UNIT - I

Introduction to Software Engineering:
A generic view of process: Software Engineering , process framework, The Nature of Software, Software Engineering, Software Myths.
Process Models: A Generic Process Model
An Agile view of Process: Introduction to Agility and Agile Process.

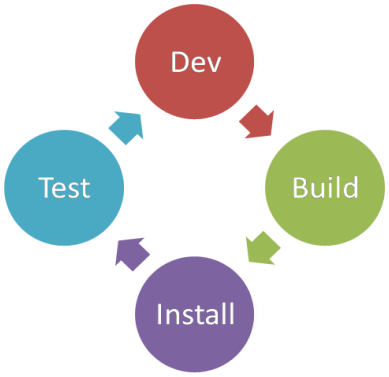
Introduction to Software Engineering

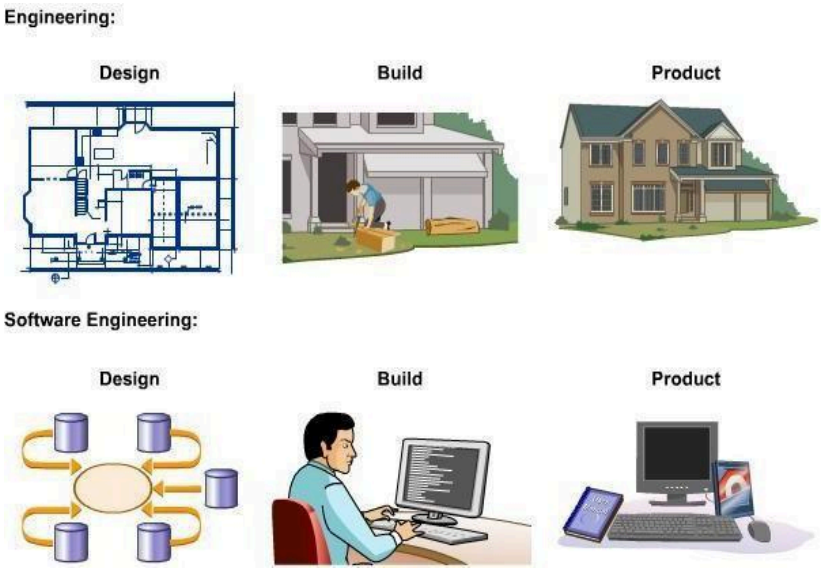
Software engineering is an engineering discipline that is concerned with all aspects of software development and production. We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines. Software engineers should adopt a systematic and organized approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

Definition of SOFTWARE ENGINEERING

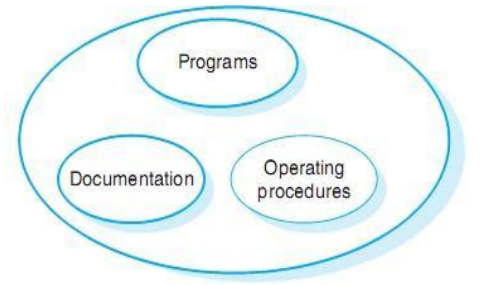
SE is defined as systematic, disciplined and quantifiable approach for the development, operation and maintenance of software by applying engineering principles.

- Software engineering can be defined as —The establishment and use of sound engineering principles in order to obtain economically software that is reliable and woks efficiently on real time machines.‖





Software is defined as computer programs, procedures, rules and possibly associated documentation and data pertaining to the operation of a computer based systems. —Computer Softwarell is synonymous with —software productll.



Software = Program + Documentation + Operating Procedures

Another definition of Software is

- Instructions
  - Programs that when executed provide desired function
- Data structures
  -Enable the programs to adequately manipulate information
- Documents
  -Describe the operation and use of the

programs Characteristics of software

- Software is developed or engineered;
- it is not manufactured in the classical sense.
- Software does not wear out. However it deteriorates due to change.
- Software is custom built rather than assembling existing components.

Categories of Software

Seven Broad Categories of software are challenges for software engineers

- System software
- Application software
- Engineering and scientific software
- Embedded software
- Product-line software
- Web-applications
- Artificial intelligence software

Legacy software are older programs that are developed decades ago. The quality of

legacy software is poor because it has in-extensible design, convoluted code, poor and nonexistent documentation, test cases and results that are not achieved.

As time passes legacy systems evolve due to following reasons:

- The software must be adapted to meet the needs of new computing environment or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it interoperable with more modern systems or database
- The software must be re architected to make it viable within a network environment.

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 tools 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

PROCESS FRAMEWORK:

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:** 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.

