be loosely coupled to one another and to the external environment.*** Coupling is achieved in many ways—via a component interface, by messaging, through global data. As the level of coupling increases, the likelihood of error propagation also increases .

Principle 8. ***Design representations (models) should be easily understandable.*** The purpose of design is to communicate information to practitioners(software developers,testers,maintainers).

Principle 9. ***The design should be developed iteratively. With each iteration, the designer should strive for greater simplicity.***

The first iterations work to refine the design and correct errors, but later iterations should strive to make the design as simple as is possible.

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. 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.

- The initial focus of testing is at the component level, often called ***unit testing.***
- Other levels of testing include
  i)*integration testing* (conducted as the system is constructed).
  ii)*validation testing* that assesses whether requirements have been met for the complete system (or software increment)
  (iii) *acceptance testing* that is conducted by the customer in an effort to exercise all required features and functions.

The following set of fundamental principles and concepts are applicable to coding and testing:

Coding Principles. The principles that guide the coding task are related to programming languages, and programming methods.

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

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

Programming principles: *As you begin writing code, be sure*

• Consider the use of pair programming.

• Select data structures that will meet the needs of the design.

• 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.

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

• Conduct a code walkthrough when appropriate.

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

• Refactor the code.

Testing Principles. 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.
Davis suggests a set of testing principles that have been adapted for use

Principle 1. ***All tests should be traceable to customer requirements.*** The objective of software testing is to uncover errors.

Principle 2. ***Tests should be planned long before testing begins.***

- Test planning can begin as soon as the requirements model is complete.
- 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.

Principle 4. ***Testing should begin "in the small"(Components-Unit Test) and progress toward testing "in the large."***

First test the modules and then test the entire system(software).

Principle 5. ***Exhaustive testing is not possible.***

it is impossible to execute every combination of paths during testing.

Deployment Principles

- The deployment activity encompasses three actions: delivery, support, and feedback.
- For agile process model each delivery cycle provides the customer and end users with an operational software increment that provides usable functions and features.
- **Principle 1.** ***Customer expectations for the software must be managed.***

the customer expects more than the team has promised to deliver, and disappointment occurs immediately.

Principle 2. ***A complete delivery package should be assembled and tested.*** A CD-ROM or other media (including Web-based downloads) containing all executable software, support data files, support documents, and other relevant information should be assembled and thoroughly beta-tested with actual users.

Principle 3. ***A support regime must be established before the software is delivered.*** An end user expects responsiveness and accurate information when a question or problem arises.

Principle 4. ***Appropriate instructional materials must be provided to end users.*** The software team delivers more than the software itself.

Appropriate training aids (if required) should be developed and troubleshooting guidelines should be provided.

Principle 5. *Buggy software should be fixed first, delivered later.* Under time pressure, some software organizations deliver low-quality increments with a warning to the customer that bugs "will be fixed in the next release."This is a mistake.

Chapter 3

Requirements Engineering:

- The tasks and techniques that lead to an understanding of requirements is called requirements engineering.
- Requirements engineering is a major software engineering action that begins during the communication activity and continues into the modeling activity.
- .Requirements engineering builds a bridge to design and construction.
- Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analyzing need, assessing feasibility, validating the specification and managing the requirements and transformed into an operational system.
- It encompasses seven distinct tasks: inception, elicitation, elaboration, negotiation, specification, validation, and management.
- It is important to note that some of these tasks occur in parallel and all are adapted to the needs of the project.

Inception:

- At inception, establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.

Elicitation:

- Ask the customer, the users, and others what the objectives for the system or product are, how the system or product is to be used on a day-to-day basis.
- A number of problems that are encountered as elicitation occurs.

• **Problems of scope.** The customers/users specify unnecessary technical details that may confuse, rather than clarify, overall system objectives.

• **Problems of understanding.** The customers/users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain.

• **Problems of volatility.** The requirements change over time. To help overcome these problems, requirements gathering in an organized manner.

Elaboration:

- The information obtained from the customer during inception and elicitation is expanded and refined during elaboration.
- Elaboration is driven by the creation and refinement of user scenarios that describe how the end user will interact with the system.
- Each user scenario is parsed to extract analysis classes.
- The attributes of each analysis class are defined, and the services that are required by each class are identified.
- The relationships and collaboration between classes are identified, and a variety of supplementary diagrams are produced.

Negotiation:

- It's relatively common for different customers or users to propose conflicting requirements.
- recouncile these conflicts through a process of negotiation.
- Customers,other stakeholders are asked to rank requirements and then discuss conflicts in priority.
- Using an iterative approach that prioritizes requirements, assesses their cost and risk, and addresses internal conflicts, requirements are eliminated, combined, and/or modified so that each party achieves some measure of satisfaction.

