# UNIT I

**Introduction to Software Engineering: A Generic view of Process:** Software Engineering, Process Framework, CMM Process Patterns, Process Assessment.

**Process Models:** Prescriptive Models, Waterfall Model, Incremental Process Models, Evolutionary Process Models, Specialized Process Models, The Unified Models, Personal and Team Process Models, Process Technology, Product and Process.

**An Agile view of Process:** Introduction to Agility and Agile Process, Agile Process Models.

---

**Software**: **"**It is defined as an organized set of instructions to deliver a desired output by considering various processes and functions.**"**

Now-a-days software acts both as product (Application Software) and vehicle for delivering a product (System Software).

- As product, software delivers computing potential embodied by computer hardware (or) by a network of computers that are accessible by local hardware.
- As the vehicle used to deliver product, software acts as basis for control of PC (OS), Communication of information (Networks) and Creation and Control of other programs (Software Tools, Environments).

***Software Development***: "It is a creative activity, where a software system is developed from initial concept through a working system".

***Software Maintenance:*** "Process of changing a developed system once it is delivered".

***Software Evolution:*** "Evolutionary process where software is conceptually changed over its lifetime in response to the changing requirements".

**Software Characteristics:**

1. Software is developed (or) engineered, it is not manufactured.
2. Software doesn't "wear out". Sometimes, it may deteriorate with too many changes.
3. Most of softwares continue to be custom build, component-based assembly. [As software is both product and vehicle to carry product].

**Types of Software Applications:** Various types of softwares include:

**1)** **System Software: "**A collection of programs written to service other programs.**"**
**2)** **Application Software: "**It consists of stand-alone programs that solve a specific business need.**"**
3) **Engineering/Scientific Software: "**Characterized by numerical algorithm.**"** Ex: Astronomy etc.
4) **Embedded Software: "**Resides within a product/system to implement and control features and functions for end user.**"** (Ex: keypad control for oven.)
**5)** **Web-Applications: "**Set of linked hyper text files which provide e-commerce and B2B (Business to Business) application.**"**
6) **AI Software: "**Makes use of non-numerical algorithms to solve complex problems.**"** Ex: Robotics.
7) **Product-line Software: "**Capable to be used by many different customers.**"** (Ex: DBMS in BSE.)
**8)** **Open Source: "**Available for all customers and can be modified.**"**
**9)** **Net Sourcing: "**WWW is used as a content provider. So Software Engineers can develop simple and sophisticated applications.**"**
**10)** **Ubiquitous Computing: "**Allows small devices to communicate across vast networks.**"**

## A GENERIC VIEW OF PROCESS

**Software Engineering-A Layered Technology:** According to IEEE, "Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software i.e., application of engineering to software.

Software engineering is a layered technology. The foundation for software engineering is the ***process*** layer. The software engineering process is the glue that holds the technology layers together and enables timely development of software. Process defines a framework that must be established for effective delivery of software engineering technology. The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, ***quality is ensured***, and change is properly managed.

Software engineering ***methods*** provide the technical how-to's for building software. Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.

Software engineering ***tools*** provide automated or semi automated support for process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called ***Computer Aided Software Engineering (CASE)***, is established.

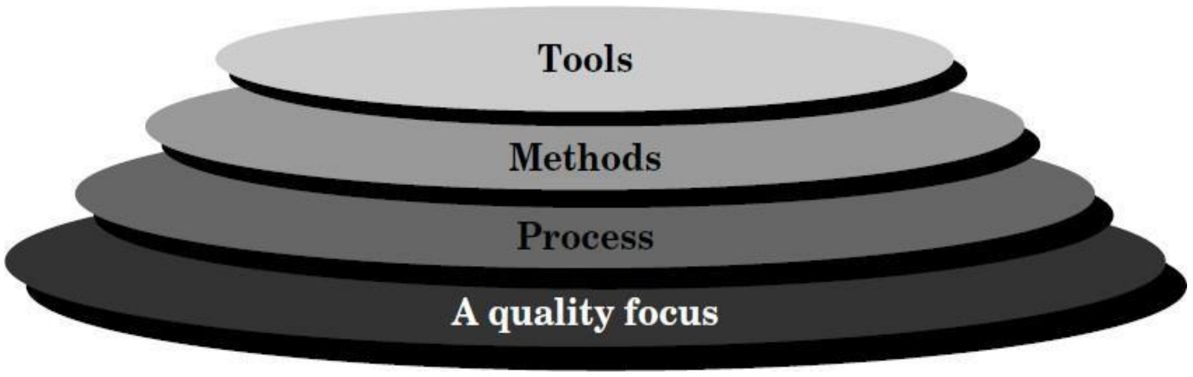


**Fig:** Software Engineering Layers

**A Process Framework: "**It establishes a foundation for complete software process by identifying a small number of framework activities and umbrella activities that are applicable to all software projects.**"**

- ✓ Each framework activity is populated by a set of Software Engineering actions (Ex: Design) – collection of related tasks.
- ✓ Each action is populated with individual *work tasks*.
- ✓ Software is *determinate* if order and timing of inputs, processing and outputs is predictable, otherwise it is referred an *indeterminate*.

**Framework Activities:**

1. **Communication:** It involves heavy communication and collaboration with customer and other stakeholders. It encompasses requirements gathering and related activities.
2. **Planning:** It plans for Software Engineering work that follows. It describes technical tasks to be conducted, resources that will be required, likely risks, work products to be produced and a work schedule.
3. **Modeling:** This activity focuses on creation of models that allow stakeholders (customer, developer) to better under software requirements and design that will achieve requirements.

4. **Construction or Development:** It combines code generation and testing to uncover errors in the code. (Manual, automated actions).
5. **Deployment:** The software (completed/partial increment) is delivered to the customer for evaluation and feedback of it.

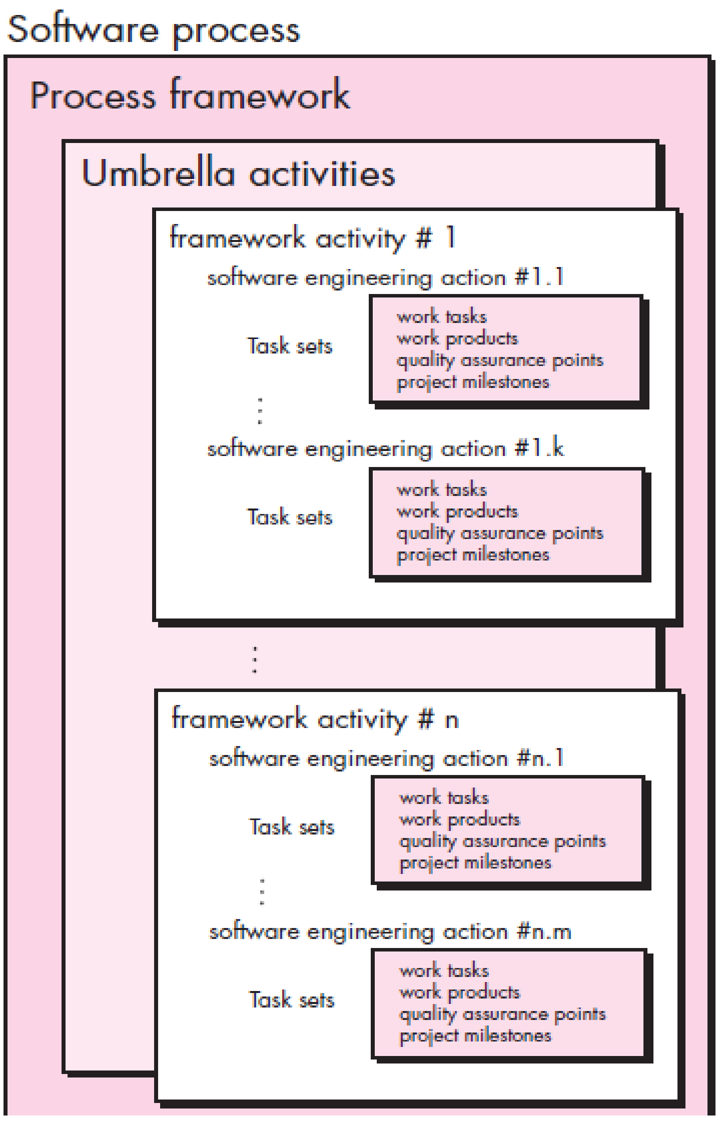


**Fig:** Process Framework

**Umbrella Activities:**
1. ***Software Project Tracking and Control*** – Assess progress against the project plan and maintain schedule.
2. ***Risk Management*** – Assesses risk that effect project outcome (or) Quality of product.
3. ***Software Quality Assurance*** – Activities to ensure software quality.
4. ***Formal Technical Reviews*** **-** Assess work products to uncover and remove errors before next action.
5. ***Software Configuration Management*** – Manages effects of changes throughout the software process.
6. ***Measurement*** – Defines and collects process, project, product measures to meet customer requirements.
7. ***Reusability Management*** – Defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
8. ***Work Product Preparation and Production*** – Focuses on activities required to create work product such as models, documents, logs, forms and lists.

**Product and Process:** If process is weak, the end products will undoubtedly suffer. But an obsessive over- reliance on process is also dangerous.

**Process Technology:** "Process Technology tools are used to help software organizations to analyze their current process, organize work tasks, controls and monitor progress, manage technical quality".

**Process Assessment:** It is required to ensure that the process meets set of basic principles that are needed for a successful software engineering.

