What is Software?

Software can be described as a collection of instruction that is executed to get desired functionalities. It is different from a program in the sense that software includes programs, documentation and operating procedures.

Characteristics of software compared to hardware

- Software is developed, not manufactured.
- Does not wear out.
- Has reusable components.
- Flexible.

Types of software

System software

Software that provides a platform to other software such as Microsoft Windows, GNU/Linux, macOS.

Application Software

Programs designed for end users like Notepad, Firefox, Media players.

Engineering/Scientific Software

Software built with techniques and formulae specific to specific scientific or engineering field, examples are MATLAB, AUTOCAD.

Embedded Software

Programs that control machines or devices such as Traffic light control mechanism, Motorcycle dashboard display.

Product-line Software

A group of products sharing common features that are used to satisfy specific needs of a mission like MS Office supporting home use, business use, enterprise use etc.

Web Application Software

Programs that are run on web servers and are accessed through a web browser, examples are Gmail, Facebook etc.

Artificial Intelligence Software

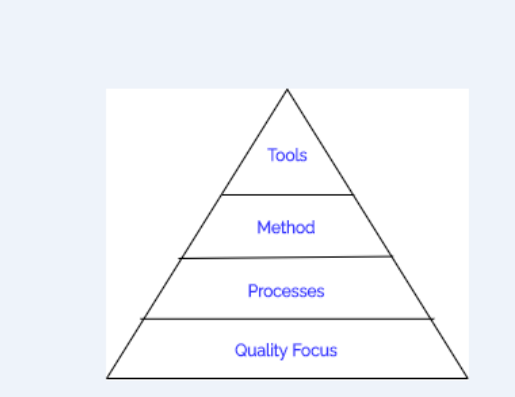
Software that is capable of intelligent behaviour, like in Robotics, Game Playing etc.

Legacy Software System

Software systems that were developed decades ago which becomes costly and risky to maintain and evolve.

Software Engineering

Software engineering can be defined as a systematic and disciplined engineering approach applied for building and maintaining software.



It is a layered technology comprising

- **Quality focus** is its central support element since improving the quality of process and products through quality improvement techniques is the organisation's main focus.
- **The process layer** provides a framework for the management of software project.
- **The method layer** includes methods such as communication, design, coding and other technical aspects for the development of software.
- **The tools layer** provides supporting tools for the Method and Process layers.

Software Engineering Practice
Essence of practice

General steps in carrying out software engineering are,

- Understand the problem through communication.
- Plan a solution by modelling and designing.
- Carry out the plan (coding).
- Examine result for correctness (testing).

Principles of best practice

- Software should provide value to users.
- Keep the software design simple.
- The vision of the project should be kept in mind throughout.
- Keep your work products in an understandable format for others to work with them easily.
- Design in a way that can incorporate changes in future.
- Build reusable components.
- Think before acting or learn before starting.

Software Myths (False Belief)

