| | |
|---|---|
| I | **Software Engineering Models:** Introduction to Software Engineering; Introduction to Software; Types of software; Scope and necessity of Software Engineering; Software Components and Software Characteristics; Software Life Cycle Models: Classical Water Fall Model, Iterative Water Fall Model, Prototype Model, Evolutionary Model, Spiral Model; Comparison of different Life Cycle Models. |

Date :
Page :

# Introduction to Software Engineering.

Software Engineering is an engineering technique branch related to the evolution of software product using well-defined scientific principles techniques, and procedures. The result of software engineering is an effective and reliable software product.

Software engineering is a systematic, disciplined quantifiable study and approach to the design development, operation and maintenance of a software system.

Software: 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.

# Types of Software

1) Application Software: This is end-user programs that help you perform tasks or achieve a desired outcome. Internet browsers, photo-editing software are some examples. Software that performs special functions or provides functions that are much more than the basic operation.

Features:

Performs more specialized tasks like word processing.
Requires more storage software.
More Interactive
Easy to design and understand.
High-level language.

Types

1) General Purpose Software: Used for variety of tasks and not limited to performing a specific task only. Ex - MS-word / MS-Excel.

2) Customized Software: Used to perform specific task or designed for specific organisations Ex - Railway, Invoice management.

3) Utility Software: Used to support the comp. infrastructure. To analyze, configure, optimize and take care of system. Ex - disk repair, antivirus

2) System Software: It directly operates the comp hardware and provides the basic functionality to the users as well as to other software to operate smoothly. Controls a computer's internal functioning and hardware devices as monitor, storage devices etc. It helps user applications to to communicate with hardware.

Features:
Closer to computer system.
Low-level language.
Difficult to design.
Fast in working speed.
Less Interactive.

1) Operating System: The first software that loads into computer's memory. Manages all resources such as memory, CPU, printer, hard disk, etc.
Ex- Linux, Apple macOS, Microsoft Windows etc

2) Language Processor: It converts programs written in high-level programming languages like Java, C++ into set of instructions that are easily readable by machines.

3) Device Driver: A device driver is a program that controls a device and helps that device to perform its functions. It knows how to control or manage that device.

# IMPORTANCE OF SOFTWARE ENGINEERING

1) Reduces Complexity: Big software is always complicated and challenging to progress. It divides big problems into various small issues. And then start solving each small issue one by one. Then solved independently.

2) Minimize Software Cost: Software needs a lot of hardwork and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn the cost for software productions becomes less as compared to any software that does not use this method.

3) To decrease Time: Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. So if you are making your software according to the software engineering method, then it decreases time.

4) Handling Big Projects : To handle a big project without any problem, the company must go for software engineering method.

5) Reliable Software : Software should be secure. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reality. reliability.

6) Effectiveness : Effectiveness comes if anything has made according to the standards. Software standard are the big target of companies to make it more effective. So, software becomes more effective in act with help of software engineering.

## OBJECTIVES / SOFTWARE CHARACTERISTICS

1) Maintainability
2) Efficiency.
3) Correctness.
4) Reusability.
5) Testability.
6) Reliability.
7) Portability.
8) Adaptability.
9) Interoperability.

Date :
Page :

# NEED OF SOFTWARE ENGINEERING

The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.

1) Huge Programming: It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.

2) Adaptability: If the software procedure were not based on scientific and engineering ideas, it would be simpler to recreate new software than to scale an existing one.

3) Cost: As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the paper process not adapted.

4) Quality Management: Better procedure of software development provides a better & quality software product.

Date :
Page :

5) Dynamic Nature: The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one

# SOFTWARE LIFE CYCLE MODELS

There are different software development life cycle models specify and design, which are followed during the software development phase. Models are 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.

1) Waterfall Model:

Universally accepted SDLC model. The whole process of software development is divided into various phases. The waterfall model is a continuous software development model in which developed is seen as flowing steadily downwards through some steps. Winston Royce introduced the model in 1970. The developer must complete every phase before the next phase begins.