**Approaches:**

- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI):** It provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment
- **SPICE: S**oftware **P**rocess **I**mprovement and **C**apability d**E**termination.
- **SCAMPI: S**tandard **C**MMI **A**ssessment **M**ethod for **P**rocess **I**mprovement. It provides a five step process assessment model that includes: Initiating, Diagnosing, Establishing, Acting, and Learning.
- **ISO 9001:2000 for software:** Generic standard to improve the overall quality of products, system and services of software organization. It adopted a plan-do-check-act cycle for continuous process improvement

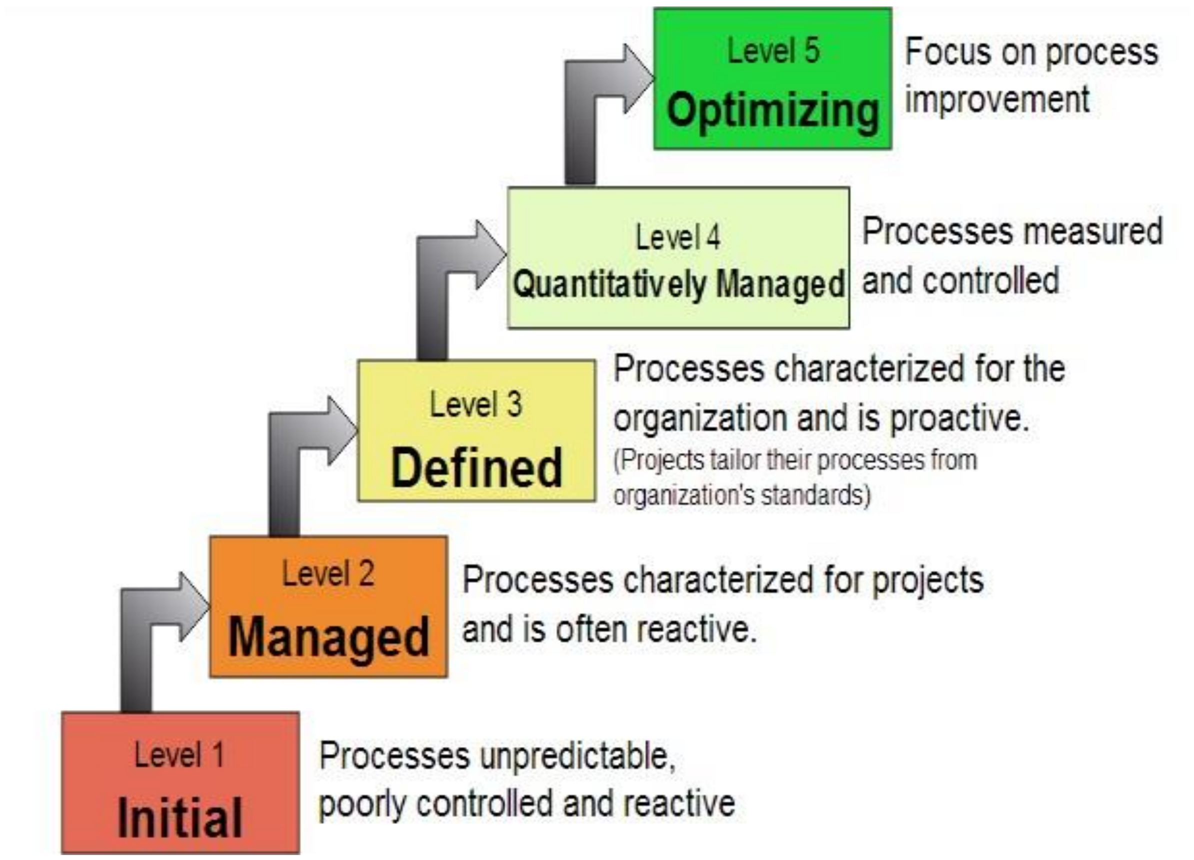
## CMMI

Capability Maturity Model Integration is a maturity model used to rank software development organizations. It is proposed by Software Engineering Institute (SEI). It represents process model as 1.Continuous Model 2.Staged Model

**CMM: "**It is maturity framework strategy that focuses on continuously improving the development and management of organization workforce.**"**

**Capability Levels:** The Capability levels depend on Key Process Areas (KPA). These are given as:

1. **Level 0 (Incomplete):** The process area (Ex; Requirement Management) is not performed or does not achieve all goals and objectives defined by CMMI for level 1 capability.
2. **Level 1 (Initial/Performed):** All specific goals of process area have been satisfied. Work tasks required to produce defined work products are being conducted.
3. **Level 2 (Repeatable/Managed):** All level 1 criteria have been satisfied.
   - ✓ All work related to PA is up-to-date with organization expectations.
   - ✓ All people doing work have access to adequate resources to get the job done i.e., stakeholders are actively involved.
   - ✓ Work task and work products are "monitored, controlled, reviewed and evaluated".
4. **Level 3 (Defined):** All level 2 criteria are achieved. "The process is tailored from organization's set of standard other processes and contributes work products, measures and other process improvement information to organizational process assets"
5. **Level 4 (Quantitatively managed):** All level 3 criteria are satisfied. Quantitative objectives for quality and process performance are established and used as criteria in managing the process.
6. **Level 5 (Optimized):** All level 4 criteria are satisfied. Process Area (PA) is adapted and optimized using quantitative needs to meet changing customer needs and continually improve efficiency of Process Area.


Level 5
Optimizing
Focus on process improvement
Level 4
Quantitatively Managed
Processes measured and controlled
Level 3
Defined
Processes characterized for the organization and is proactive.
(Projects tailor their processes from organization's standards)
Level 2
Managed
Processes characterized for projects and is often reactive.
Level 1
Initial
Processes unpredictable, poorly controlled and reactive

SEI has associated key process areas (KPAs) with each of the maturity levels. KPAs describe those software engineering functions (e.g., software project planning, requirements management) that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics:

- *Goals*—the overall objectives that the KPA must achieve.
- *Commitments*—requirements (imposed on the organization) that must be met to achieve the goals or provide proof of intent to comply with the goals.
- *Abilities*—those things that must be in place (organizationally and technically) to enable the organization to meet the commitments.
- *Activities*—the specific tasks required to achieve the KPA function.
- *Methods for monitoring implementation*—the manner in which the activities are monitored as they are put into place.
- *Methods for verifying implementation*—the manner in which proper practice for the KPA can be verified.

Eighteen KPAs are defined across the maturity model and mapped into different levels of process maturity. The following KPAs should be achieved at each process maturity level:

**Process maturity level 2**
- ✓ Software configuration management
- ✓ Software quality assurance
- ✓ Software subcontract management
- ✓ Software project tracking and oversight
- ✓ Software project planning
- ✓ Requirements management

**Process maturity level 3**
- ✓ Peer reviews
- ✓ Intergroup coordination
- ✓ Software product engineering

- ✓ Integrated software management
- ✓ Training program
- ✓ Organization process definition
- ✓ Organization process focus

**Process maturity level 4**

- ✓ Software quality management
- ✓ Quantitative process management

**Process maturity level 5**

- ✓ Process change management
- ✓ Technology change management
- ✓ Defect prevention

Each of the KPAs is defined by a set of *key practices* that contribute to satisfying its goals. The key practices are policies, procedures, and activities that must occur before a key process area has been fully instituted. The SEI defines *key indicators* as "those key practices or components of key practices that offer the greatest insight into whether the goals of a key process area have been achieved".

CMMI defines each process area in terms of:

- ➢ **Specific Goals:** Essential characteristics which must exist in all the activities implied by a given Process Area.
- ➢ **Specific Practices:** Set of tasks to be accomplished to achieve specific goals. Ex: SG and SP for "Project Planning"

  SG1: Establish Estimates

  SP 1.1 – Establish Project Scope

  SP 1.2 – Establish Work Product Estimation SP 1.3 – Define Project Life Cycle

  SP 1.4 – Determine estimates of effort cost.

  SG2: Develop a Project Plan

  SG3: Obtain commitment to the plan
- ➢ **Generic Goals:** These are used to achieve a particular capability.
- ➢ **Generic Practices:** Practices that correspond to the level goal must be achieved.

**PROCESS PATTERNS:** Software process can be a collection of patterns that define a set of activities, work tasks, work products and relational behaviors. *Template* to describe a process pattern contains:

- ✓ **Pattern Name:** It should describe function within software process. (Ex: Customer-Communication).
- ✓ **Intent:** Purpose of pattern. Can be explained with diagrams.
- ✓ **Type:** Three types of patterns are there:
  1. **Task Patterns:** Software Engineering action or work task that is part of process and relevant to successful Software Engineering practice defined.
  2. **Stage Patterns:** Defines "a framework activity with multiple work tasks in it", for the process. Ex: Communication contains Requirement Gathering
  3. **Phase Pattern:** Defines "sequence of framework activities that occur with the purpose", may be iterative. Ex: Spiral Model
- ✓ **Initial Context:** Condition under which pattern applies are described.
- ✓ **Problem:** Problem to be solved by pattern is described.
- ✓ **Resulting Context:** Conditions that result after implementation.
- ✓ **Related Patterns:** List of process patterns that are related.
- ✓ **Known Uses/Examples:** Instances where patterns are applicable.

