**UNIT I**

**Product and Process**

**Software Engineering** provides a standard procedure to design and develop a software.

The term **software engineering** is the product of two words, **software**, and **engineering**.

The **software** is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.

Computer programs and related documentation such as requirements, design models and user manuals.

**Engineering** is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain**, and **improve frameworks, processes, etc**.

**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 adequately manipulate information, and

(3) document that describes the operation and use of the programs.

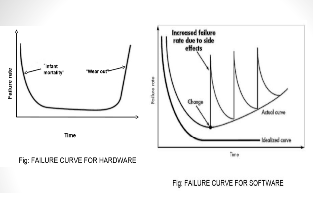
Characteristics of software

➢ 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.

- Although the industry is moving towards component based construction, most software continues to be custom built



**Seven Broad Categories** of software are challenges for software engineers

• **System software**- System software is a collection of programs written to service other programs. System software: such as compilers, editors, file management utilities.

**• Application software**: stand-alone programs for specific needs. This software are used to controls business needs. Ex: Transaction processing.

• **Artificial intelligence software**- Artificial intelligence (AI) software makes use of nonnumeric algorithms to solve complex problems.

Application within this area include robotics, pattern recognition, game playing.

• **Engineering and scientific software**-Engineering and scientific software have been characterized by "number crunching" algorithm.

• **Embedded software** resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)

• **Product-line software** focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)

• **WebApps** (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.

**THE SOFTWARE PROCESS**

• A process is a collection of activities, actions, and tasks that are performed when some work product is to be created. An activity strives to achieve a broad objective with which software engineering is to be applied. An action encompasses a set of tasks that produce a major work .A task focuses on a small,but well-defined objective that produces a tangible outcome.

• A generic process framework for software engineering encompasses **five**

**activities:**

• **Communication**- Before any technical work can commence, it is critically important to communicate and collaborate with the customer (and other stakeholders11 The intent is to understand stakeholders’ objectives for the project and to gather requirements that help define software features and functions.

• **Planning**-Any complicated journey can be simplified if a map exists. The map—called a software project plan—defines the software engineering work by describing the technical tasks to be conducted, the risks that are likely, the resources that will be required, the work products to be produced, and a work schedule.

• **Modeling**-A software engineer creating models to better understand software requirements and the design that will achieve those requirements.

• **Construction**-This activity combines code generation (either manual or automated) and the testing that is required to uncover errors in the code.

• **Deployment**-The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback based on the evaluation.

• **Software engineering** process framework activities are complemented by a number of umbrella activities. Typical umbrella activities include:

• **Software project** tracking and control—allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.

• **Risk management**—assesses risks that may affect the outcome of the project or the quality of the product.

• **Software quality assurance**—defines and conducts the activities required to ensure software quality.

**• Technical reviews**—assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

• **Measurement**—defines and collects process, project, and product measures that assist the team in delivering software that meets stakeholders’ needs; can be used in conjunction with all other framework and umbrella activities.

• **Software configuration management**—manages the effects of change throughout the software process.

• **Reusability management**—defines criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.

• **Work product preparation and production**—encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