The five phases are:

a) Requirement analysis and specification phase:

* Understand exact needs of customer.
* It describes 'what' of the system to be produced.
* A large doc called "Software Requirement Specification" is created, contains detailed description of system working.

b) Design Phase:

* To transform the requirements gathered into suitable form which permits coding.
* Defines overall architecture with high level and detailed design.
* Documented as "Software Design Document".

c) Implementation and unit testing:

* If SDD is complete, the implementation or coding phase proceeds smoothly.

* Small modules are tested in isolation initially. After that these are tested by writing some overhead code to check interaction b/w modules & flows of immediate output.

4) Integration and System Testing :

* Highly crucial as quality is determined by effectiveness of the testing carried out.
* The modules are tested for their interactions with each other and with system.

5) Operation and Maintenance phase :

* Task performed by every user once the software has been delivered to the customer, installed, and operational.

# Spiral Model

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

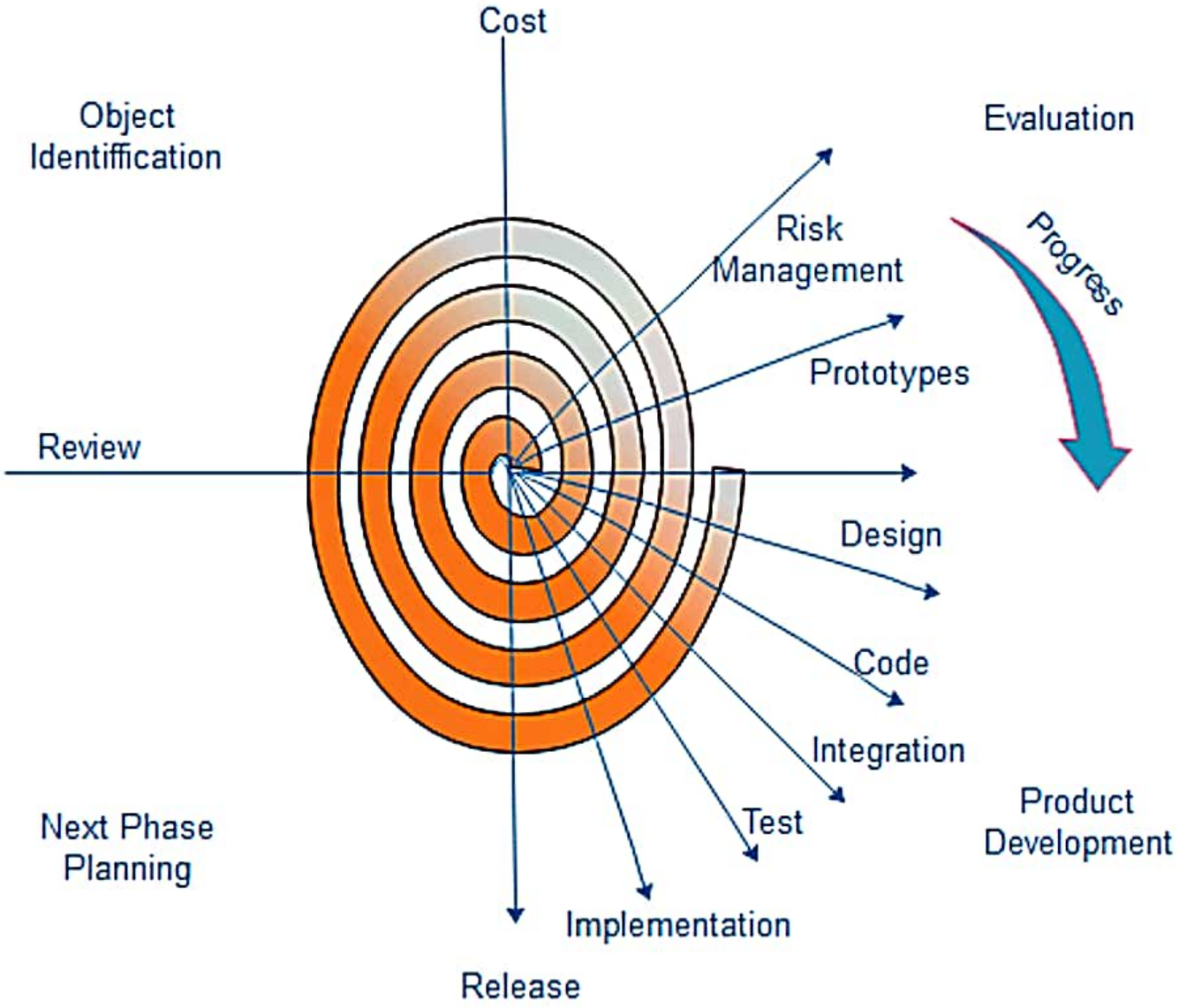