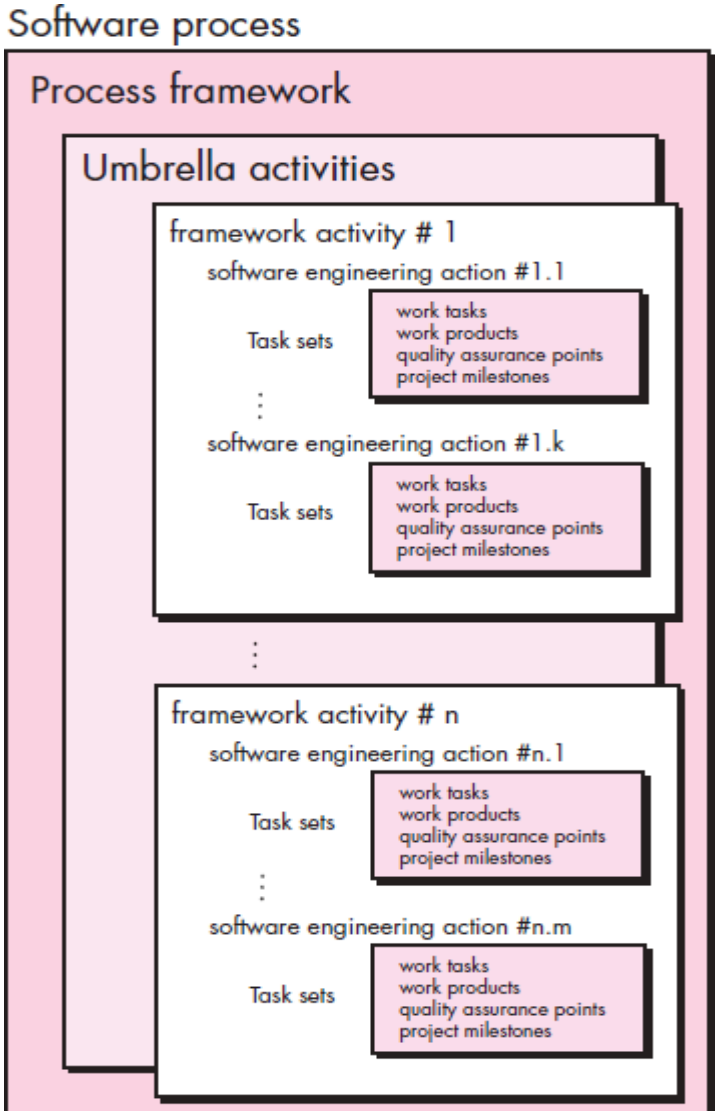


Fig: Process Framework

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.

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:
- **Task Patterns:** Software Engineering action or work task that is part of process and relevant to successful Software Engineering practice defined.
- **Stage Patterns:** Defines "a framework activity with multiple work tasks in it", for the process. Ex: Communication contains Requirement Gathering
- **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.

THE NATURE OF SOFTWARE :

Software is:

(1) instructions (computer programs) that when executed provide desired features, function, and performance;

(2) data structures that enable the programs to ad- adequately manipulate information, and

(3) descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

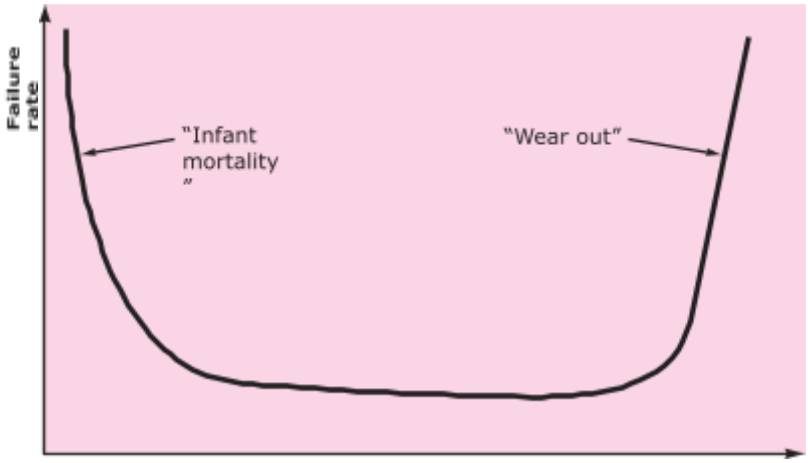
Characteristics of software:

1. *Software is developed or engineered; it is not manufactured in the classical sense.*

Although some similarities exist between software development and hard- ware manufacturing, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent.

2. *Software doesn't "wear out."*

Failure curve for hardware



Failure curve for software

Time

3. *Although the industry is moving toward component-based construction, most software continues to be custom built.*

A software component should be designed and implemented so that it can be reused in many different programs. Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts.