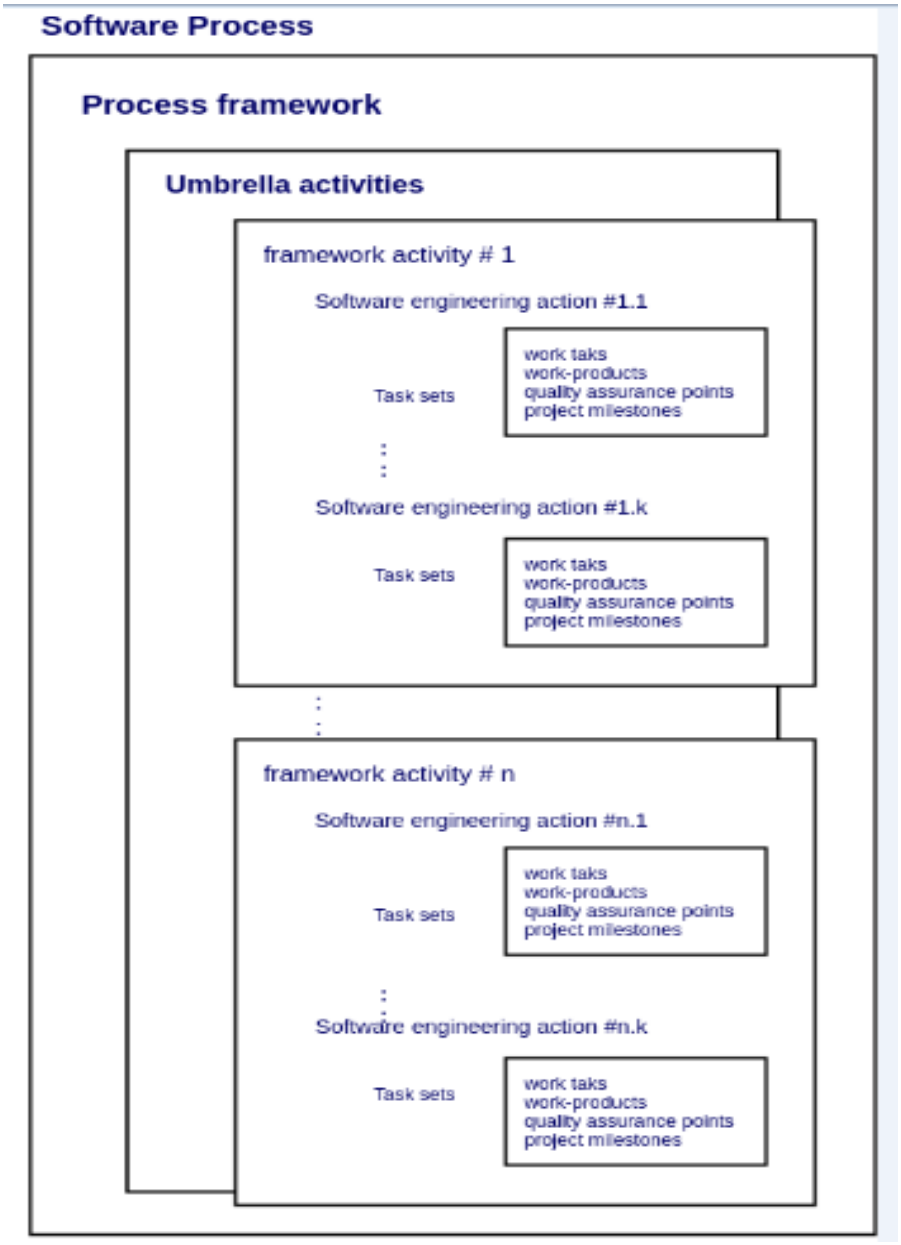
- **Management myths**
  • Software practitioners follow the book of standards and procedures for software development.
  • If a delay occurs, programmers can be added at any point.
  • Outsourcing helps.
- **Customer myths**
  • General requirements and not a detailed one is needed for starting programming.
  • Software is easy to change.
- **Practitioner's myths (Developer's myths)**
  • Software development is all about coding.
  • Only a finished software can be tested.
  • Software engineering has plentiful documentation and will slow down the process.

Software Process

The software process comprises activities performed to create a software product. It deals with the technical and management aspects of software development.

Software process includes :

- **Tasks** – focus on a small, specific objective.
- **Action** – set of tasks that produce a major work product.
- **Activities** – group of related tasks and actions for a major objective.



A process framework for software engineering defines five **framework activities**. Framework activities include communication, planning, modelling, construction and deployment. Each framework activity includes a set of engineering actions and each action defines a set of tasks that incorporates work products, project milestones and software quality assurance (SQA) points that are required. **Umbrella activities** are carried throughout the process.

Process Framework Activities

- **Communication** – Communicate with stakeholders and customers to obtain objectives of the system and requirements for the software.
- **Planning** – Software project plan has details of resources needed, tasks and risk factors likely to occur, schedule.
- **Modelling** – Architectural models and design to better understand the problem and for work towards the best solution.
- **Construction** – Generation of code and testing of the system to rectify errors and ensuring all specified requirements are met.
- **Deployment** – Entire software product or partially completed product is delivered to the customer for evaluation and feedback.

Umbrella activities

Activities that occur throughout a software process for better management and tracking of the project

- **Software project tracking and control** – Compare the progress of the project with the plan and take steps to maintain a planned schedule.
- **Risk management** – Evaluate risks that can affect the outcome and quality of the software product.
- **Software quality assurance (SQA)** – Conduct activities to ensure the quality of the product.
- **Technical reviews** – Assessment of errors and correction done at each stage of activity.
- **Measurement** – All the measurements of the project and product features.
- **Software configuration management (SCM)** – Controlling and tracking changes in the software.
- **Reusability management** – Back up work products for reuse and apply the mechanism to achieve reusable software components.
- **Work product preparation and production** – Project planning and other activities used to create work product are documented.