Specification:

- A specification can be a written document, a set of graphical models, a formal mathematical model, a collection of usage scenarios, a prototype, or any combination of these.
- A "standard template" should be developed and used for a specification.

Validation:

- The work products produced during of requirements engineering are assessed for quality during a validation step.
- Requirements validation examines the specification , all software requirements have been stated unambiguously, inconsistencies, and

errors have been detected and corrected and that the work products conform to the standards established for the process, the project, and the product.

- The primary requirements validation mechanism is the technical review

Requirements management.:

- Requirements for computer-based systems change, and the desire to change requirements persists throughout the life of the system.
- Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project proceeds.

Establishing Ground Work

The steps required to establish the groundwork for an understanding of software requirements.

Identifying Stakeholders

- Business operations managers, product managers,marketing people, internal and external customers, end users, consultants, product engineers, software engineers, support and maintenance engineers, and others.
- Each stakeholder has a different view of the system, achieves different benefits when the system is successfully developed, and is open to different risks if the development effort should fail.

Recognizing Multiple Viewpoints

- Because many different stakeholders exist, the requirements of the system will be explored from many different points of view.
- For example, the marketing group is interested in functions and features that will excite the potential market, making the new system easy to sell and End users may want features that are easy to learn and use.
- As information from multiple viewpoints is collected, requirements may be inconsistent or may conflict with one another.
- categorize all stakeholder information , that will allow decision makers to choose an internally consistent set of requirements for the system.

Working toward Collaboration

- The job of a requirements engineer is to identify areas of commonality (i.e., requirements on which all stakeholders agree) and areas of conflict or inconsistency

(i.e., requirements that are desired by one stakeholder but conflict with the needs of another stakeholder).

- stakeholders collaborate by providing their view of requirements, but a strong “project champion”(e.g., a business manager or a senior technologist) may make the final decision about which requirements make the cut.

Asking the First Questions

- Questions asked at the inception of the project should be “context free”.
- The first set of context-free questions focuses on the customer and other stakeholders, the overall project goals and benefits. For example, you might ask:
    - ✓ Who is behind the request for this work?
    - ✓ Who will use the solution?
- These questions help to identify all stakeholders who will have interest in the software to be built.
- The next set of questions enables you to gain a better understanding of the problem and allows the customer to voice his or her perceptions about a solution:

✓ What problem(s) will this solution address?

✓ Can you show me the business environment in which the solution will be used?

- The final set of questions focuses on the effectiveness of the communication activity itself. Gause and Weinberg call these “meta-questions” and propose the following list:

✓ Are you the right person to answer these questions? Are your answers “official”?

✓ Am I asking too many questions?

✓ Can anyone else provide additional information?

These questions (and others) will help to “break the ice” and initiate the communication that is essential to successful elicitation.

Elicting Requirements

- Requirements elicitation (also called *requirements gathering*) combines elements of problem solving, elaboration, negotiation, and specification. **Collaborative Requirements Gathering:**Many different approaches to collaborative requirements gathering have been proposed.
- Each makes use of a slightly different scenario, but all apply some variation on the following basic guidelines:

- ✓ Meetings are conducted and attended by both software engineers and other stakeholders.
- ✓ Rules for preparation and participation are established.
- ✓ An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
- ✓ A “facilitator” (can be a customer, a developer, or an outsider) controls the meeting.
- ✓ A “definition mechanism” (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements.

Quality Function Deployment

- *Quality function deployment* (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software.
- QFD emphasizes an understanding of what is valuable to the customer and then deploys these values throughout the engineering process.
- QFD identifies three types of requirements :

  Normal requirements. The objectives and goals that are stated for a product or system during meetings with the customer. If these requirements arepresent, the customer is satisfied. Examples of normal requirements might be requested types of graphical displays.

  Expected requirements. These requirements are implicit to the productor system and may be so fundamental that the customer does not explicitly state them. Their absence will be a cause for significant dissatisfaction. Examples of expected requirements are: ease of human/machine interaction, overall operational correctness and reliability, and ease of software installation.

  Exciting requirements. These features go beyond the customer’s expectations and prove to be very satisfying when present. For example, softwarefor a new mobile phone comes with standard features, but is coupled with a set of unexpected capabilities (e.g., multitouch screen, visual voice mail) that

delight every user of the product.

- QFD uses customer interviews and observation, surveys, and examination of historical data (e.g., problem reports) as raw data for the requirements gathering activity.

- These data are then translated into a table of requirements called the *customer voice table*.It is reviewed with the customer and other stakeholders.

Usage Scenarios

- As requirements are gathered, an overall vision of system functions and features begins to materialize.
- understand how these functions and features will be used by different classes of end users.
- To accomplish this, developers and users can create a set of scenarios that identify a thread of usage for the system to be constructed.
- The scenarios, often called *use cases* , provide a description of how the system will be used.