Software Application Domains:

System software—a collection of programs written to service other pro- grams. Some system software (e.g., compilers, editors, and file management utilities) processes complex, but determinate, information structures. Other systems applications (e.g., operating system components, drivers, networking software, telecommunications processors) process largely indeterminate data.

Application software—stand-alone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision making. In addition to conventional data processing applications, application software is used to control business functions in real time (e.g., point-of-sale transaction processing, real-time manufacturing process control).

Engineering/scientific software—has been characterized by “number crunching” algorithms. Applications range from astronomy to volcanology, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

Embedded software—resides within a product or system and is used to implement and control features and functions for the end user and for the system itself. Embedded software can perform limited and esoteric functions (e.g., key pad control for a microwave oven) or provide significant function and control capability (e.g., digital functions in an automobile such as fuel control,

dashboard displays, and braking systems).

Product-line software—designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications).

Web applications—called "WebApps," this network-centric software cate- gory spans a wide array of applications. In their simplest form, WebApps can be little more than a set of linked hypertext files that present information using text and limited graphics.

Artificial intelligence software—makes use of nonnumerical algorithms to solve complex problems that are not amenable to computation or straightfor- ward analysis. Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

Legacy Software

Legacy software systems were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms. The pro- liferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve.

legacy systems often evolve for one or more of the following reasons:

- The software must be adapted to meet the needs of new computing environ- ments or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it interoperable with other more modern systems or databases.
- The software must be re-architected to make it viable within a network environment.

Software Myths

Software Myths- beliefs about software and the process used to build it - can

be traced to the earliest days of computing. Myths have a number of attributes that have made them insidious. For instance, myths appear to be reasonable statements of fact, they have an intuitive feel, and they are often promulgated by experienced practitioners who—know the score.

Management Myths

Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. A software manager often grasps at belief in a software myth, If the Belief will lessen the pressure.

Myth : *We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?*

Reality : The book of standards may very well exist, but is it used?

- Are software practitioners aware of its existence?
- Does it reflect modern software engineering practice?
- Is it complete? Is it adaptable?
- Is it streamlined to improve time to delivery while still maintaining a focus on Quality?

In many cases, the answer to these entire question is no.

Myth : *If we get behind schedule, we can add more programmers and catch up (sometimes called the Mongolian horde concept)*

Reality : Software development is not a mechanistic process like manufacturing. In the words of Brooks —Adding people to a late software project makes it later.‖ At first,

this statement may seem counter intuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort

Myth : *If we decide to outsource the software project to a third party, I can just relax and let that firm build it.*

Reality : If an organization does not understand how to manage and control software project internally, it will invariably struggle when it out sources software project.

Customer Myths

A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing /sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths led to false expectations and ultimately, dissatisfaction with the developers.

Myth : *A general statement of objectives is sufficient to begin writing programs we can fill in details later.*

Reality : Although a comprehensive and stable statement of requirements is not always possible, an ambiguous statement of objectives is a recipe for disaster. Unambiguous requirements are developed only through effective and continuous communication between customer and developer.

Myth : *Project requirements continually change, but change can be easily accommodated because software is flexible.*

Reality : It's true that software requirement change, but the impact of change varies with the time at which it is introduced. When requirement changes are requested early, cost impact is relatively small. However, as time passes, cost impact grows rapidly – resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.

Practitioner Myths

Myth(1) ☾ Once we write the program and get it to work, our job is done—

Reality ☾ 60% to 80% of all effort expended on software occurs after it is delivered

Myth(2) ☾ Until I get the program running, I have no way of assessing its quality

Reality ☾ Formal technical reviews of requirements analysis documents, design documents, and source code (more effective than actual testing)

Myth(3) ☾ The only deliverable work product for a successful project is the working program—

Reality ☾ Software, documentation, test drivers, test results

Myth(4) ☾ Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down

Reality ☾ Creates quality, not documents; quality reduces rework and provides software on time and within the budget.

PROCESS MODELS:

A GENERIC PROCESS MODEL:

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:

- **Communication:** It involves heavy communication and collaboration with customer and other stakeholders. It encompasses requirements gathering and related activities.
- **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.
- **Modeling:** This activity focuses on creation of models that allow stakeholders (customer, developer) to better under software requirements and design that will achieve requirements.
- **Construction:** It combines code generation and testing to uncover errors in the code. (Manual, automated actions).
- **Development:** The software (completed/partial increment) is delivered to the customer for evaluation and feedback of it.