***Fig. Spiral Model***

**Each cycle in the spiral is divided into four parts:**

**Objective setting:** Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

**Risk Assessment and reduction:** The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.

**Development and validation:** The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

**Planning:** Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.

The **risk-driven** feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

## Advantages

- High amount of risk analysis
- Useful for large and mission-critical projects.

## Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects.

In this Model, you can start with some of the software specifications and develop the first version of the software. After the first version if there is a need to change the software, then a new version of the software is created with a new iteration. Every release of the Iterative Model finishes in an exact and fixed period that is called iteration.

The Iterative Model allows the accessing earlier phases, in which the variations made respectively. The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.

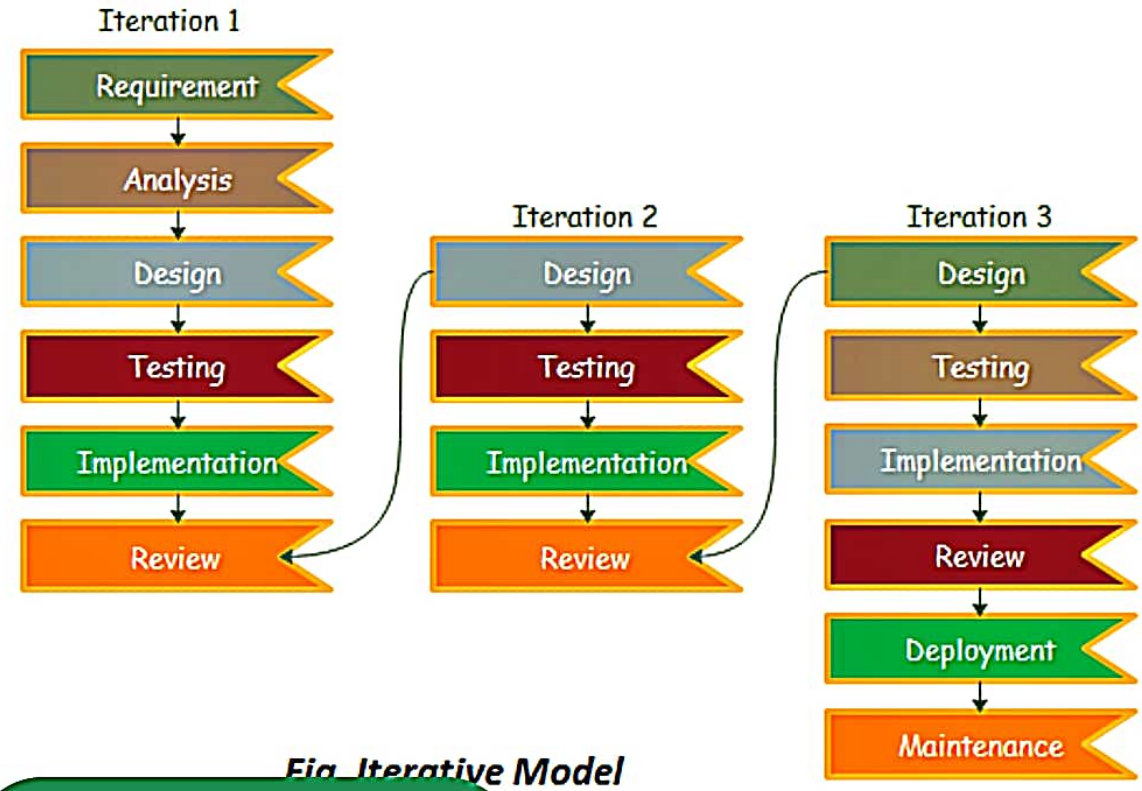



*Fig. Iterative Model*

**1. Requirement gathering & analysis:** In this phase, requirements are gathered from customers and check by an analyst whether requirements will fulfil or not. Analyst checks that need will achieve within budget or not. After all of this, the software team skips to the next phase.