Process flow

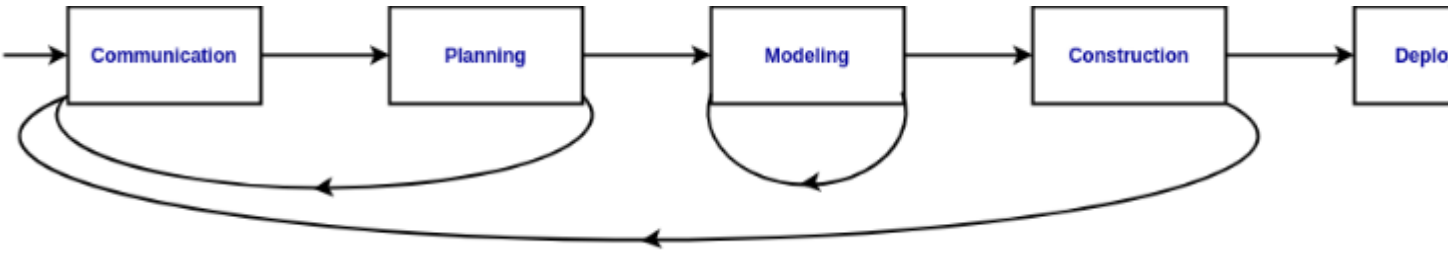
Process flow determines how activities, actions and tasks are arranged with respect to sequence and time.

Linear process flow

Linear process flow executes each activity in sequence.

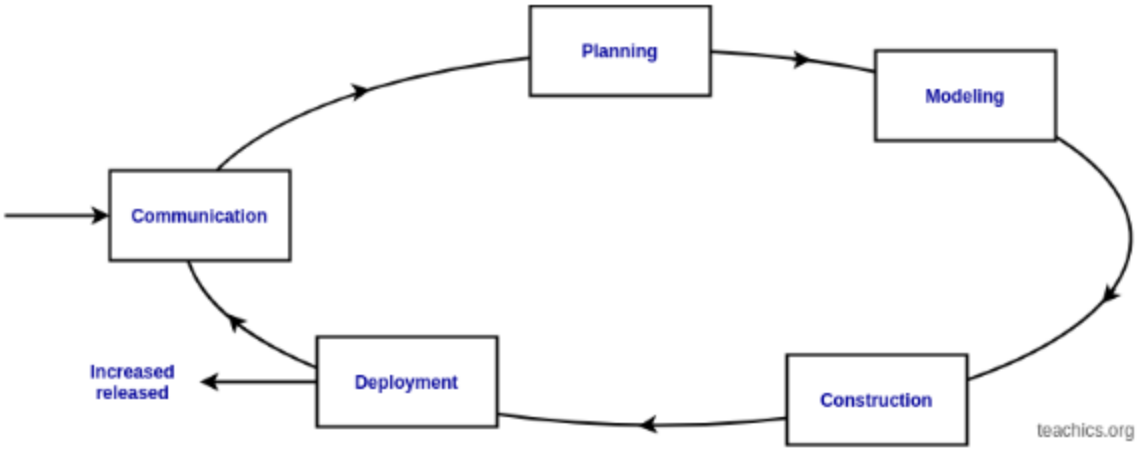
Iterative process flow

Iterative process flow repeats one or more activities before starting next.



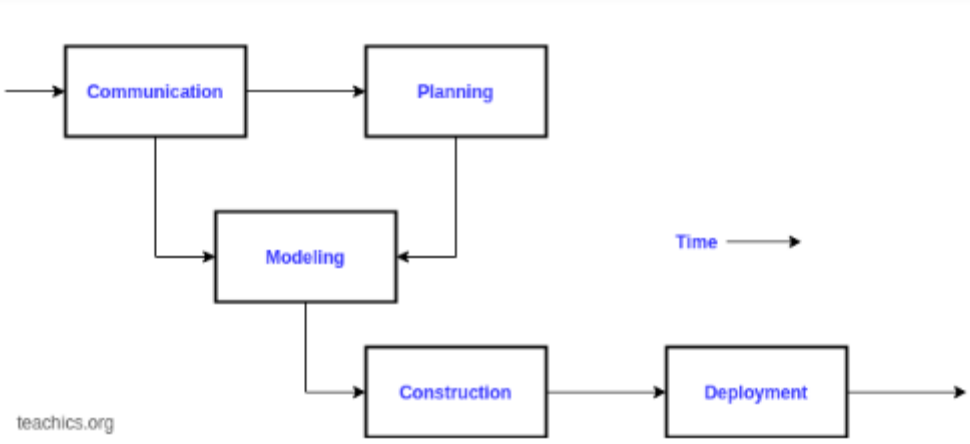
Evolutionary process flow

Evolutionary process flow carry out activities in a circular way.