**ESSENCE OF SOFTWARE PRACTICE**

• 1. Understand the problem

• 2. Plan a solution

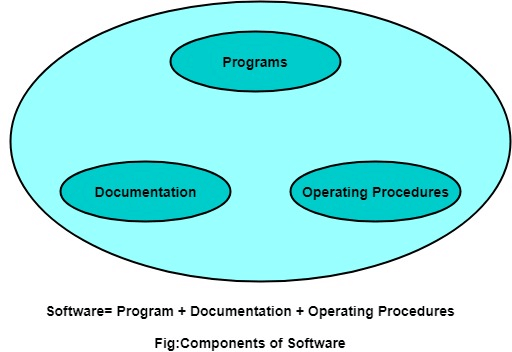
• 3. Carry out the plan

• 4. Examine the result for accuracy

Program vs. Software

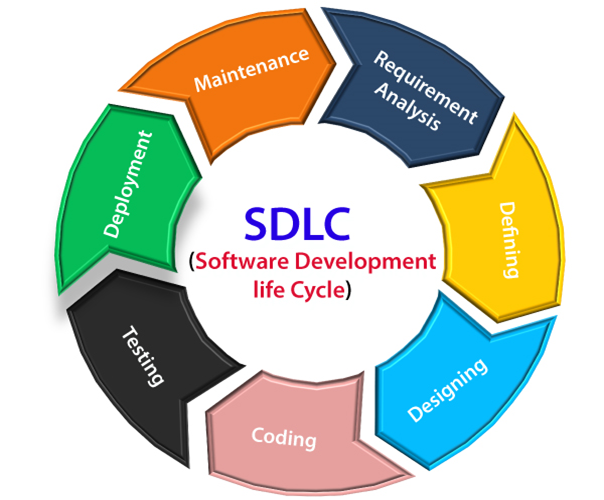
Software is more than programs. Any program is a subset of software, and it becomes software only if documentation & operating procedures manuals are prepared.

There are three components of the software as shown in fig:



SDLC Cycle/Software process Model

SDLC Cycle represents the process of developing software. SDLC framework includes the following steps:



The stages of SDLC are as follows:

**Stage1: Planning and requirement analysis**

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

**For Example**, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

**Stage2: Defining Requirements**

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

**Stage3: Designing the Software**

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gatherin

**Stage4: Developing the project**

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

**Stage5: Testing**

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.

During this stage, unit testing, integration testing, system testing, acceptance testing are done.

**Stage6: Deployment**

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

**Stage7: Maintenance**

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.

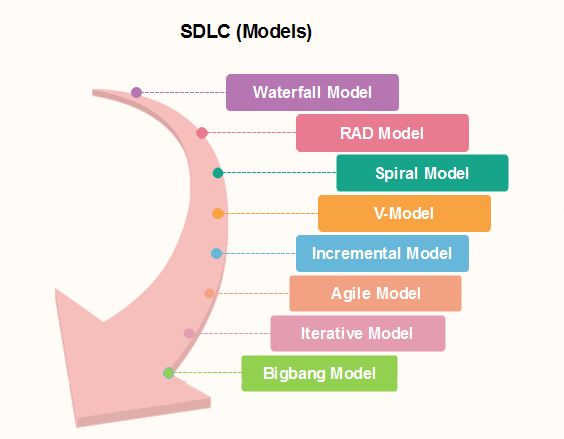
# **SDLC Models**

Software Development life cycle (SDLC) is a spiritual model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.

There are different software development life cycle models specify and design, which are followed during the software development phase. These models are also called "**Software Development Process Models**." Each process model follows a series of phase unique to its type to ensure success in the step of software development.

**Here, are some important phases of SDLC life cycle:**

The waterfall is a universally accepted SDLC model. In this method, the whole process of software development is divided into various phases.ay V



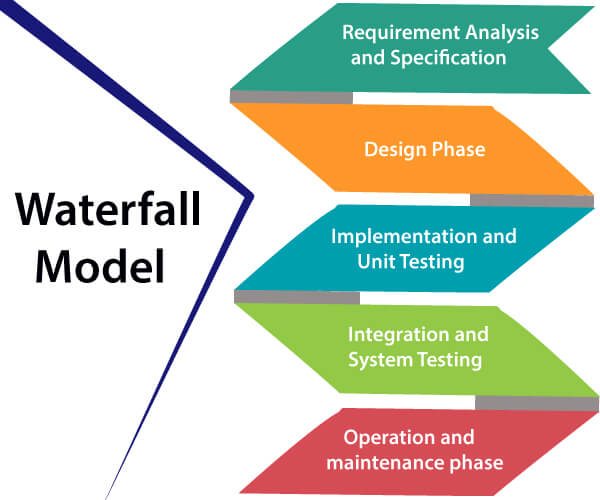
Waterfall Model:

The waterfall is a universally accepted SDLC model. In this method, the whole process of software development is divided into various phases.

The waterfall model is a continuous software development model in which development is seen as flowing steadily downwards (like a waterfall) through the steps of requirements analysis, design, implementation, testing (validation), integration, and maintenance.

Linear ordering of activities has some significant consequences. First, to identify the end of a phase and the beginning of the next, some certification techniques have to be employed at the end of each step. Some verification and validation usually do this mean that will ensure that the output of the stage is consistent with its input (which is the output of the previous step), and that the output of the stage is consistent with the overall requirements of the system.

**1. Requirements analysis and specification phase:** The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how."In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.