**2. Design:** In the design phase, team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.

**3. Implementation:** In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.

**4. Testing:** After completing the coding phase, software testing starts using different test methods. There are many test methods, but the most common are white box, black box, and grey box test methods.

**5. Deployment:** After completing all the phases, software is deployed to its work environment.

**6. Review:** In this phase, after the product deployment, review phase is performed to check the behaviour and validity of the developed product. And if there are any error found then the process starts again from the requirement gathering.

**7. Maintenance:** In the maintenance phase, after deployment of the software in the working environment there may be some bugs, some errors or new updates are required. Maintenance involves debugging and new addition options.

# Advantage(Pros) of Iterative Model:

1. Testing and debugging during smaller iteration is easy.
2. A Parallel development can plan.
3. It is easily acceptable to ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

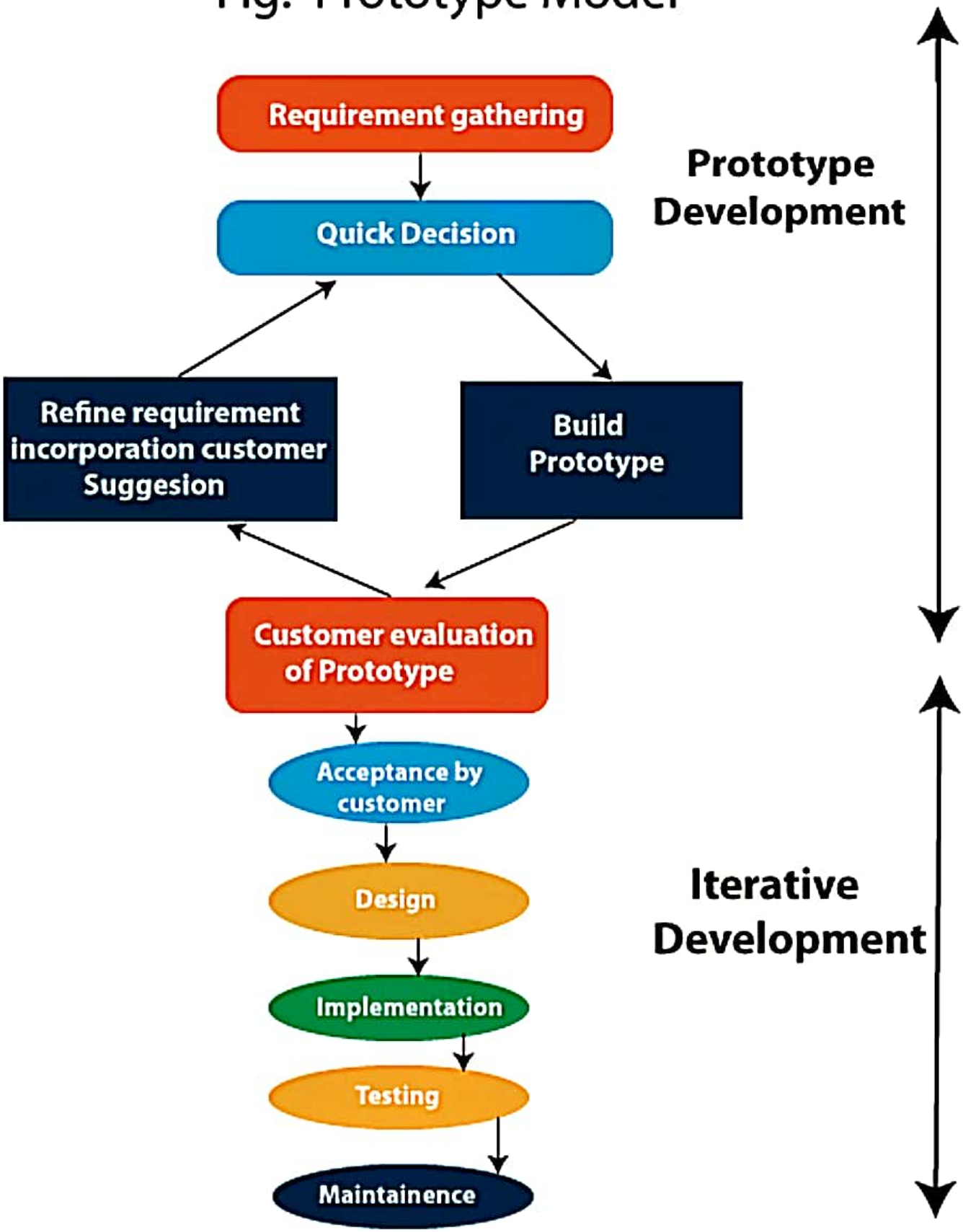
# Disadvantage(Cons) of Iterative Model:

1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

# Prototype Model

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.