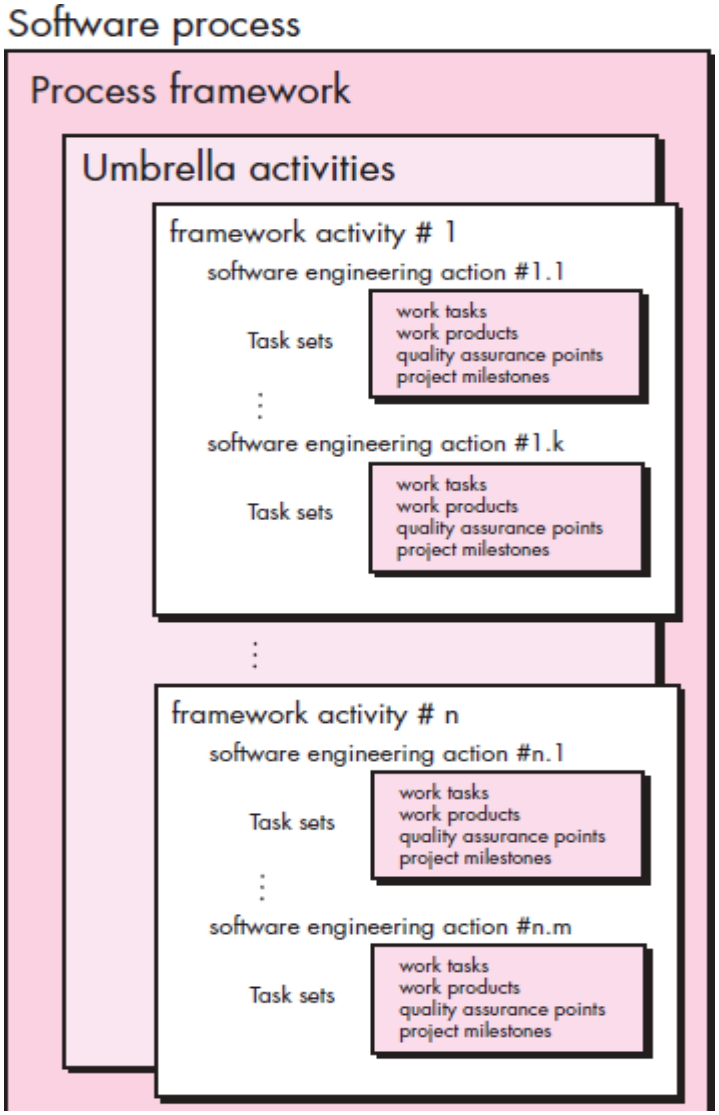


Fig: Process Framework

Umbrella Activities:

- ***Software Project Tracking and Control* –** Assess progress against the project plan and maintain schedule.
- ***Risk Management* –** Assesses risk that effect project outcome (or) Quality of product.
- ***Software Quality Assurance* –** Activities to ensure software quality.
- ***Formal Technical Reviews* -** Assess work products to uncover and remove errors before next action.
- ***Software Configuration Management* –** Manages effects of changes throughout the software process.
- ***Measurement* –** Defines and collects process, project, product measures to meet customer requirements.