**2. Design Phase:** This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

**3. Implementation and unit testing:** During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

**4. Integration and System Testing:** This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

**5. Operation and maintenance phase:** Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

**Advantages of Waterfall model**

* This model is simple to implement also the number of resources that are required for it is minimal.
* The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
* The start and end points for each phase is fixed, which makes it easy to cover progress.
* The release date for the complete product, as well as its final cost, can be determined before development.
* It gives easy to control and clarity for the customer due to a strict reporting system.

**Disadvantages of Waterfall model**

* In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
* This model cannot accept the changes in requirements during development.
* It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
* Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

**RAD or Rapid Application Development process** is an adoption of the waterfall model; it targets developing software in a short period. The RAD model is based on the concept that a better system can be developed in lesser time by using focus groups to gather system requirements.

* Business Modeling
* Data Modeling
* Process Modeling
* Application Generation
* Testing and Turnover

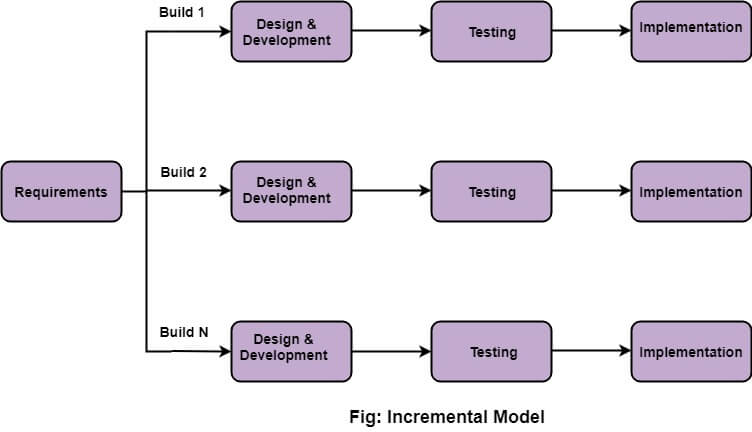
**The spiral model** is a **risk-driven process model**. This SDLC model helps the group to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc. The spiral technique is a combination of rapid prototyping and concurrency in design and development activities.

Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the goals, and the constraints that exist. This is the first quadrant of the cycle (upper-left quadrant).

The next step in the cycle is to evaluate these different alternatives based on the objectives and constraints. The focus of evaluation in this step is based on the risk perception for the project.

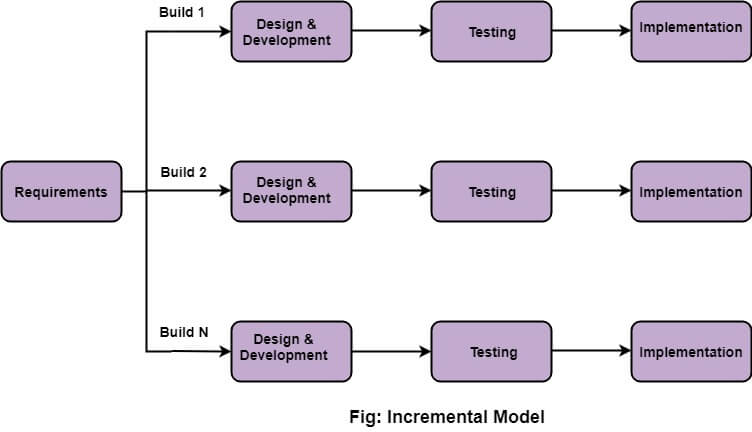
The next step is to develop strategies that solve uncertainties and risks. This step may involve activities such as benchmarking, simulation, and prototyping.

The **incremental model** is not a separate model. It is necessarily a series of waterfall cycles. The requirements are divided into groups at the start of the project. For each group, the SDLC model is followed to develop software. The SDLC process is repeated, with each release adding more functionality until all requirements are met. In this method, each cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.



# Incremental Model

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. The process continues until the complete system achieved.