Fig: Prototype Model




# Steps of Prototype Model

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

# Advantage of Prototype Model

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
    - Costs customer money
    - Needs committed customer
    - Difficult to finish if customer withdraw
    - May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required

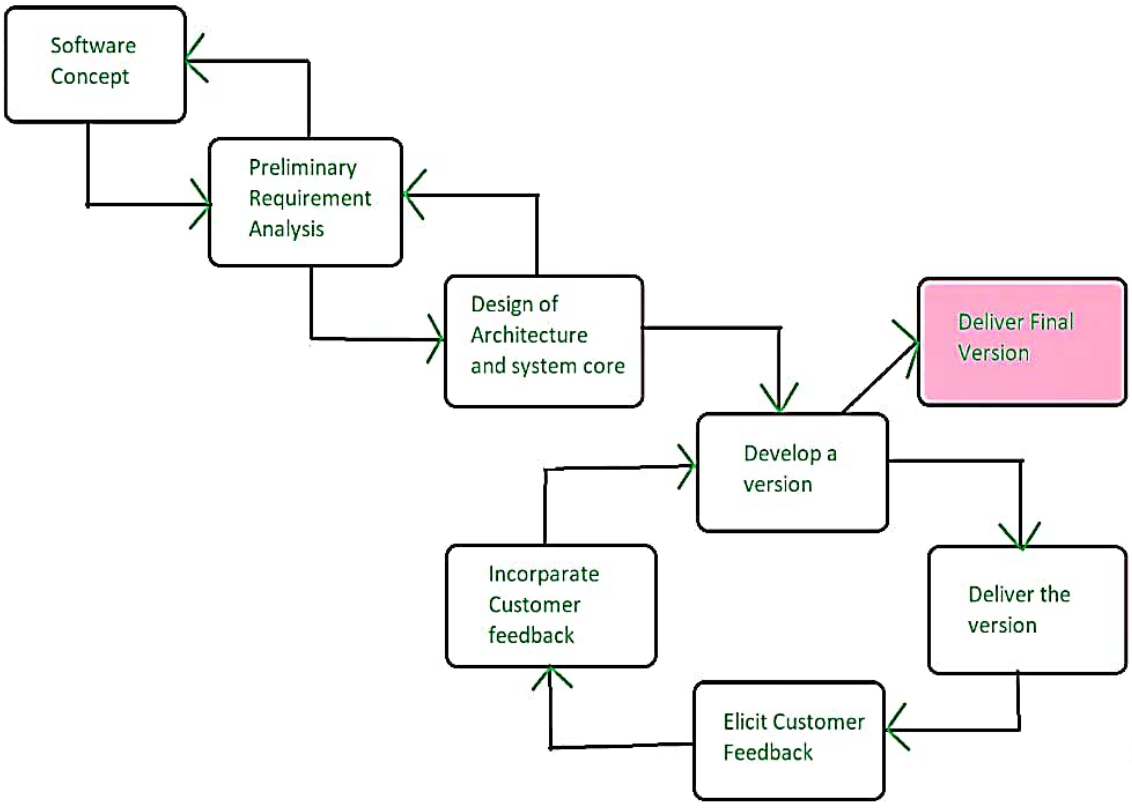
# Software Engineering | Evolutionary Model

**Evolutionary model** is a combination of Iterative and Incremental 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.