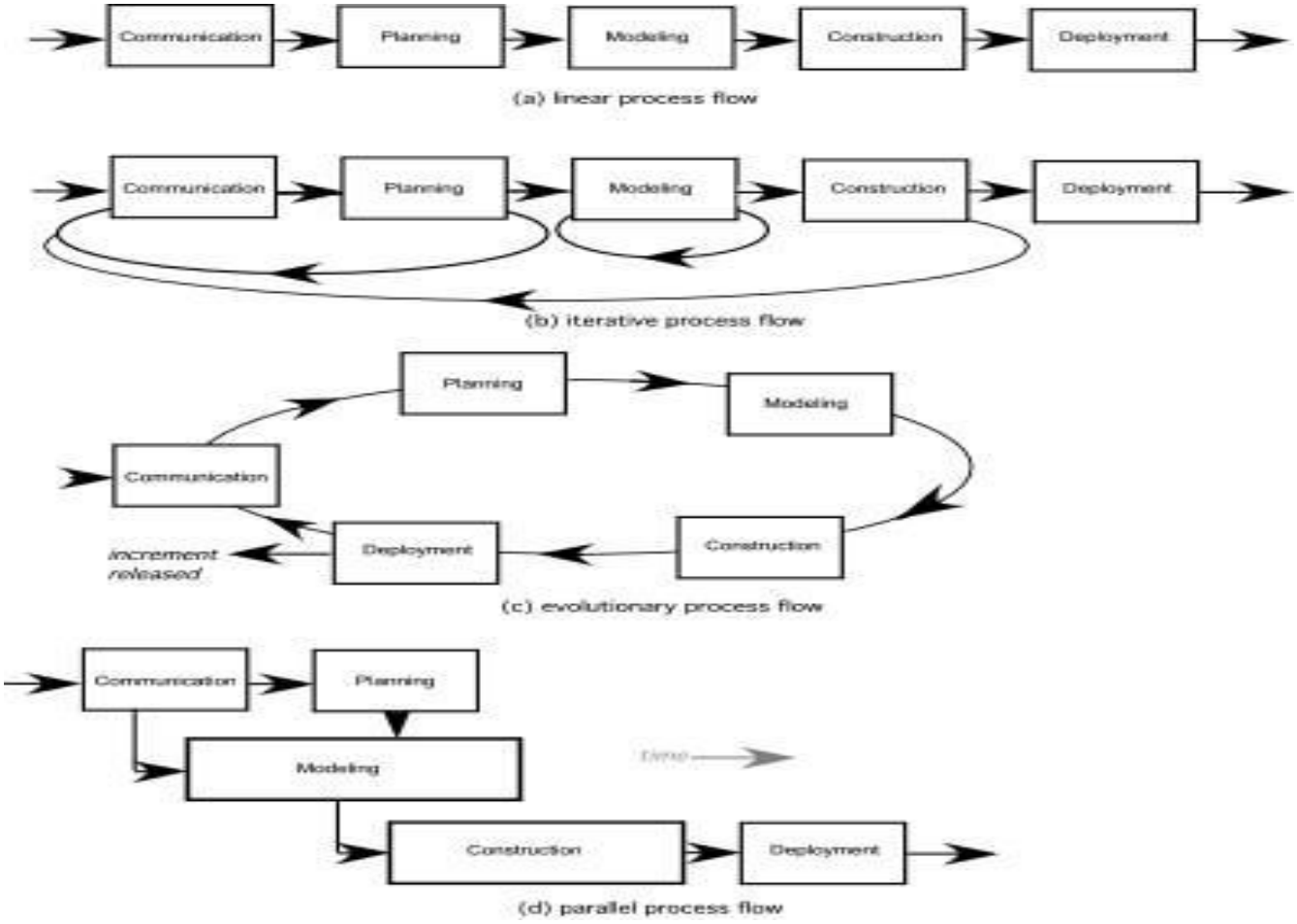
- ***Reusability Management*** **–** Defines criteria for work product reuse and establishes mechanisms to achieve reusable components.
- ***Work Product Preparation and Production*** **–** Focuses on activities required to create work product such as models, documents, logs, forms and lists.

IDENTIFYING A TASK SET:

A task set defines the actual work to be done to accomplish the objectives of a software engineering action.

- A list of the task to be accomplished
- A list of the work products to be produced
- A list of the quality assurance filters to be applied

PROCESS FLOW:



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:
- **Task Patterns:** Software Engineering action or work task that is part of process and relevant to successful Software Engineering practice defined.
- **Stage Patterns:** Defines “a framework activity with multiple work tasks in it”, for the process. Ex: Communication contains Requirement Gathering
- **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.

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.

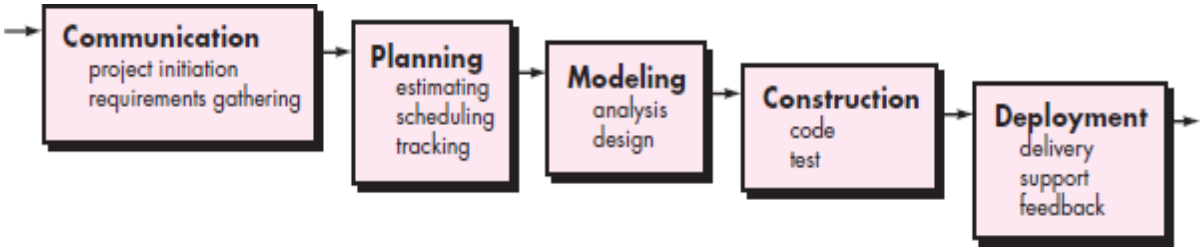


Fig: Waterfall Model.

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.