Elicitation Work Products

- The work products produced as a consequence of requirements elicitation will vary depending on the size of the system or product to be built.
- For most systems, the work products include

1. A statement of need and feasibility.
2. A bounded statement of scope for the system or product.
3. A list of customers, users, and other stakeholders who participated inrequirements elicitation.
4. A description of the system's technical environment.
5. A list of requirements (preferably organized by function) and the domain constraints that apply to each.
6. A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
7. Any prototypes developed to better define requirements.

- Each of these work products is reviewed by all people who have participated in requirements elicitation.

Developing Usecases:

- "A use case describes the system's behavior under various conditions as the system responds to a request from one of its stakeholders . "
- A use case tells a stylized story about how an end user interacts with the system under a specific set of situations.
- The first step in writing a use case is to define the set of "actors" that will be involved in the story.

- *Actors* are the different people represent the roles that people (or devices) play as the system operates.
- Every actor has one or more goals when using the system.
- Once actors have been identified, use cases can be developed.
- number of questions that should be answered by a use case:

- 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?

Basic *SafeHome* requirements, we define four actors: **homeowner** (a user), **setup manager** (likely the same person as **homeowner,** but playing a different role), **sensors** (devices attached to the system), and the **monitoring and response subsystem** (the central station that monitors the *SafeHome* home security function). For the purposes of this example, we consider only the **homeowner** actor. The **homeowner** actor interacts with the home security function in a number of different ways using either the alarm control panel or a PC:

.Considering the situation in which the homeowner uses the control panel, the basic use case for system activation follows:

1.The homeowner observes the *SafeHome* control panel to determine if the system is ready for input. If the system is not ready, a *not ready* message is displayed on the LCD display.

2. The homeowner uses the keypad to key in a four-digit password. The password is compared with the valid password stored in the system. If the password is incorrect and reset itself for additional input. If the password is correct, the control panel awaits further action.

3. The homeowner select the keys in *stay* or *away* to activate the system.

   ✓ *Stay* activates only perimeter sensors (inside motion detecting sensors are deactivated).

   ✓ *Away* activates all sensors.

4. When activation occurs, a red alarm light can be observed by the homeowner.

The basic use case presents a high-level story that describes the interaction between the actor and the system.

Building The Requirement Model

- The analysis model is to provide a description of the required information, functional, and behavioral domains for a computer-based system.

- The model changes dynamically as learn more about the system to be built, and other stakeholders understand what they really require.

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 to consider requirements from different viewpoints and to find inconsistencies and ambiguity of the gathered requirements.
- ➢ **Scenario-based elements:** The system is described from the user's point of view using a scenario-based approach. For example, basic use case.
- ➢ **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 behaviors.
- ➢ **Behavioral elements:** The behavior of a computer-based system can have a effect on the design that is chosen and the implementation approach that is applied. Therefore, the requirements model must provide modeling elements that represent behavior.

Analysis Patterns Anyone who has done requirements engineering on morethan a few software projects then identify the some problems reoccur across all projects within a 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.
- Advantages of Analysis patterns are:
  - ✓ Analysis patterns speed up the development of abstract analysis models that capture the main requirements of the concrete problem
  - ✓ Analysis patterns facilitate the transformation of the analysis model into a design model by suggesting design patterns and reliable solutions for common problems.
- 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.

Negotiating Requirements:

- The negotiation is to develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team.
- The best negotiations strive for a “win-win” result.
- That is, stakeholders win by getting the system or product that satisfies the majority of their needs and you (as a member of the software team) win by working to realistic and achievable budgets and deadlines.
- Boehm defines a set of negotiation activities at the beginning of each software process iteration.
- The following activities are defined:

1. Identification of the system or subsystem’s key stakeholders.
2. Determination of the stakeholders’ “win conditions.”
3. Negotiation of the stakeholders’ win conditions to reconcile them into a set of win-win conditions for all concerned (including the software team).
Successful completion of these initial steps achieves a win-win result.

Validating Requirements:

- As each element of the requirements model is created, it is examined for inconsistency,omissions, and ambiguity.
- The requirements represented by the model are prioritized by the stakeholders and grouped within requirements packages that will be implemented as software increments.
- A review of the requirements model addresses the following questions:

• Is each requirement consistent with the overall objectives for the system/product?
• Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
• Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
• Is each requirement bounded and unambiguous?
• Does each requirement have attribution?
• Do any requirements conflict with other requirements?
• Is each requirement achievable in the technical environment ?
• Is each requirement testable, once implemented?
• Does the requirements model properly reflect the information, function, and

behavior of the system to be built?

- These and other questions should be asked and answered to ensure that the requirements model is an accurate reflection of stakeholder needs and that it provides a solid foundation for design.