## The various phases of incremental model are as follows:

**1. Requirement analysis:** In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

**2. Design & Development:** In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

**3. Testing:** In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

**4. Implementation:** Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

**Advantage of Incremental Model**

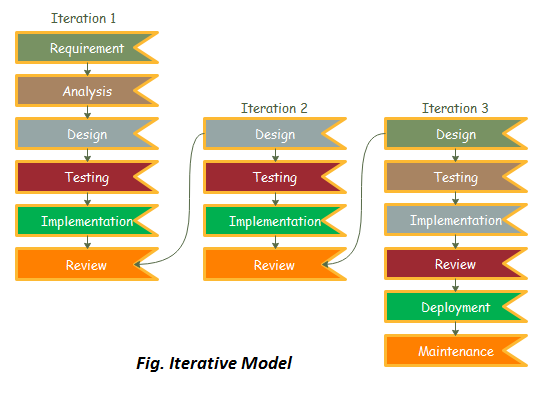
* Errors are easy to be recognized.
* Easier to test and debug
* More flexible.
* Simple to manage risk because it handled during its iteration.
* The Client gets important functionality early.

**Disadvantage of Incremental Model**

* Need for good planning
* Total Cost is high.
* Well defined module interfaces are needed.

**Iterative Model**

It is a particular implementation of a software development life cycle that focuses on an initial, simplified implementation, which then progressively gains more complexity and a broader feature set until the final system is complete. In short, iterative development is a way of breaking down the software development of a large application into smaller pieces.



Evolutionary model is useful in many cases. Two most important cases are mentioned below;

1. It is very **useful in a large project** where you can easily find a module for step by step implementation. The evolutionary model is used when the users need to start using the many features instead of waiting for the complete software.
2. The evolutionary model is also very **useful in object-oriented software development** because all the development is divided into different units

## Advantages of Evolutionary Model

There are many advantages of evolutionary model, Some main advantages are mentioned below;

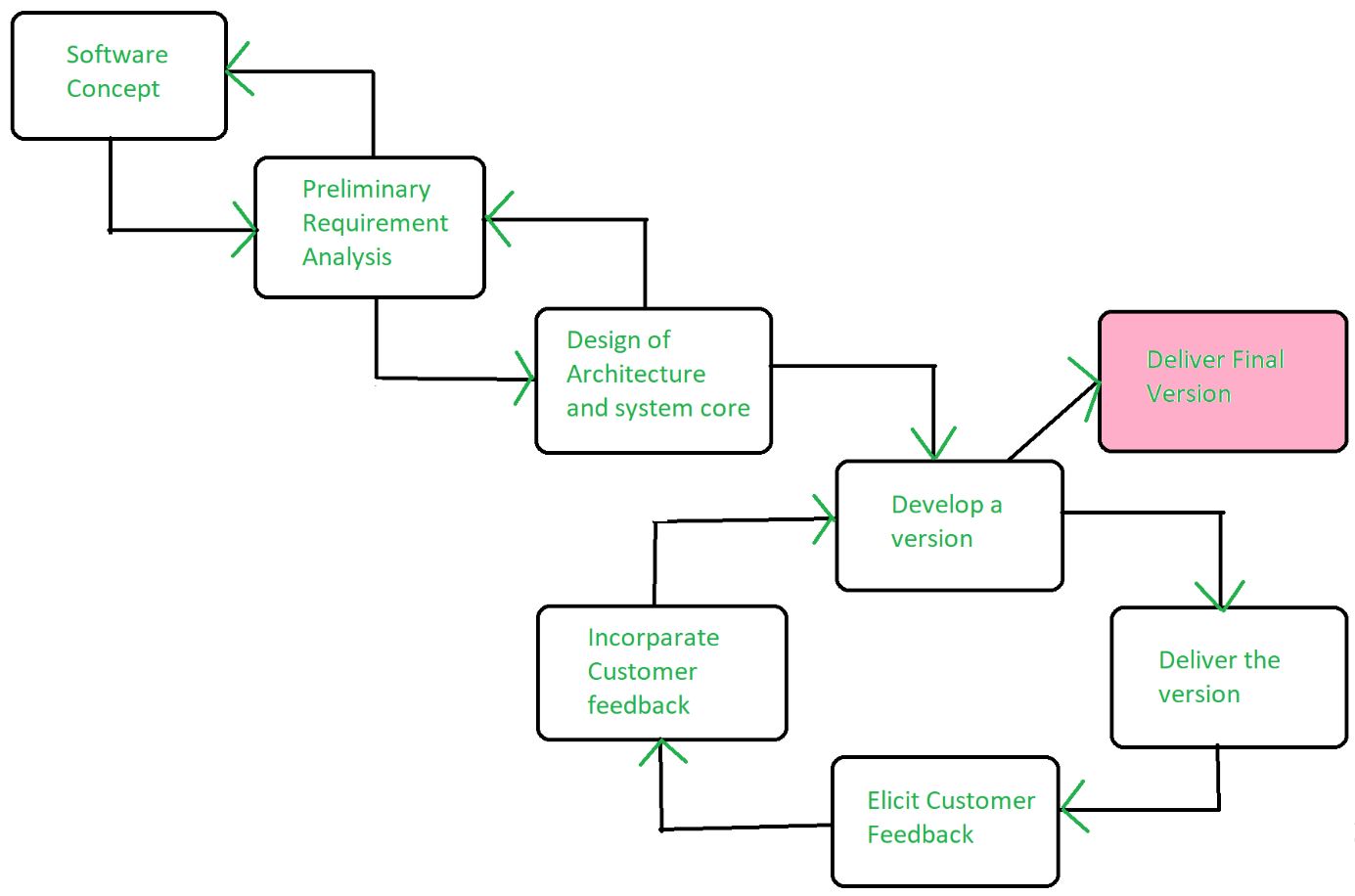
1. The big advantage of the evolutionary model is that t**he user has checked every stage** during the development and it is helpful in achieving customer confidence.
2. There are fewer chances of errors because all the modules are well seen.
3. It helps to**reduce the risk** of software projects.
4. It also **reduces the cost of development**.
5. **Minimize serious problems during testing**