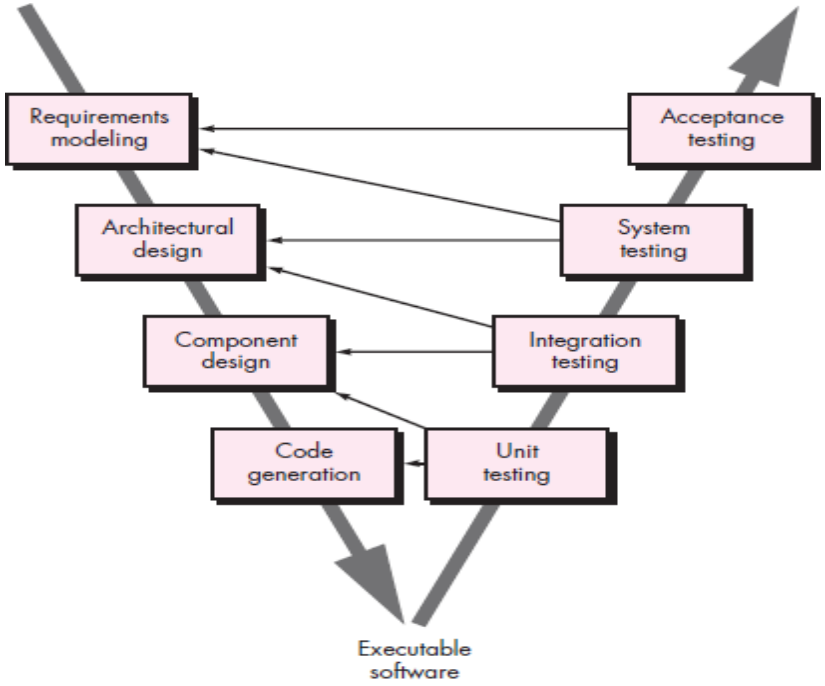


Fig: The V – Model

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.

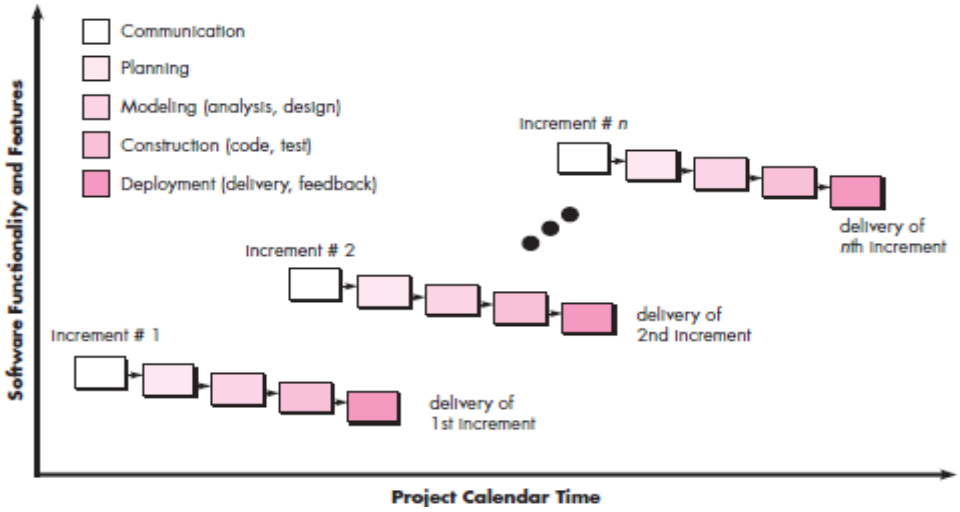


Figure: Incremental Process model.

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.