**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.

| | |
|---|---|
| II | **Software Requirement Analysis and *Project Planning*:** Requirements Analysis; Feasibility Study: *Software* Requirements Specification (SRS), *Characteristics of SRS*, Components of SRS; IEEE Standards for *SRS*; *Project* Planning; Software Cost Estimation: Basic *COCOMO model*, Intermediate COCOMO model, Complete *COCOMO model*. |

Date :
Page :

**Requirement Inspection**: The inspection func. is concerned with understanding the situation and events that have initiated the software project. This helps one to understand how software is developed is going to be helpful and thereby determine what features the software should possess.

**Identification of stakeholder**: Defined as anyone who benefits in a direct or indirect way from the software system being developed. The usual stakeholders are management marketing people consultants system analysists software engineers and support maintenance engineers of a software firm and buyers and end users of the software. These stakeholders have different needs. The quality of software is determined by the extent to which it satisfies the needs of all its stakeholder. Hence at inspection, the requirement engineer identify different stakeholders.

It is important to identify different categories of stakeholders and examine their varied viewpoints for the sake of convenience we have categorized them into 2 types:

1) One class of stakeholders are those for whom the software is made. These are peoples who's job are effected or improved, who use its software.

2) Other class of stakeholders are - those who work to build and sell this software to the users. They include managers of the software firm, consultant, system analyst, marketing, software engineering and support and maintenance engineer.

**Requirement Elicitation**: Elicitation is about seeking information about the software, the system and the business. The following info. can be sort from customers and the users.
What are the objective of software product?
What is to be accomplished by the software?
How does the software fit into the need of the business?
How is the software to be used on a day to day basis?

The questions based on 5 W's and H and how to fulfil these requirements can be asked to elicit info. about software requirement.
How these questions can be used to elicit user requirement?

**WHO**; To know about organisational unit types of job or client to which the requirement relates.
**WHAT**: To know about task that need to be performed.

Date:
Page:

WHERE: To know about type of machine (client server etc) on which the processing is required to be performed.
WHEN:

TOOLS

1) Through Interview: for elicitation of users requirement both qualitative and quantitative form of information. Interview are often the best mean to elicit quantitative info- such as opinions, policies and narrative descriptions of activities and problem. Many people who are not able to express ~~well~~ themself well in writing feel free to discuss idea freely. Often easier to schedule an interview with senior managers than to have them fill up questionnaires. The analyst can interview people one at a time or in groups. Depending on requirement, it can be structured or unstructured.
Structured Interviews are ~~not~~ formal and questions to be asked are generally framed in advance. Ensures info. wording of questions for all respondent

Unstructured Interviews uses free questions The open and free almosphere provides greater opportunities to learn about feelings ideas and beliefs of respondent.

2) Through Questionnaires: Due to time constraint only a limited of persons can be interviewed. A questionnaire is a convenient way to contact a larger no. of people to get their views about various aspect of system. When the system is large or the scope of study covers several departments, Questionnaire can be distributed to all appropriate persons inviting them to furnish necessary facts about the system.

3) Through Record Review: A good volume of records are often available in many organisation which provides useful info. about existing system. The term 'record' refers to the written policy manuals and standard processing procedures that most organisation maintain as a guide for manager and employees. Manuals are doc. that describe existing procedures and operations of organisation.

4) Through On-Site Observation: Observing people in the act of executing their jobs is an effective technique of gathering facts about a system. Observation as a fact finding techniques is widely accepted by psychologist, sociologist, and industrial engineers for studying various activity being perform in an org.

Date :
Page :

# SOFTWARE REQUIREMENT SPECIFICATION

It accomodates life cycle evolution from a growth and requirement change.
It incorporates software quality objective into the product. It focuses on early errors deduction.

## Requirement Engineering Process.

The first step in it is to gain an understanding of the problem for which the software is being developed for - this all necessary info about the problem are gathered. Based on the info the initial requirement specification of the software is developed. The initial specification is further refined after discussion with the perspective user and stake holder

## Types of Software Requirement

1) **Design Requirement**: It specifies the design of the system or system component

2) **Functional Requirement**: It specifies the function that a system or system component must perform.

3) Implementation Requirement: It specifies the coding or construction of a system or system component. This affected the technical effectiveness.

4) Interface Requirement: It specify the external items with which the system or its component must in transit and it is and the construction on format, timing or other factors calls by such interaction.

5) Performance Requirement: It imposes conditions on a function requirement for e.g. a requirement that specify the speed, accuracy or memory uses with which a given function must be perform.