**PERSONAL AND TEAM PROCESS MODELS:** These models were proposed by Watts Humpry. Each software engineer can create a process that best fits his or her needs. A team can also create a process that meets narrower needs of individuals and broader needs of organization.

**Personal Software Process (PSP):** Every developer uses some process o build the computer software. The process may be Adhoc, may change daily and may not be efficient. PSP process model defines ***5 framework activities***:

1. Planning
2. High-Level Design
3. High-Level Design Review
4. Development
5. Post-mortem.

**Team Software Process (TSP):** An ideal software team can be of 3-20 Software Engineers (Integrated Product Teams). Its goal is to build a "self-directed" project team to produce high-quality software.

**Objectives:**

1. Build self-directed teams that plan and track their work, establish goals and own their processes and plans.
2. Show managers how to motivate their teams and sustain peak performance.
3. Accelerate software project improvement to achieve CMM level-5 targets.
4. Guide high-maturity organization to improve process standards.

***Framework Activities*:**

1. Launch (Communication and Planning).
2. High-level Design.
3. Implementation.
4. Integration and Test.
5. Post-mortem.

Note: TSP uses scripts (sequence of Tasks), forms, and standards to guide team members.

## PROCESS MODELS

## PRESCRIPTIVE MODELS

Prescribes set of process elements such as framework activities, Software Engineering actions, tasks, work products, assurance and charge control mechanism. Each process model prescribes a work flow. Various Prescriptive models include:

1) **THE WATERFALL MODEL:** This model is proposed by Winston Royce. It is also called as ***classic life cycle***. It is the oldest paradigm (model) in Software Engineering.

It suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, then with Modeling, construction and deployment.

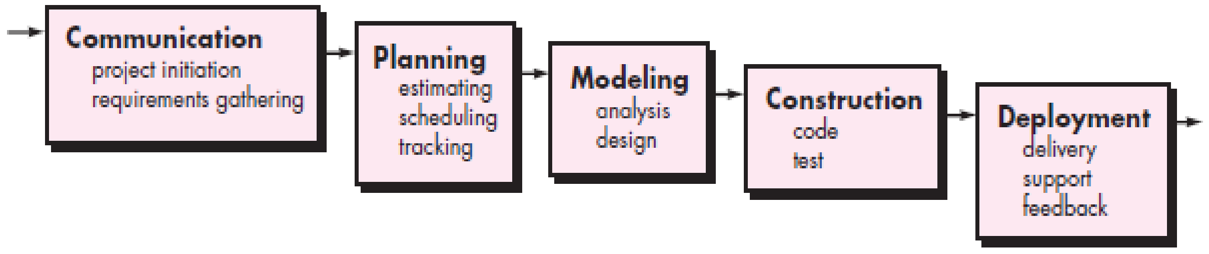


**Fig**: Waterfall Model.

The framework activities of the Waterfall model include:

1. **Communication:** It involves heavy communication and collaboration with customer and other stakeholders. It encompasses requirements gathering and related activities.
2. **Planning:** It plans for Software Engineering work that follows. It describes technical tasks to be conducted, resources that will be required, likely risks, work products to be produced and a work schedule.
3. **Modeling:** This activity focuses on creation of models that allow stakeholders (customer, developer) to better under software requirements and design that will achieve requirements.
4. **Construction:** It combines code generation and testing to uncover errors in the code. (Manual, automated actions).
5. **Development:** The software (completed/partial increment) is delivered to the customer for evaluation and feedback of it.

**Problems:**

- Real projects rarely follow the sequential flow.
- It’s often difficult for customer to state all requirements explicitly.
- Working version of program(s) will not be available until late in project time-span. So, customer must have patience.

A variation in the representation of the waterfall model is called the *V-model*. The V-model depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities. As software team moves down the left side of the V, basic problem requirements are refined into progressively more detailed and technical representations of the problem and its solution. Once code has been generated, the team moves up the right side of the V, essentially performing a series of tests (quality assurance actions) that validate each of the models created as the team moved down the left side. In reality, there is no fundamental difference between the classic life cycle and the V-model. The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work.

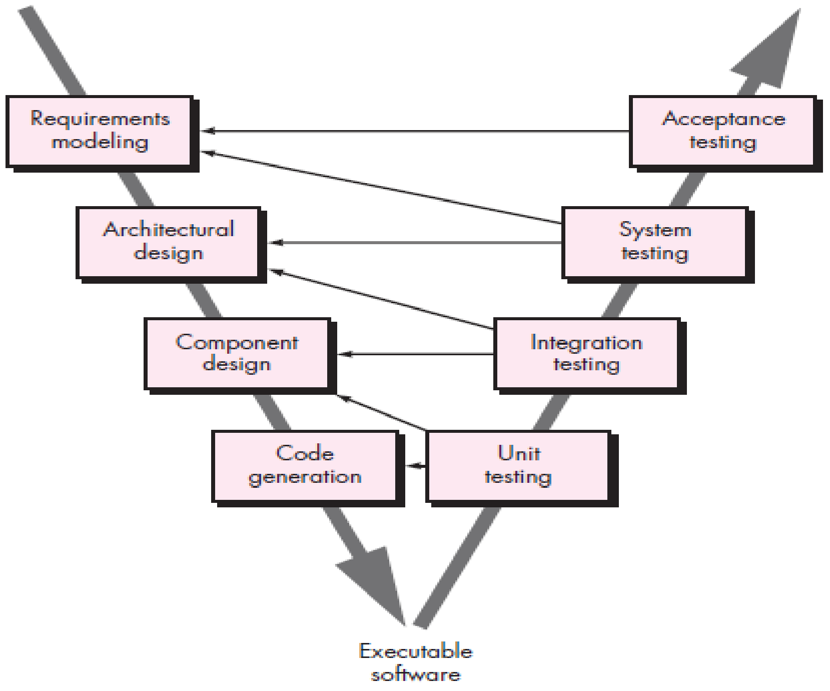


**Fig:** The V – Model

2) **INCREMENTAL PROCESS MODEL:** "It divides the software development process into certain number of increments with each increment comprising 5 phases of waterfall model". Each linear sequence produces "deliverable increments" of software. Ex: WORD software (Entering Data, Editing, Spell check etc.)

First increment is often a core product i.e., basic requirements are addressed, supplementary features are not delivered. The core product is used and evaluated by the customer, based on that plans for next increment development. This process is repeated till complete product is produced.

The framework activities of the Incremental Process model include:

1. **Communication:** It involves heavy communication and collaboration with customer and other stakeholders. It encompasses requirements gathering and related activities.
2. **Planning:** It plans for Software Engineering work that follows. It describes technical tasks to be conducted, resources that will be required, likely risks, work products to be produced and a work schedule.
3. **Modeling:** This activity focuses on creation of models that allow stakeholders (customer, developer) to better under software requirements and design that will achieve requirements.
4. **Construction:** It combines code generation and testing to uncover errors in the code.
5. **Development:** The software (completed/partial increment) is delivered to the customer for evaluation and feedback of it.

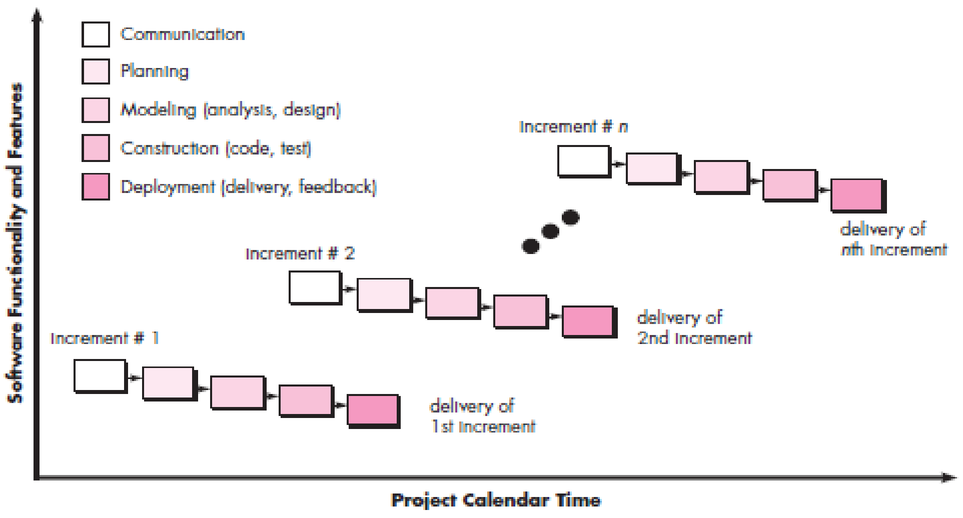

Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
Increment # n
delivery of nth increment
Increment # 2
delivery of 2nd increment
Increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

**Figure:** Incremental Process model.

**Advantages:**

- ✓ Technical risks reduced, with each increment.
- ✓ When team size is small, it is the correct choice.
- ✓ Customer can expect a core product in short time-span.

3) **RAD MODEL:** RAD stands for Rapid Application Development. It is an incremental software process model that emphasizes short development cycle. It is a version of waterfall model with rapid development. If requirements are well understood and project scope is constrained RAD process enables development team to create a "fully functional system" within a very short time period (60- 90 days). Each major function can be addressed by a separate RAD team and then integrated to form a whole project.

The framework activities of RAD model include:

1. **Communication:** It involves heavy communication and collaboration with customer and other stakeholders. It encompasses requirements gathering and related activities.
2. **Planning:** It plans for Software Engineering work that follows. It describes technical tasks to be conducted, resources that will be required, likely risks, work products to be produced and a work schedule.