**Disadvantages**

There are many dis-advantages of evolutionary model, Some main advantages are mentioned below;

1. The **delivery of full software can be late** due to different changes by customers during development.
2. It is **difficult to divide the problem into several parts**, that would be acceptable to the customer which can be incrementally implemented and delivered.

**OR**

**Evolutionary model** is a combination of [Iterative](https://www.geeksforgeeks.org/software-engineering-iterative-waterfall-model/) and [Incremental model](https://www.geeksforgeeks.org/software-engineering-incremental-process-model/) of software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done. It is better for software products that have their feature sets redefined during development because of user feedback and other factors. The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle. Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves with time. All the models have the disadvantage that the duration of time from start of the project to the delivery time of a solution is very high. Evolutionary model solves this problem in a different approach.   
   
Evolutionary model suggests breaking down of work into smaller chunks, prioritizing them and then delivering those chunks to the customer one by one. The number of chunks is huge and is the number of deliveries made to the customer. The main advantage is that the customer’s confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements. The model allows for changing requirements as well as all work in broken down into maintainable work chunks. **Application of Evolutionary Model:**

1. It is used in large projects where you can easily find modules for incremental implementation. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. Evolutionary model is also used in object oriented software development because the system can be easily portioned into units in terms of objects.

**Necessary conditions for implementing this model:-**

* Customer needs are clear and been explained in deep to the developer team.
* There might be small changes required in separate parts but not a major change.
* As it requires time, so there must be some time left for the market constraints.
* Risk is high and continuous targets to achieve and report to customer repeatedly.
* It is used when working on a technology is new and requires time to learn.

**Advantages:**

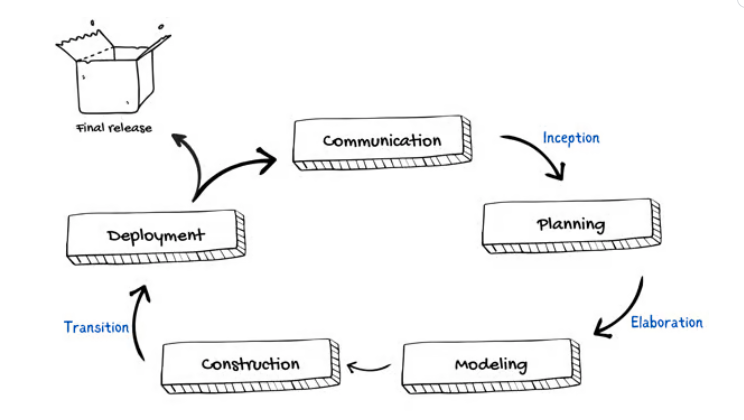
* In evolutionary model, a user gets a chance to experiment partially developed system.
* It reduces the error because the core modules get tested thoroughly.