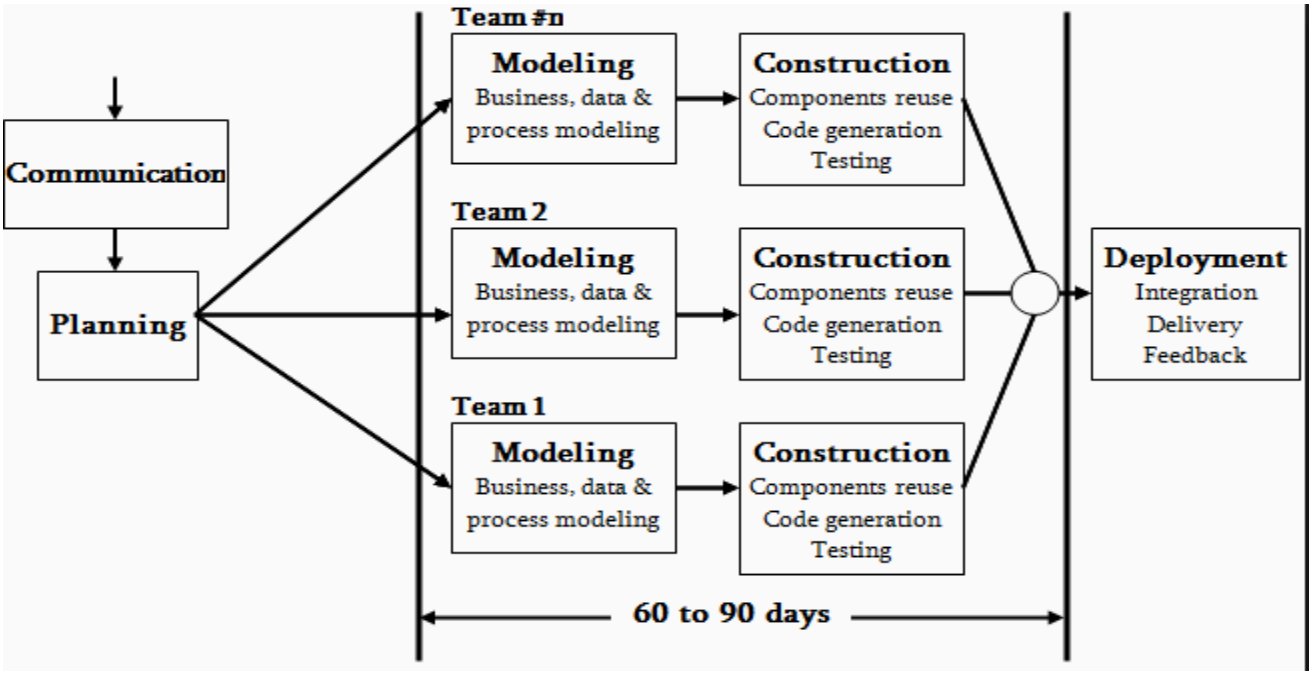


Fig: RAD Process model.

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.

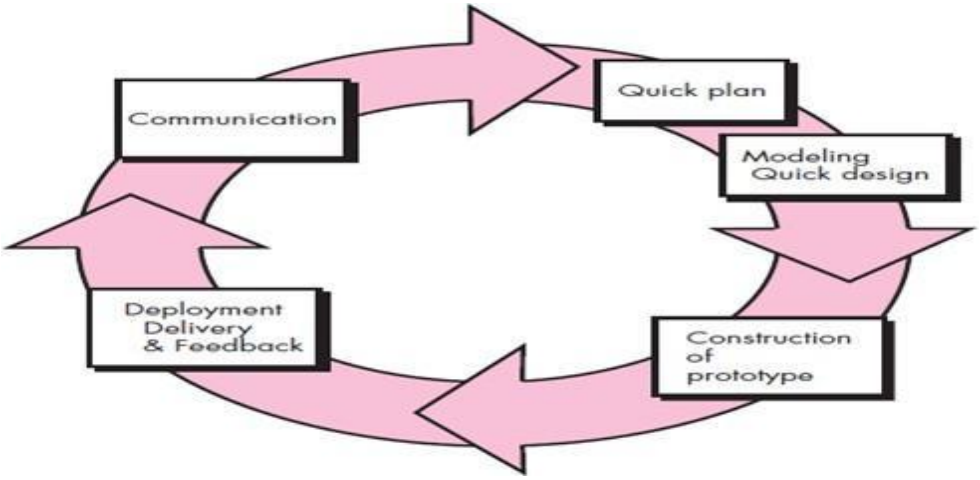


Fig: Prototyping Model

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) Cyclic Approach for increasing system's degree of definition and implementation while decreasing degree of risk-Risk is considered as each revolution is made.
2) Anchor Point Milestone 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.

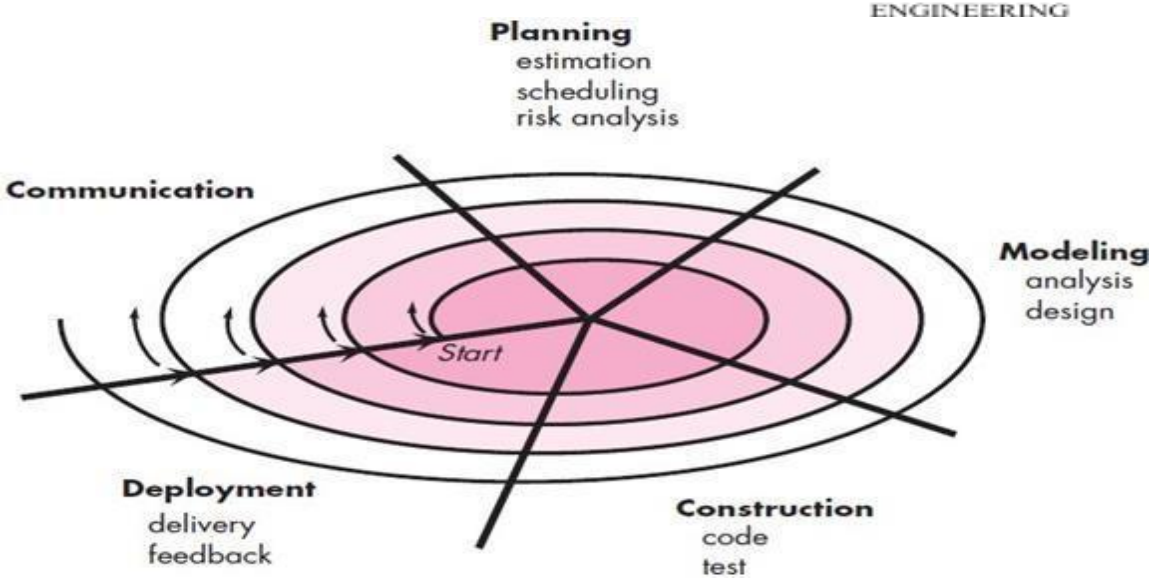


Fig: Spiral Model

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 is over.
- ✔ 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.