3. **Modeling:** This activity focuses on creation of models that allow stakeholders (customer, developer) to better under software requirements and design that will achieve requirements.
4. **Construction:** It combines code generation and testing to uncover errors in the code.
5. **Development:** The software (completed/partial increment) is delivered to the customer for evaluation and feedback of it.

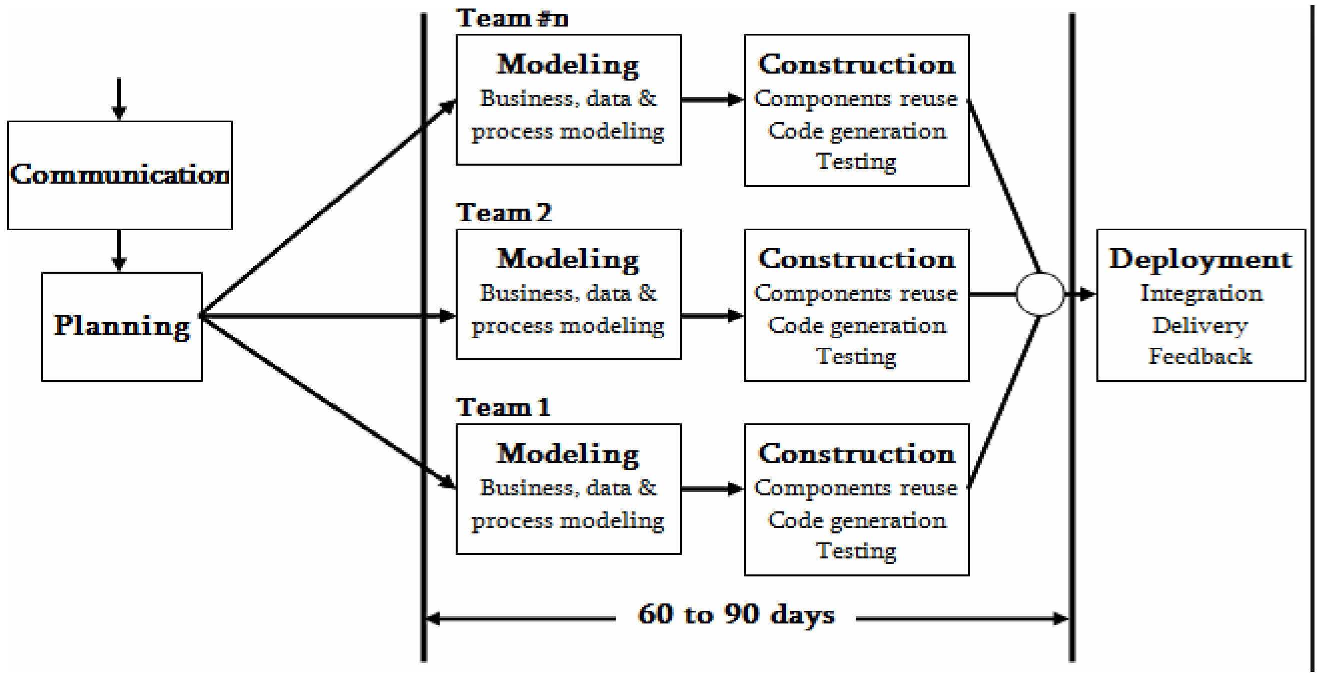


**Fig:** RAD Process model.

**Advantages:**

- Project with 2-3 months deadline opt for RAD.
- Task is divided among teams to fasten development process.

**Disadvantages:**

- Large projects using RAD may not work.
- If high performance is an issue, RAD may not work.
- RAD may not be appropriate when technical risks are high.

## EVOLUTIONARY PROCESS MODELS

These models are specially designed to accommodate a product that evolves over time. These are iterative and enable software engineers to develop more complete software versions.

1) **PROTOTYPING:** Prototyping model is used when customer defines a set of objectives, but does not identify detailed input, processing output requirements, developer is unsure of efficiency of algorithm, adaptability of operating system etc, where phased model is inappropriate.

Prototyping model can be used as a standalone process model. Prototyping paradigm assists the software engineer and customer to better understand what is to be built when requirements are fuzzy. Prototype helps to identify software requirements.

Prototyping paradigm begins with communication, then quickly planning the prototyping iteration, modeling quick design, construction of prototype, and the prototype is deployed and then evaluated by the customer/user. Feedback is used to refine requirements for the software.

Prototype can serve as “the first system”, where users get a feel of actual system and developers get to build something immediately.

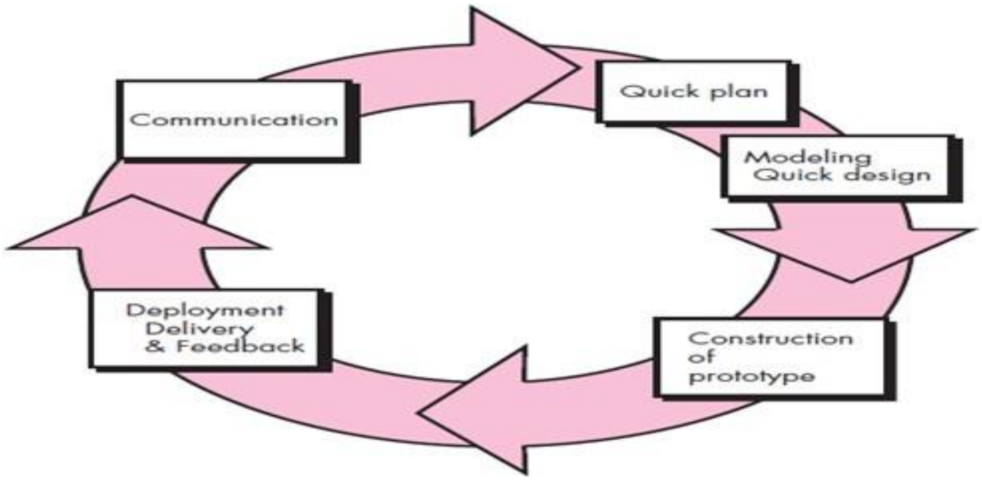


**Fig:** Prototyping Model

*Prototyping can be problematic for following reasons:*

- Developers may not consider overall software quality. (Few “fixes” are applied to satisfy customer).
- The developer often compromises in the implementation to get a prototype working quickly. The key is to define the rules of the game at the beginning; i.e., customer and developer must both agree that prototype is built to serve as a mechanism for defining requirements.

**2) THE SPIRAL MODEL:** The Spiral model is proposed by “Boehm”. This model was developed to encompass the best features of waterfall model and prototyping. It is a ***risk-driven*** process model with the risk analysis feature.

**Features:**

1) <u>Cyclic Approach</u> for increasing system’s degree of definition and implementation while decreasing degree of risk-Risk is considered as each revolution is made.
2) <u>Anchor Point Milestone</u> for ensuring stakeholders commitment to feasible and mutually satisfactory systems solution. (Milestone is a combination of work products and conditions).

Spiral model may be viewed as a Meta model, as it can accommodate any process development model. Software is developed as a series of evolutionary releases. Project manager adjusts planned number of iterations to complete the software. During early iterations prototype is generated and during later iterations complete version is developed.

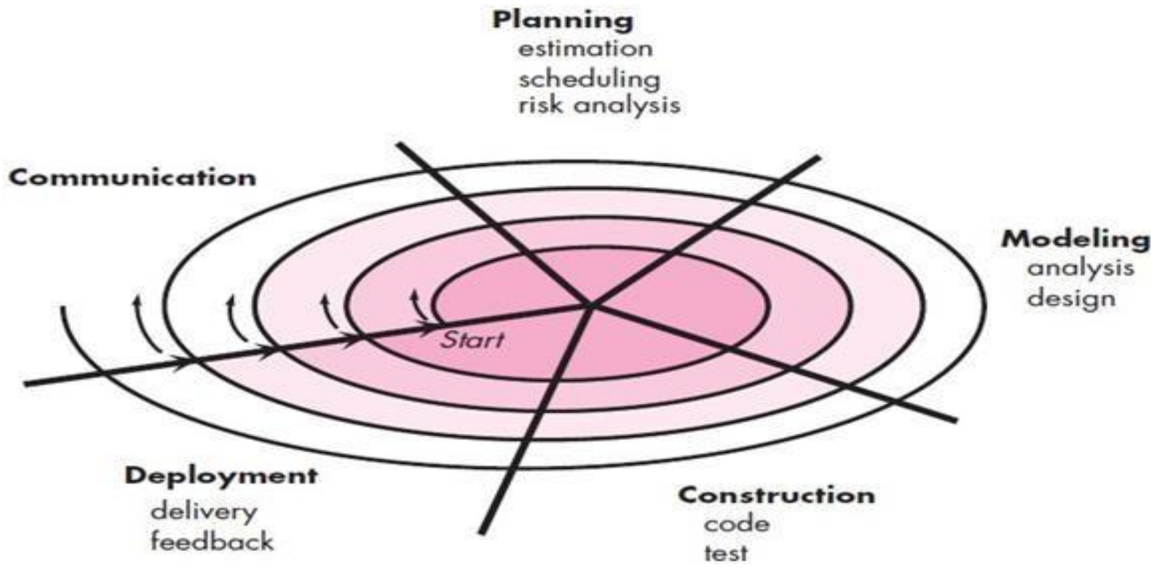


**Fig:** Spiral Model

In Spiral model, the first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral model might be used to develop a prototype and then progressively more sophisticated versions of the software. Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the software.