**Disadvantages:**

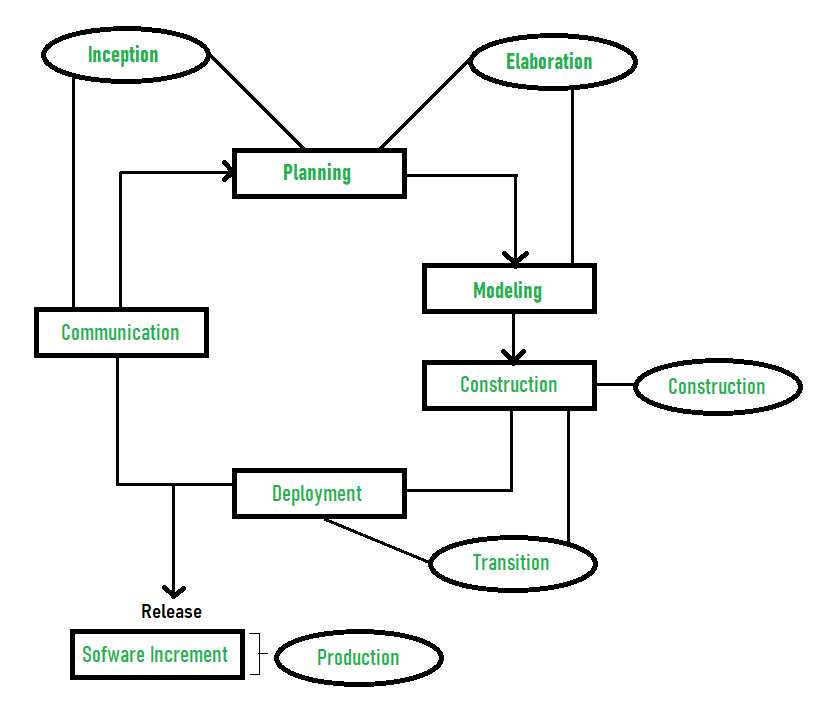
Sometimes it is hard to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented and delivered.

**Unified Process Model**

Unified Process is **based on the enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation**. The system is developed incrementally over time, iteration by iteration, and thus this approach is also known as iterative and incremental software development.



**Rational Unified Process (RUP)** is a software development process for object-oriented models. It is also known as the Unified Process Model. It is created by Rational corporation and is designed and documented using UML (Unified Modeling Language).

**Phases of RUP:** There is total of five phases of the life cycle of RUP: 

1. **Inception –**
   * Communication and planning are the main ones.
   * Identifies the scope of the project using a use-case model allowing managers to estimate costs and time required.
   * Customers’ requirements are identified and then it becomes easy to make a plan for the project.
   * The project plan, Project goal, risks, use-case model, and Project description, are made.
   * The project is checked against the milestone criteria and if it couldn’t pass these criteria then the project can be either canceled or redesigned.
2. **Elaboration –**
   * Planning and modeling are the main ones.
   * A detailed evaluation and development plan is carried out and diminishes the risks.
   * Revise or redefine the use-case model (approx. 80%), business case, and risks.
   * Again, checked against milestone criteria and if it couldn’t pass these criteria then again project can be canceled or redesigned.
   * Executable architecture baseline.
3. **Construction –**
   * The project is developed and completed.
   * System or source code is created and then testing is done.
   * Coding takes place.
4. **Transition –**
   * The final project is released to the public.
   * Transit the project from development into production.
   * Update project documentation.
   * Beta testing is conducted.
   * Defects are removed from the project based on feedback from the public.
5. **Production –**
   * The final phase of the model.
   * The project is maintained and updated accordingly.

**Advantages:**

1. It provides good documentation, it completes the process in itself.
2. It provides risk-management support.
3. It reuses the components, and hence total time duration is less.
4. Good online support is available in the form of tutorials and training.