Parallel process flow

Parallel process flow executes one or more activities in parallel with each other.



Defining a framework activity

Consider communication activity. For small project, this can be defined as having tasks set:

- Making a phone call with stakeholder.
- Discuss requirements and note them down.
- Organize requirements.
- Mail stakeholder for review and approval.

For large projects, this may have extended actions such as feasibility study, elicitation of requirements, elaboration of requirements, specification documents, validation etc.

Identifying a task set

Task set is the actual work to be done to achieve an objective of engineering action. For small project, consider **elicitation action in communication activity**, this may include :

- Prepare a list of stakeholders of the project.
- Organize a meeting for stakeholders.
- Discuss requirements.
- Finalize requirements list.
- Make a list of issues raised.

For large projects extraneous steps may be added to elicitation such as interviewing each stakeholder separately before the group meeting, more techniques are applied in discussing requirements…etc.

Process patterns

Process patterns are patterns used to describe problems and their solutions in the context of software process. Problems can arise at different levels such as :

- Problems associated with a complete process model
- Within a framework activity
- Within an action in an activity

Patterns can be described using a **pattern template** which include :

- **Pattern name**
- **Intent** (describes process pattern in one or two paragraphs)
- **Forces and Types** (environment in which problem is encountered is identified and the type of pattern is specified)
    - **Stage pattern** – explains problems related to framework activity and it may include multiple task patterns as well.
      Example: 'Establishing Communication' (stage pattern) includes 'Requirements Gathering'(task pattern)
    - Task pattern – describes problems related to software engineering action or task such as Requirements Gathering.
    - Phase pattern – defines a sequence of framework activities and ensures each section in the activity is addressed correctly such as in prototyping, spiral model etc.
- Initial context – Situation to which the pattern is applied. Defines work done so far before the pattern is applied. Actions and activities that have taken place before the pattern is introduced.
- **Problem** – Specific problem that is to be solved by the pattern.
- **Solution** – Implements pattern and initial context is modified to resolve the problem.
- **Resulting contex**t – Situation which will result from carrying out the process pattern solution.
- **Related patterns** – A list of patterns related to the current one is documented for further reference.
- **Known uses or examples** – Indicates where or how the process pattern is applicable.

***An example*:**

Pattern name – requirements unclear.
Intent – an approach to build a prototype so that stakeholder can assess and determine

specific requirements.
Type – phase pattern.
Initial context – stakeholders have been identified, communication mode has been selected, initial understanding of problem and scope of the project determined.
Problem – recognized that stakeholder has a general idea of requirements and cannot place specific requirements.
Solution – prototyping can help the stakeholder to be more specific about requirements and hence prototype can be evolved.
Resulting context – a prototype fulfilling basic requirements is approved by stakeholder and the prototype may be evolved or thrown for making a new one which will become the finished product.
Related patterns – customer communication, iterative design, requirement extraction.
Known uses/examples – unclear requirements problem can be solved through prototyping.

Prescriptive process models

Prescriptive process models prescribe a set of framework and other activities, quality assurance points, and software process-related elements. They define a workflow among these elements that shows their inter-relationship.

The process models described here are,

Waterfall Model.

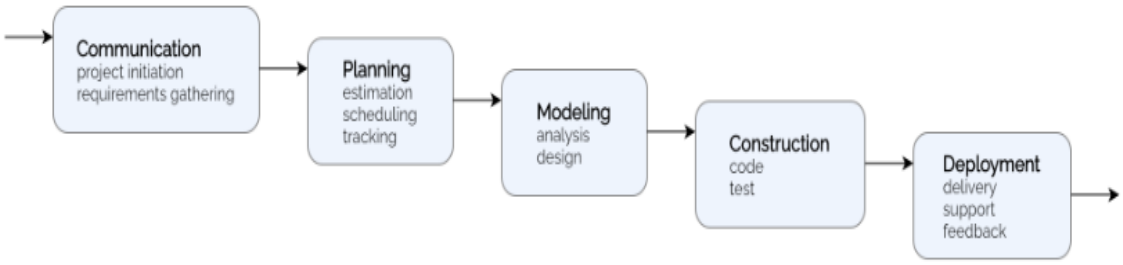
Incremental Process Model.