- ✓ The first circuit around the spiral might represent a "Concept Development Project" that starts at the core of the project and continues till concept development isover.
- ✓ Then with the next spiral "New Product Development Project" commences. New product will evolve through a number of iterations around spiral.
- ✓ Next circuit around the spiral might be used to represent a "Product Enhancement Project". The spiral, when characterized in this way, remains operative until software is retired.

**Advantages:**

- ▪ It is a realistic approach to the development of large scale systems and software. (Software evolves as the process progresses).
- ▪ It uses and enables the developer to apply the prototyping approach to any stage in evolution of product.
- ▪ Considers technical risks at all the stages of the project, and reduces risks before they become problematic.
- ❖ Like other paradigms, spiral model is not a panacea (Medicine).It demands considerable risk assessment expertise for success. If a major risk is not covered and managed, problems will occur.

3) **CONCURRENT DEVELOPMENT MODEL:** It is also called as "Concurrent Engineering". This model is represented schematically as a series of framework activities, Software Engineering actions and tasks, and their associated states concurrently. It strives to make all software development activities to be concurrently implemented.

   - Ex: "Modeling" activity for spiral model is accomplished by invoking prototyping and/or analysis Modeling and specification and design.

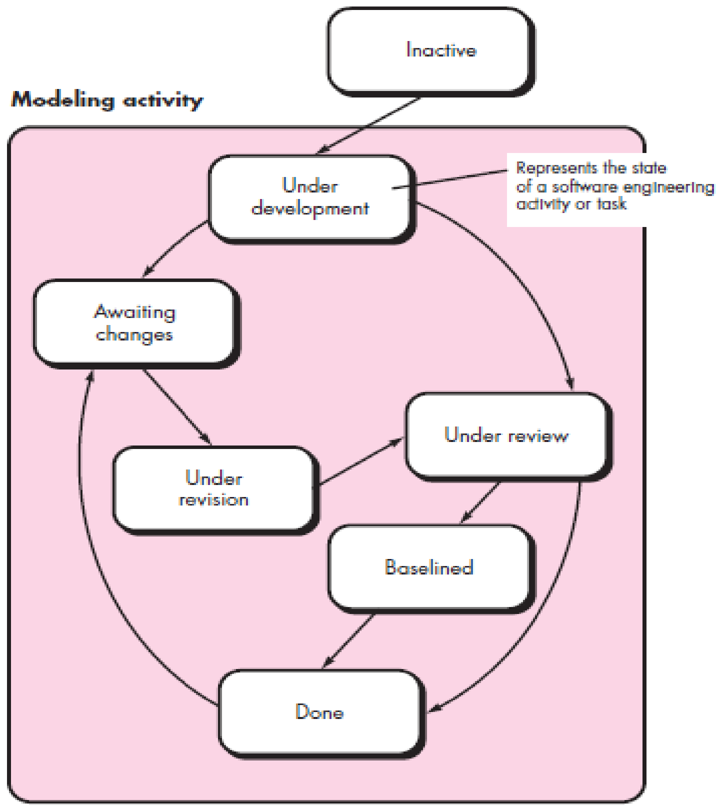


**Fig:** One element of concurrent model development model.

All activities (communication/modeling/construction etc) exist concurrently but reside in different states. ***State*** is an externally observable mode of behavior). For example, early in a project, the communication activity has completed its first iteration and exists in the awaiting changes state. Modeling activity which was in none state will now move to under development state.

- ❖ This model defines a series of events which will trigger transition from state to state for each of Software Engineering activities, actions/tasks.

**Advantages:**

- ✓ Applicable to all types of software development, provides accurate picture of current state of a project.
- ✓ The software engineering activities, tasks and actions are defined as a network of activities, rather than sequence of events.

**Evolutionary Models Drawbacks:**

- Prototyping poses a problem to project planning because of uncertain number of cycles required to construct product.
- Do not establish maximum speed of evolution.
- May not give flexibility and extensibility for the software process.

## SPECIALIZED PROCESS MODELS

These models are used when a narrowly defined Software Engineering approach is chosen.

**1. COMPONENT-BASED DEVELOPMENT:** Commercial Off-The-Shelf (COTS) software components are used when software is to be build. These components provide targeted functionality so that component to be integrated into the software.

- ➢ It incorporates many characters of the spiral model. It is evolutionary in nature, composes applications from pre-packaged (COTS) software components.

**Steps:**

1. Available component-based products are researched and evaluated for the application domain in question.
2. Component integration issues are considered.
3. Software architecture is designed to accommodate the components.
4. Components are integrated into the architecture.
5. Comprehensive testing is used (conducted to ensure proper functionality).

**Advantage:** Software reuse-Important to produce high quality software.

**2. FORMAL METHODS MODEL:** Specialized software development approach that uses mathematical based techniques for specifying, developing and verifying the computer softwares. Formal Methods Model helps the software developers to apply correct mathematical notations to create the issue of insufficiency, inconsistency and uncertainty of the software by applying mathematical analysis.

During design phase, formal methods model acts as a program verifier and help Software Engineers to detect and correct these errors, which are otherwise very difficult to be detected. This model assures defect free software.

**Drawbacks:**

1. Time consuming and expensive.
2. Software Engineers need extensive training to apply this model.
3. Clients needed to be technically sound for proper communication.

- ❖ Because of these reasons Formal Methods Models are used only in development of high integrity software applications where safety and security is of atmost importance.

3. **ASPECT-ORIENTED SOFTWARE DEVELOPMENT (AOSD):** As modern computer based systems become more sophisticated and complex, some concerns (security, fault tolerance, memory management etc) span the entire architecture. When concern cut across multiple system functions, features and information, they are referred as crosscutting concerns. Aspectual Requirements define those crosscutting concerns that have impact across the software architecture.

AOSD, often referred to as Aspect-Oriented Programming (AOP), is a relatively new software engineering paradigm which provides a process for defining, specifying, designing and constructing aspects (crosscutting concerns).

Presently there is no distinct Aspect-Oriented Process. If such an approach is developed, then it must integrate the characteristics of both spiral and concurrent model, because of their evolutionary and parallel natures respectively.

## THE UNIFIED PROCESS MODEL: *This model is also referred as RUP (Rational Unified Process).*

Unified Process refers to a methodology of extracting the most essential activities of conventional software development phases (communication, planning, Modeling, construction and deployment) and characterizing them, so that they can be applied in the Agile (highly valued) software development.

**History:** Jacobson, Rumbaugh and Greedy Booch developed the Unified Process, a framework for Object-Oriented Software Engineering using UML. Today, Unified Process and UML are used on Object- Oriented projects of all kinds.

- The iterative, incremental model proposed by the Unified Process can and should be adapted to meet specific project needs.

**Phases of the Unified Process:**

1. **Inception Phase:** The Inception Phase encompasses both customer communication and planning activities. By collaborating with customer and end-users, business requirements are identified and described through a set of use cases [sequence of actions that are performed by an actor (Ex: A person, machine, and another system). As actor interacts with the software, use cases provide project scope.

**The Inception Phase must:**

i. Produce a business case.
ii. Identify business requirements, business and process risks.
iii. Give overall vision for the project, as the outputs result in various documents/work products.

**Work Products:**

| | |
|---|---|
| 1) Vision Documents | 5) Initial Risk Assessment |
| 2) Initial use-case model | 6) Project Plan [Phases and Interaction] |
| 3) Initial Project Glossary | 7) Business Model (if necessary) |
| 4) Initial Business case | 8) One or more Prototypes |

2. **Elaboration Phase:** The Elaboration Phase encompasses planning and Modeling activities. This phase refines and expands preliminary use-cases that were developed in inception phase. The Elaboration Phase expands the architectural representation to five views: 1) Use case Model, 2) Analysis Model, 3) Design Model, 4) Implementation Model, 5) Deployment Model.

Elaboration Phase creates an "executable architectural baseline" that represents "first cut" executable system-prototype. Architectural baseline provides viability of the project but not all features and functions required to use the system. Modifications to the plan may be made at this time.