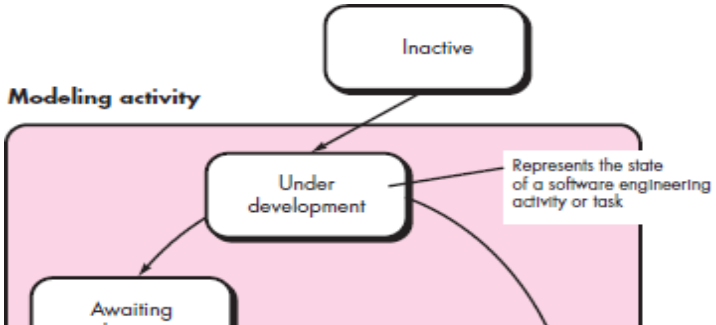


Fig: One element of concurrent model development model.

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.

AN AGILE VIEW OF PROCESS

Agility:

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.

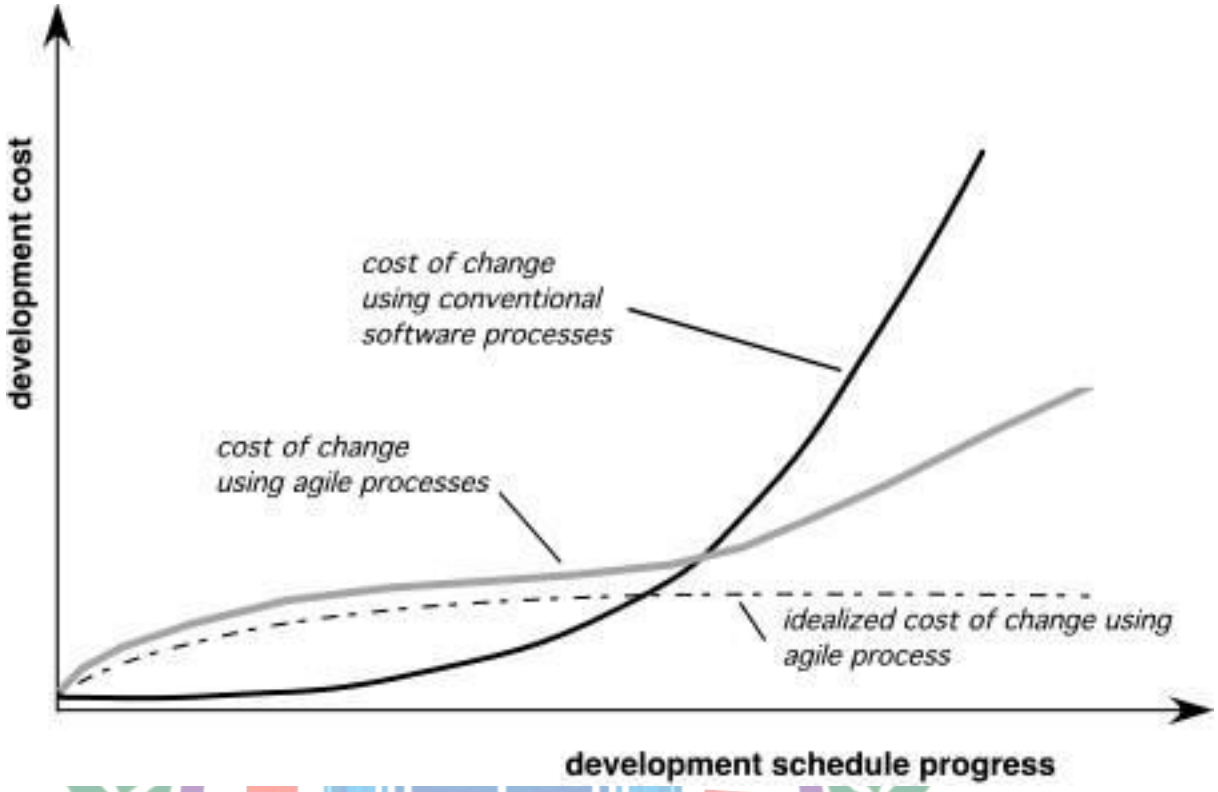
Agile Process:

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.



iv.

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.