**Disadvantages:**

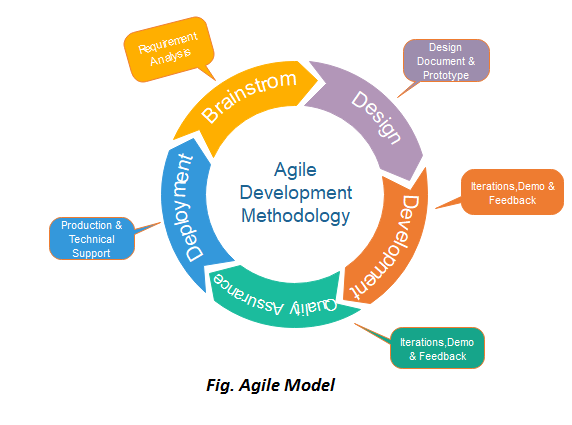
1. Team of expert professional is required, as the process is complex.
2. Complex and not properly organized process.
3. More dependency on risk management.
4. Hard to integrate again and again.

**Agile Development Model**

# **Agile Model**

The meaning of Agile is swift or versatile."**Agile process model**" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.



## Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

**1. Requirements gathering:** In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

**2. Design the requirements:** When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

**3. Construction/ iteration:** When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

**4. Testing:** In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

**5. Deployment:** In this phase, the team issues a product for the user's work environment.

**6. Feedback:** After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Agile Testing Methods:

* Scrum
* Crystal
* Dynamic Software Development Method(DSDM)
* Feature Driven Development(FDD)
* Lean Software Development
* eXtreme Programming(XP)

**Extreme Processing (XP)**

**Extreme Programming − A way to handle the common shortcomings**

Software Engineering involves −

* Creativity
* Learning and improving through trials and errors
* Iterations

Extreme Programming builds on these activities and coding. It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactoring continuously.

Extreme Programming is based on the following values −

* Communication
* Simplicity
* Feedback
* Courage
* Respect

## What is Extreme Programming?

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.

e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where −

* Each practice is simple and self-complete.
* Combination of practices produces more complex and emergent behavior.

### **Embrace Change**

A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.

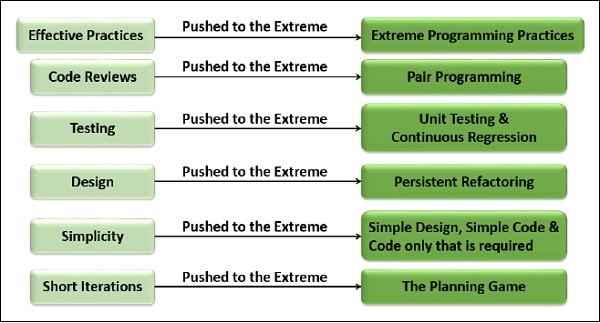
This can be achieved with −

* Emphasis on continuous feedback from the customer
* Short iterations
* Design and redesign
* Coding and testing frequently
* Eliminating defects early, thus reducing costs
* Keeping the customer involved throughout the development
* Delivering working product to the customer

Why is it called “Extreme?”

Extreme Programming takes the effective principles and practices to extreme levels.

* Code reviews are effective as the code is reviewed all the time.
* Testing is effective as there is continuous regression and testing.
* Design is effective as everybody needs to do refactoring daily.
* Integration testing is important as integrate and test several times a day.
* Short iterations are effective as the planning game for release planning and iteration planning.



History of Extreme Programming

Kent Beck, Ward Cunningham and Ron Jeffries formulated extreme Programming in 1999. The other contributors are Robert Martin and Martin Fowler.

In Mid-80s, Kent Beck and Ward Cunningham initiated Pair Programming at Tektronix. In the 80s and 90s, Smalltalk Culture produced Refactoring, Continuous Integration, constant testing, and close customer involvement. This culture was later generalized to the other environments.

In the Early 90s, Core Values were developed within the Patterns Community, Hillside Group. In 1995, Kent summarized these in Smalltalk Best Practices, and in 1996, Ward summarized it in episodes.

In 1996, Kent added unit testing and metaphor at Hewitt. In 1996, Kent had taken the Chrysler C3 project, to which Ron Jeffries was added as a coach. The practices were refined on C3 and published on Wiki.

Scrum practices were incorporated and adapted as the planning game. In 1999, Kent published his book, ‘Extreme Programming Explained’. In the same year, Fowler published his book, Refactoring.

Extreme Programming has been evolving since then, and the evolution continues through today.

Success in Industry

The success of projects, which follow Extreme Programming practices, is due to −

* Rapid development.
* Immediate responsiveness to the customer’s changing requirements.
* Focus on low defect rates.
* System returning constant and consistent value to the customer.
* High customer satisfaction.