**Work Products:**

1) Use case Model
2) Supplementary Requirements
3) Analysis Model
4) Software Architecture Prototype
5) Executable Architecture Description
6) Preliminary Design Model
7) Revised Risk List
8) Project plan, includes
   a) Iteration Plan
   b) Adapted workflow
   c) Milestones
   d) Technical work products
9) Preliminary User Manual

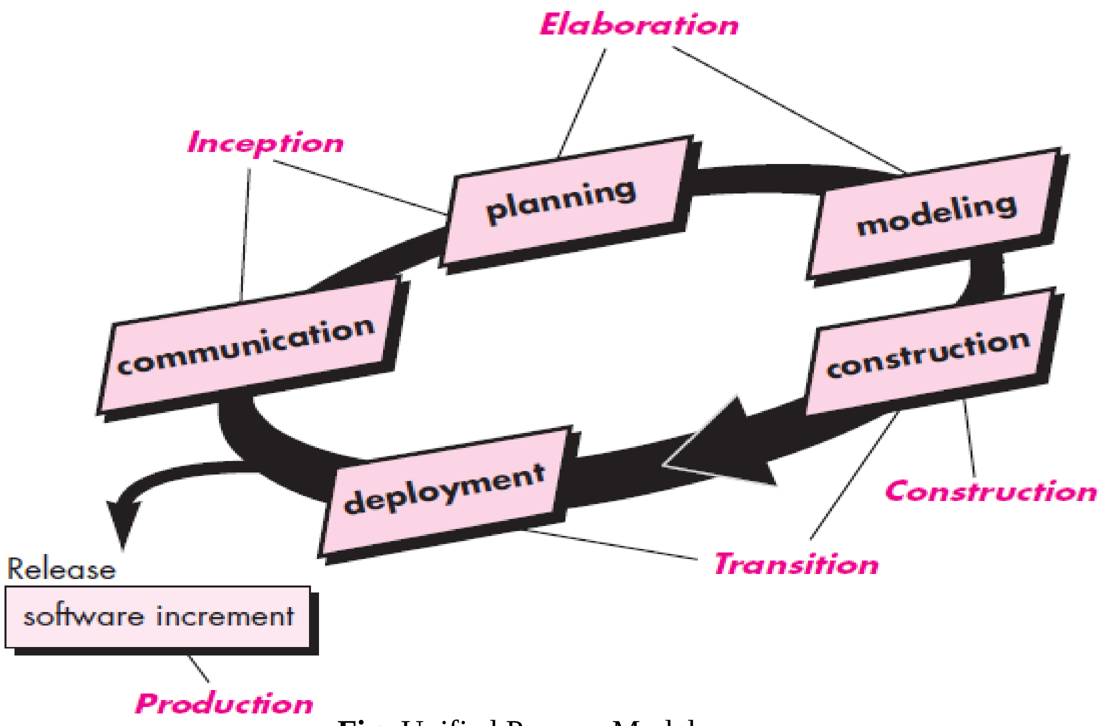


**Fig:** Unified Process Model.

3. **Construction Phase:** The Construction Phase is same as construction activity, where the application is coded and tested. The Construction Phase develops suitable code for each component of the software. To do this, analysis and design models started in the elaboration phase are completed to reflect the final version of software increment.
   - All necessary and required features and functions of software increment (release) are implemented in source code.
   - Unit tests are designed and executed for each software increment.
   - Integration activities (Component Assembly and Integration Testing) are conducted.

**Work Products:**

1) Design Model.
2) Software Components
3) Integrated Software Increment
4) Test Plan and Procedure.
5) Test Cases
6) Support Document
   a. User Manuals
   b. Installation Manuals
   c. Description of Current Increment

4. **Transition Phase:** The Transition Phase encompasses latter stages of construction and first part of deployment activities. Software is given to end-users for beta testing and user feedback about both defects and necessary changes. Software team creates necessary support information (Ex: user manuals, installation procedures) required for release.

   **Work Products:**
   1) Delivered software increment
   2) General User Feedback
   3) Beta Test Report

5. **Production Phase:** In the Production Phase, on-going use of software is monitored, support for operating environment is provided and defect reports and request for changes are submitted and evaluated.

❖ Construction, Transition and Production phases are being conducted concurrently sometimes. So, five Unified Process phases do not occur in a sequence.

## AN AGILE VIEW OF PROCESS

**Q: What is Agility?**

**Ans:** Agility is dynamic, content specific, aggressively change and growth oriented. Agile software is highly valued software. Agile team is a nimble team able to respond to changes appropriately.

***The Agile Alliance defines 12 principles to achieve agility:***

1. Our highest priority is to satisfy customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even later in development. Agile processes harness change for customer’s competitive advantage.
3. Deliver working software frequently, from couple of weeks to months.
4. Business people and developers must daily work together throughout the project.
5. Build projects around motivated individuals.(Give them support environment and trust to get the job done).
6. Most efficient and effective method of conveying information in a development team is face-to face conversation.
7. Working software is primary measure of progress.
8. Agile processes promote sustainable development. (Users, sponsors, developers should maintain a constant pace).
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity is essential. (Art of maximum amount of work not done).
11. Self-organizing teams are required for best architectures, requirements and designs.
12. At regular intervals, team will tune and adjust its behavior to become more effective.

**Q: What is an Agile Process?**

**Ans:** Any Agile software process has 3 assumptions about software projects:

i. It is difficult to predict in advance which software requirements, customer priorities will change and which will persist.

ii. For many types of software, design and construction are interleaved (performed together). It is difficult to predict how much design is necessary before construction is used.

iii. Analysis, design, construction and testing are not as predictable as we might like.

**Q: How do we create a process that can manage unpredictability?**
**Ans:** Agile process must be adaptable, to have process adaptability. An agile software process must adapt incrementally. Customer feedback will make the process effective. Software increments must be delivered in short time periods, so that adaption keeps pace with change (unpredictability).
**Human Factors:** Agile development focuses on the talents and skills of individuals, modeling the process to specific people and teams.
**Traits that must exist among people of Agile Team:**

1. **Competence:** It encompasses innate talent, specific software knowledge of the process which the team applies. Skill and knowledge of process should be taught to all agile team members.
2. **Common Focus:** Although, Agile team members perform different tasks and bring different skills to be project, all should be focused on one goal-to deliver a working software increment to the customer within the time promised.
3. **Collaboration:** Team members must collaborate with one another, with customer and with business managers, as Software Engineering is
   1) Assessing, analysing, using information that is communicated to software team.
   2) Creating information that will help customer.
   3) Building information (DBs) that provides business value for customer.
4. **Decision Making Ability:** Agile team is given autonomy decision making authority for both technical and project issues.
5. **Fuzzy Problem-Solving Ability:** Agile team will continually have to deal with ambiguity and changes. Lesson learned any problem solving activity benefits the team later in the project.
6. **Mutual Trust and Respect:** Agile team should be a "Jelled" team. Jelled team exhibits the trust and respect requirement for the project.
7. **Self-Organisation**: It implies 3 things:
   a) Agile team organises itself for the work to be done.
   b) Agile team organises the process to best accommodate its environment.
   c) Agile team organises the work schedule to achieve project delivery.

## AGILE PROCESS MODELS: Many similarities among these approaches.

1) **EXTREME PROGRAMMING (XP):** XP uses Object-Oriented approach for development. The four Framework activities are: Planning, Design, Coding and Testing.

1. **Planning:** Planning begins with creation of a set of stories that describe required features and functionality for software to be built. Each story is written by customer and is placed on an index card. Customer assigns a value (priority) to it based on business value of it. XP team members then assess each story and assign a cost measured in development weeks to it. If the story will require more than 3 development weeks, customer is asked to split it into smaller stories, assignment of value and cost will occur again.

   Once a basic commitment (agreement on stories to be included, delivery date and other project matters) is made for a release, XP team orders stories that will be developed in one of three ways:
   1. All stories will be implemented immediately.
   2. Stories with highest value will be implemented first.
   3. Riskiest stories will be moved up in schedule and implemented first.

After first project release (software increment) has been delivered, XP team computes project velocity. Project velocity is number of customer stories released during first release. It is used to:

a) Estimate delivery dates and schedule for subsequent releases.
b) Determine whether an over-commitment has been made for all stories across development project. If so, content of releases is modified or end-delivery dates are changed.

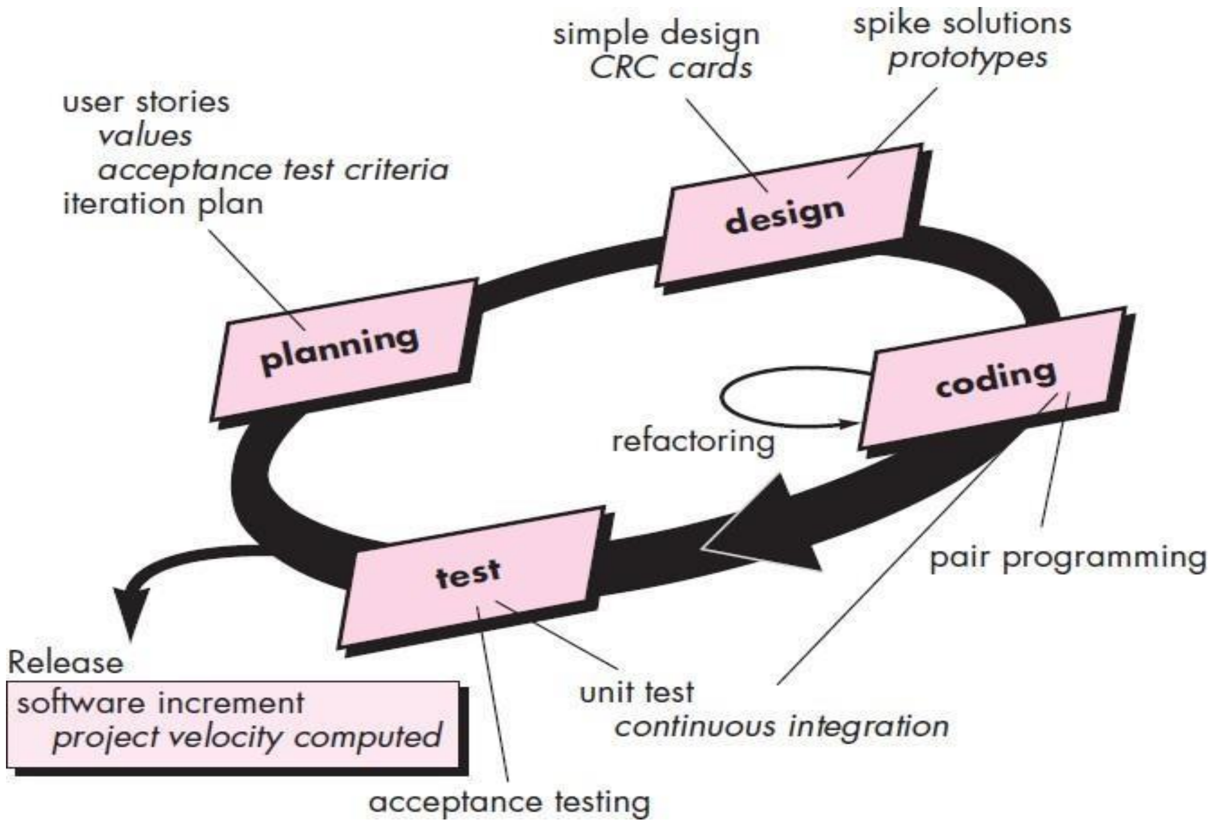


**Figure:** XP Model.

2. **Design:** XP design follows KIS (Keep It Simple) principle. A simple design is always preferred over more complex representation. XP encourages use of CRC (Class-Responsibility Collaborator) to identify and organise object-oriented classes that are relevant to current software increment. CRC cards are only design work produce in XP process.