Evolutionary Process Model.

Spiral Model

The Waterfall Model

The Waterfall model is also known as ‘Linear sequential model‘ or ‘Classic life cycle model‘. It is used in small projects where requirements are well defined and known before starting the project. Activities are carried out in a linear and systematic fashion.



The Waterfall Model

The process starts with **communication**, where requirements are gathered from the customer and recorded.

Then goes to the **planning** stage where the cost and time constraints are estimated, a schedule is outlined and project tracking variables are defined.

Modeling is where a design based on the requirements and keeping the project constraints in mind is created. After this, code is generated and the actual building of the product is started in the construction phase.

Testing (unit testing, integration testing) is done after code completion in this phase.

Deployment is the last stage where the product is delivered, customer feedback is received and, support and maintenance for the product are provided.

Advantages of the waterfall model

- A simple model to use and implement.
- Easily understandable workflow.
- Easy to manage since requirements are known prior to the start of the project.
- Can be applied to projects where quality is preferred over cost.

Disadvantages of the waterfall model

- It may be difficult for the customer to provide all the specific requirements beforehand.
- Cannot be used for complex and object-oriented projects.

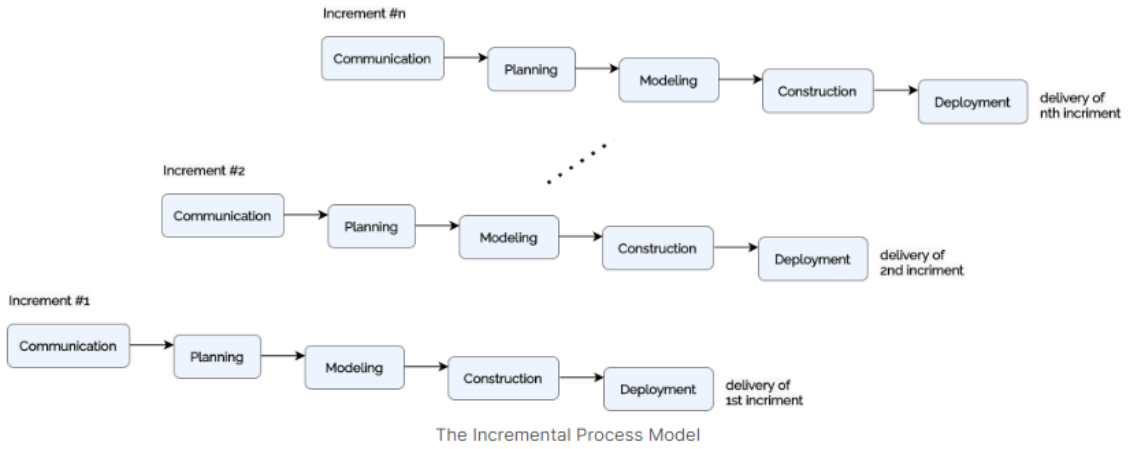
- Testing and customer evaluation are done at the last stages and hence the risk is high.
- Iteration of activities is not promoted which is unavoidable for certain projects.
- May lead to “blocking states” in which some project team members must wait for other members of the team to complete dependent tasks.

Incremental Process Model

The Incremental process model is also known as ‘Successive version model‘.

In the Incremental process model, a series of releases, called increments, are built and delivered to the customer.

First, a simple working system(core product), that addresses basic requirements, is delivered. Customer feedback is recorded after each incremental delivery. Many increments are delivered, by adding more functions, until the required system is released. This model is used when a user demands a model of product with limited functionality quickly.



The Incremental Process Model

Advantages of incremental process model

- Flexible to change requirements.
- Changes can be done throughout the development stages.
- Errors are reduced since the product is tested by the customer in each phase.
- Working software available at the early stage of the process.

- Easy to test because of small iterations.
- The initial cost is lower.

Disadvantages of incremental process model

- Requires good planning and design.
- Modules and interfaces should be well defined.
- The total cost is high.
- Demands a complete planning strategy before commencement.
- Refining requirements in each iteration may affect system architecture.
- Breaking the problem into increments needs skilful management supervising.