   If a difficult design problem of software is encountered as part of 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 which can rapidly improve the design. "Refactoring is the process of changing a software system so that it does not alter external behaviour of code yet improves the internal structure". With refactoring the design occurs continuously as the system is constructed.

3. **Coding:** According to XP after stories are developed and design work is done, the team should not move to coding, but develop unit tests on stories to be included in the current release. So, the developer can focus on what must be implemented to pass the unit tested immediately.

   XP recommends that two people work together at one workstation (system) to create code for a story. This concept is known as ***pair programming***. This helps in real-time problem solving and real-time quality assurance. For example, one person might think about cooling details, while other ensures coding standards are high.

As pair programmers complete their work, their code is integrated within the work of others. This "Continuous Integration" helps to avoid compatibility and interfacing problem and provides a "smoking testing/ smoke testing" environment that helps to uncover errors early.

4. **Testing:** The unit tests that are created should be implemented easily and repeatedly. This encourages a Regression Testing strategy whenever code is modified.
   - *Regression Testing* is the re-execution of same subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.
   - *Integration and validation testing* 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.

2) **ADAPTIVE SOFTWARE DEVELOPMENT (ASD):** This model was proposed by Jim HighSmith. This is the best technique for building complex software and systems. ASD focus on human collaboration and self-organization. ASD life cycle has 3 phases:

1. **Speculation:** Project is initiated and adaptive cycle planning is conducted. Adaptive cycle planning uses customer is mission statement, project constraints (delivery dates etc) and basic requirements to define set of release cycle in the project.
2. **Collaboration:** Motivation of people to work together in a way that multiples their talent and creative output. Collaboration is not easy, as it is not just communication. It is a matter of trust. People working together must trust one another to:
   - **a.** Criticize without animosity (strong dislike)
   - **b.** Assist without resentment (feeling of displeasure)
   - **c.** Work as harder as they do
   - **d.** Have the skill set to contribute to the work at hand
   - **e.** Communication problems (concerns in a way that leads to effective action).

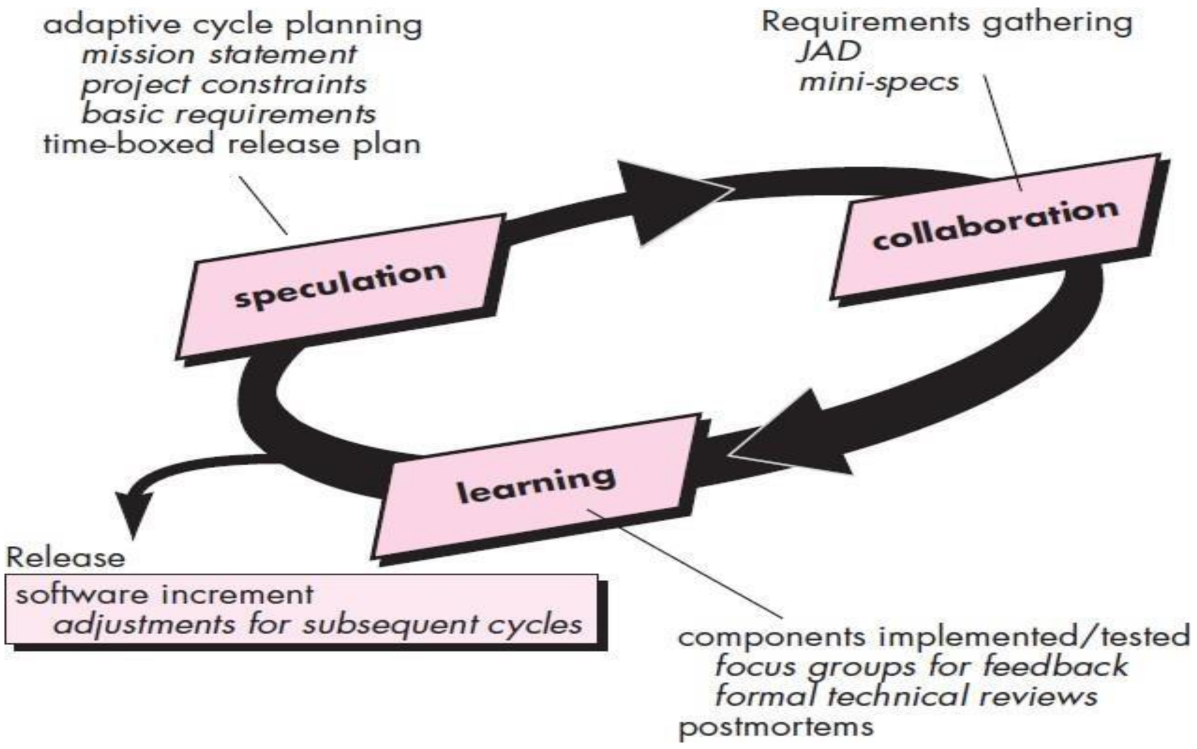


**Fig:** Adaptive Software Development

3. **Learning:** Software development may often over estimate their own understanding and learning will help them to improve their level of real understanding. ASD teams learn in 3 ways:
   a) **Focus Groups:** The customer lends users provide feedback on software increments that are being delivered. This provides direct indication of whether the product is satisfying business needs or not.
   b) **Formal Technical Reviews (FTRS):** ASD team members review the software components that are developed, improving quality and learning as they proceed.
   c) **Post-mortems:** ASD team becomes introspective (self thinking) addressing its performance and process. (With the intent of learning and then improving its approach).

3) **DYNAMIC SYSTEMS DEVELOPMENT METHOD (DSDM):** This approach "provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment".
   **Ex:** 80% of an application can be delivered in 20% of time it takes to deliver complete application.

Like XP, ASD, DSDM also suggests an iterative software process. DSDM approach to each iteration follows 80% rule, where much of the detail can be completed when more business requirement/changes are known.

DSDM Consortium is worldwide group of member companies, which uses DSDM approach. DSDM lifecycle defines 3 different iterative cycles, preceded by 2 additional life cycle activities.

1. **Feasibility Study:** Establishes business requirements and application constraints and then assesses whether the application is viable candidate for DSDM process.
2. **Business Study:** Establishes functional information requirements that allow the application to provide business value. Defines basic application architecture and identifies maintainability requirements for the application.
3. **Functional Model Iteration:** Produces a set of incremental prototypes that demonstrate functionality for the customer. It helps in gathering additional requirements from user feedback who exercises the prototype.
4. **Design and Build Iteration:** Revisits prototypes built during functional model iteration to ensure that they provide business value for end-users. Often occurs concurrently with Functional Model Iteration.
5. **Implementation:** Places latest software increment into operational environment. It should be noted that
   a) Increment may not be 100% complete.
   b) Changes may be requested as increment is put in place.

   In both cases, DSDM development work continues by returning to Functional Model Iteration activity.

❖ DSDM can be combined with XP to provide a combination approach that defines a solid process model.

4) **CRYSTAL:** Alistair Cockburn and Jim Highsmith created the "crystal family of agile methods", to achieve a software development approach that focuses on "manoeuvrability"-"a resource-limited, cooperative game of invention and communication, with a primary goal of delivering useful, working software and secondary goal of setting up for the next game".

- Crystal family is a set of agile processes that are effective for different types of projects.
- The intent is to allow agile teams to select the member of crystal family that is most appropriate for their software project and environment.

5) **SCRUM:** (Name derived from an activity during "RUGBY"**)**. Developed by Jeff Sutherland and team in early 1990's.

**Principles:**

1. Small working teams are organized to "maximize communication, minimize overhead and maximize sharing of tacit, informal knowledge".
2. Process must be adaptable to both technical and business changes "to ensure best possible product is produced."
3. Process yields frequent software increments "that can be inspected, adjusted, tested, documented and build on".
4. Developed work and teams are partitioned "into clean low coupling packets".
5. Constant testing and documentation is performed as the product is built.
6. Scrum process provides the "ability to declare a product 'done' whenever required".

Scrum principles are used to guide development activities within a process that incorporates the framework activities: Requirements, analysis, design, evolution and delivery. Scrum allows us to build softer software.

❖ With each framework activity, work tasks occur within a process pattern called a Sprint.

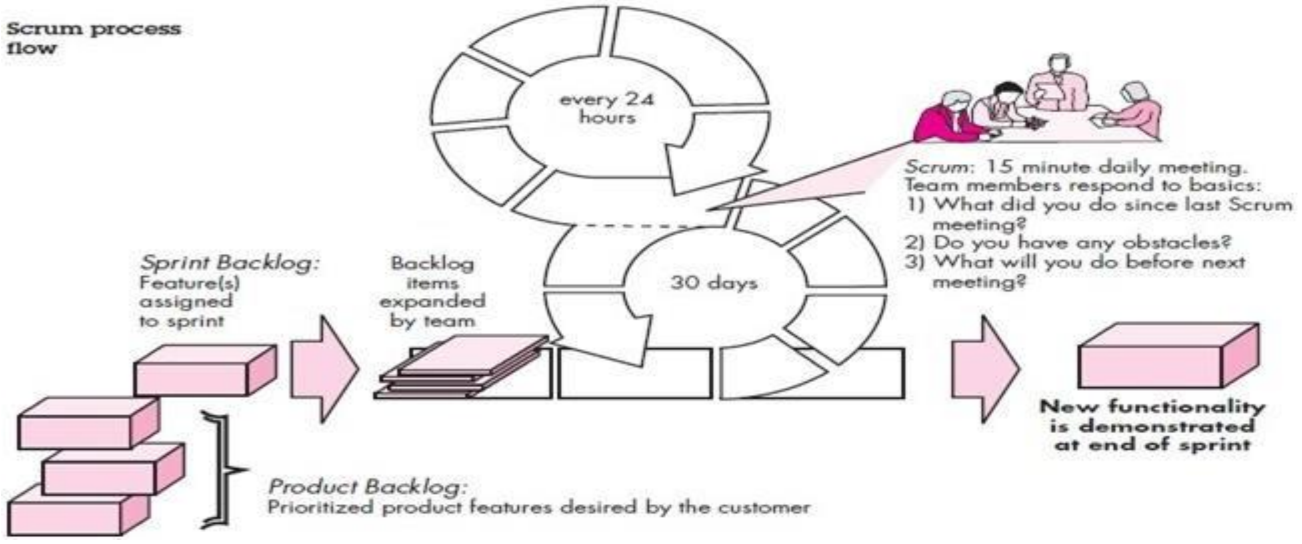

Scrum process flow
every 24 hours
Scrum: 15 minute daily meeting. Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
30 days
New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

**Fig:** Scrum process flow

Scrum emphasizes the use of a set of "software process patterns" that have proven effect for the projects with tight timelines, changing requirements and business criticality. It includes following development activities:

1) **Backlog:** "A prioritized list of project requirements or features that provide business value for the customer". Product manager assesses each backlog and updates priorities as required.
2) **Sprints: "**Consists of work units that have to achieve a requirement defined in the backlog that must be fit into a predefined time-box (30 days)".As changes are not introduced during the sprint, it allows team members to work in a short-term, but stable environment.
3) **Scrum Meeting:** "Short (15 minutes) meeting held daily by the scrum team. A team leader, called a "scrum master" leads the team meeting and assesses responses from each person. The key question asked and answered by all team members are:
    - What did you do since last meeting?
    - What obstacles are you encountering?
    - What do you plan to accomplish by next meeting?

These daily meetings help to know the problems in the team and lead to "knowledge socialization" and so promote team structure.

***Demos:*** "Delivers the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer". Demo may not contain all planned functionality.

6) **FEATURE DRIVEN DEVELOPMENT (FDD):** This model is also called as Features Design and Development. This is the Process model for Object-Oriented Software Engineering (OOSE). It is an Adaptive, Agile process that can be applied for moderately sized and largest software projects.

- In FDD, a feature "is a client-valued function that can be implemented in two weeks or less".

**Benefits:**

1. As features are small blocks of deliverable finality, users can describe, understand the relation and review them easily for ambiguity or errors.
2. Features can be organized into a hierarchical business-related grouping.
3. The team develops operational features every 2 weeks.
4. Design and code representations are easier to inspect effectively.
5. Project planning, scheduling and tracking are driven by hierarchy.

Code and his colleagues suggested a template for defining a feature:
<Action> the<result><by/for/of/to> a(n) <object>where <object> is a person, place (or) thing.
*Ex. of Features*: Add the product to a shopping cart. Display technical specification of product. Store the shipping information for a customer.

❖ A feature set group related features in to business related categories.

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

Ex: Making a product sale is feature set for above features.

- FDD approach defines five "collaborating" framework activities. These are also called as "Processes" in FDD.

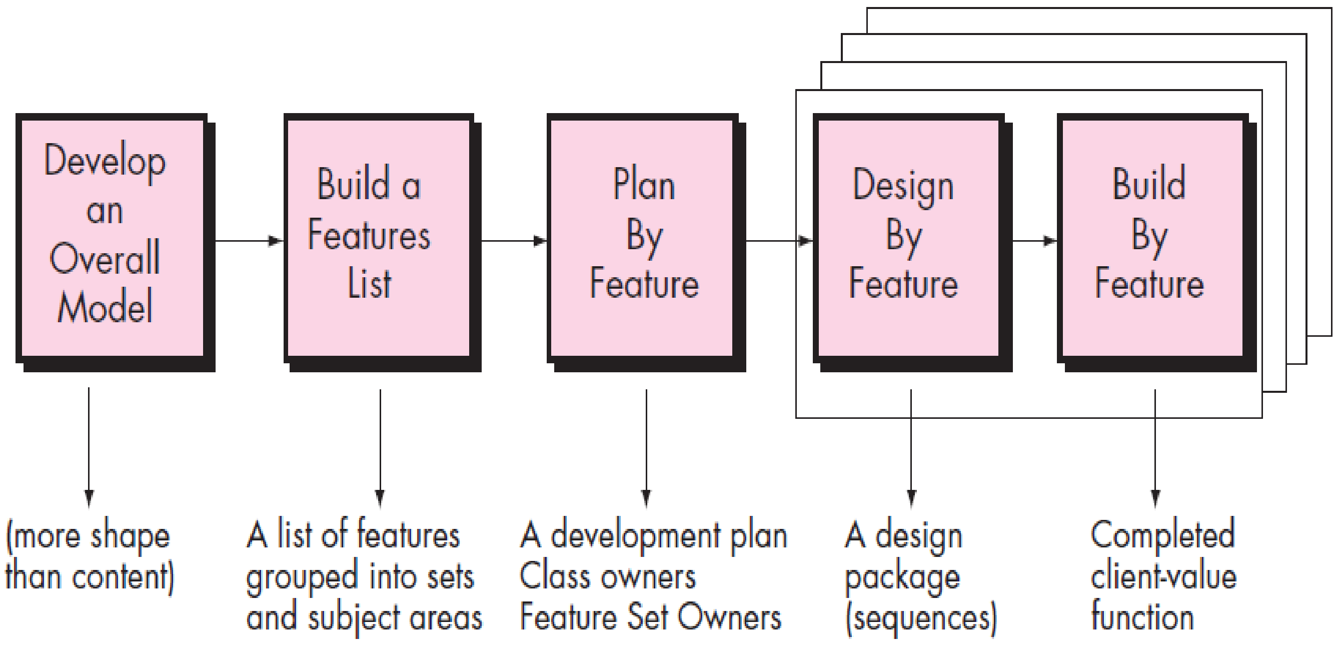

Develop an Overall Model
Build a Features List
Plan By Feature
Design By Feature
Build By Feature
(more shape than content)
A list of features grouped into sets and subject areas
A development plan Class owners Feature Set Owners
A design package (sequences)
Completed client-value function

**Fig:** FDD process.

FDD provides greater emphasis on project management guidelines and techniques than many other agile methods. If deadline pressure is significant, it is critical to determine if software increments (features) are properly scheduled. To accomplish this, FDD defines six milestones during design and implementation of features: "Design walk through, Design, Design Inspection, Code, Code Inspection, and Promote to build."

**AGILE MODELING (AM):** There are many situations in which software engineers must build large, business critical systems. Scope and complexity of such systems must be modeled so that,

- All constituencies can better understand what to be accomplished.
- Problem can be effectively partitioned among Software Engineers.
- Quality can be assessed at every step of system.

Agile Modeling is a practice-based methodology for effective Modeling and documentation of software-based systems. Agile Modeling is a collection of values, principles and practices for modeling effective software.

An Agile team must be courageous to reject any requirement, design and need to re-factor. It must have all answers, business experts and other stakeholders should be respected and embraced.

Modeling principles that make Agile Modeling unique are:

1) **Model with a Purpose:** A software engineer who uses AM should have a specific goal in mind before creating the model. Once a goal of model is identified, type of notation and level of details required will be more obvious.
2) **Use Multiple Models:** Agile Modeling suggests that each model should present a different aspect of the system and only models provide value to their developers should be used.
3) **Travel Light:** As Software Engineering work proceeds, keep only those models that will provide long-term value and discard the rest. Every work product that is kept, must be maintained as changes occur. It is required to look for best possible model from various sources.
4) **Know the models and tools used to create them:** Understand the tools used to create the models and also strengths and weaknesses of each model.
5) **Adapt Locality:** Modeling approach should be adapted to the needs of agile team.
6) **Content is more important than representation:** A perfect model that imports little useful content is not as valuable as a flawed notation with valuable content. So, focus should be on the content in model.