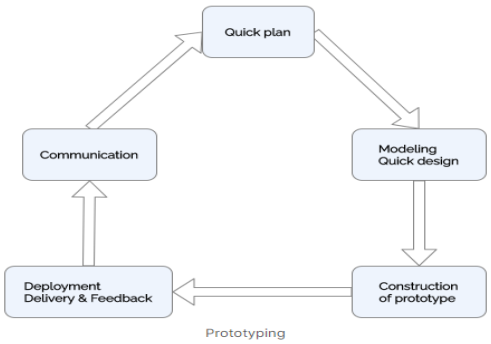
Evolutionary Process Models

Evolutionary process models are opted when the requirements may tend to change and also when the complete sophisticated product delivery cannot be done before a given deadline, but the delivery of a limited version of it is possible.

In the incremental model, complete requirements are specified beforehand and these requirements are refined over time for each increment. The evolutionary model permits requirements, plans and estimates to evolve over time. Here we discuss prototyping and the spiral model.

Prototyping

In cases when the requirements are unclear and are likely to change or when the developer is doubtful about working of an algorithm, a solution is to build a prototype and find out what is actually needed. Hence, in this model, one or more prototypes are made with unrefined currently known requirements before the actual product is made.



Prototyping

A quick design is what occurs in a prototype model. The client evaluates the prototype and gives feedback and other requirements which are incorporated in the next prototype. This is repeated until the prototype becomes a complete product that is acceptable to the client. Some prototypes are built as “throwaways”, others are “evolutionary” in nature as they evolve into the actual system.

Advantages of prototyping

Active involvement of the user.

Errors are detected earlier.

Feedback after each prototype helps in understanding the system better.

Does not need to know detailed processes, input and output from the beginning.

Disadvantages of prototyping

Multiple prototypes can slow down the process.

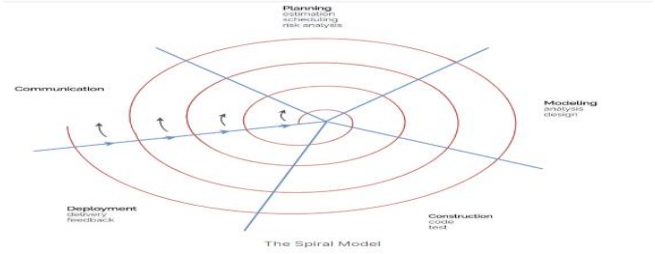
Frequent changes can increase complexity.

Unsatisfied client leads to multiple throwaways.

The customer may not be interested or satisfied after evaluating the initial prototype.

Spiral Model

In spiral model, the software is developed through a series of increments. The diagram looks like a spiral with loops where each loop is a phase. Each phase is split into four sectors/quadrant.



The Spiral Model

The first circuit around the spiral might result in the development of a product

specification. The subsequent passes around the spiral might be used to develop a prototype and then progressively more mature versions of the software.

Planning is where the objectives, alternatives and other constraints are determined. The alternatives are considered, risks in each alternative are analyzed and prototypes are refined in the risk analysis sector. At the development quadrant level risks are known and it proceeds with developing and testing the product. In the assessment sector, customer evaluation of product developed is reviewed and the next phase is planned. This loop continues until acceptable software is built and deployed.

Hence, the spiral model follows an incremental process methodology and unlike other process models, it deals with the uncertainty by applying a series of risk analysis strategies throughout the process.

Advantages of spiral model

- Reduces risk.
- Recommended for complex projects.
- Changes can be incorporated at a later stage.
- Strong documentation helps in better management.

Disadvantages of spiral model

- Costly and not recommended for small projects.
- Demands risk assessment expertise.
- Looping is a complex process.
- Heavy documentation.

Specialized Process Models

Component-Based Development(CBD)

The component Based Development Model has the characteristics of a spiral model, hence is evolutionary and iterative in nature. In this model, applications are built from pre-packaged software components that are available from different vendors.

Components are modular products with well-defined functions that can be incorporated into the project of selection.

The modeling and construction stages are begun with identifying candidate components suitable for the project.

Steps involved in this approach are implemented in an evolutionary fashion:

Components suitable for the application domain are selected.

Component integration issues are catered to.

Software architecture is designed based on the selected components.

Components are integrated into the architecture.

Testing of the functionality of components is done.

Software can be re-used in this manner, cost and time is reduced.

Formal Methods Model (FMM)

In the Formal Methods Model, mathematical methods are applied in the process of developing software. It Uses Formal Specification Language (FSL) to define each system characteristics. FSL defines the syntax, notations for representing system specifications, several objects and relations to define the system in detail.

The careful mathematical analysis that is done in FMM results in a defect-free system. It also helps to identify and correct ambiguity and inconsistency easily. This method is time-consuming and expensive in nature. Also, knowledge of formal methods is necessary for developers and is a challenge.