6) Physical Requirement: It specify a physical characteristics that a system or system component must be developed.

# Structure of SRS (Software Requirement Specification)

SRS is a final work product produced by the requirement engineer. It serves as the foundation for the subsequent foundation software engineering activities. It discribes the function and performance requirement of the software. It also lists the constraint that will affect its development.

SRS is a formal document. It uses natural language, graphical representations, mathematical models, scenarios, prototype model or any combination of these to discribe the software to be developed. There are standard templates for presenting requirement specification in a consistent and more understandable manner.

The characteristics of good SRS documents are as follows:

a) Structure : Should be well structured. A well structured document is easy to understand and modify.

b) Concise : Should be concised consistent & complete

c) Conceptual Integrity : Should have conceptual integrity so that it can be easily understood by reader without any ambiguity.

d) Viewed as black box : should only specify the functions that the software is required to perform and not how it should do the required functions. Hence, the SRS document views the software as blackbox. Also called the black box specification of a system.

e) Response to Exceptional conditions : Should list all conceivable exceptional conditions or events and specify software response to such conditions.

f) Verifiable : All requirement of the software as document should be verifiable. It should be possible to determine whether the requirements stated in the SRS document have been met in implementation or not.

Date :
Page :

# Software Metrics

Metric is quantity measure of the degree to which a given attribute is processed by a system or its component or by a process. Software metric are measures that are used to quantify different attributes of software program.
Software metric estimation and measure of different attribute of software and software develop. process.
Software metrices can be classified into 3 types:

1. Product- Metrics
2. Process Metrics
3. Resource Metrics

1. The measures of different characteristics of software product. The two important characteristics are:
   a) Size & Complexity
   b) Quality & Reliability

2. The measures of different characteristics of software development process. They quantify different aspect of the process being used to develop the software.

Date :
Page :

3. Quantify measures of various resources used in software project. Software projects uses 3 major types of resources
Human Resource
Hardware Resource
Software Resource
It denotes how efficiently the available resources are in used in project.

## Software Size Metrics

Most of initial work in product metrics is dealt with characteristics of source code. The most important is the size of software.
A no. of different software size metrics have been proposed & used. Size is typically a direct count of selected characteristics to describe volume of software product. Can be expressed in 3 following ways.

a) In Terms of Physical Size of Program.
b) In Terms of meaningful functions/services it provides.
c) In Terms of Logical size

Date :
Page :

a) Lines of Code Metrics : Traditionally the LOC or KLOC is the primary measure of software size. Most lines of code metrics count all executable instructions and data declarations but exclude comments, blanks & continuation lines. Estimate size through analogy by comparing new software to similar functions found in other existing application.
Having more detailed info about functionality of new software clearly provides basics for better comparison

## Function Point Metrics

Was developed in IBM in the year 1977
func. point of application are determined by counting number a types of function used in the application. Various functions used in application can be categorized into five types

| Category | Property |
|---|---|
| 1. External Input | All unique data of control input that cross the system boundary and cause processing to occur. |
| 2. External Output | All unique data of control output that cross the system boundary after the processing has occurred. |

Date :
Page :

3. External Inquiry All unique transaction that cross the system boundary to make active demand on system.

4. Internal File All logical of data that are stored within a system according to some predefined conceptual schema

5. External Interface All unique files or program that cross system boundary & are shared with atleast one other system or application

Date :
Page :

# Classification of S/W Projects :

The requirement of cost effort and time for s/w projects depends on their development complexity. Boehm classified s/w development projects in 3 categories based on the development complexity. These are as follows

**Organic S/W:** A development project can be considered of type organic if the project deals with developing a well understood app program; the size of the development team is reasonably small and team members are experienced in developing similar type of projects. The complexity involved in organic s/w are low.

Embedded s/w :- The complexity involved in the E.S projects are high. A development project is considered to be of embedded type if the s/w being developed is strongly coupled to complex h/w.

Semi-detached :- the complexity involved in the

semi- " s/w projects lie in b/w O.S and e.s

A s/w project maybe considered of semi-detached type if some of the staff involved in the dev. are exp. and some are in-exp. The team members may have limited exp. on related sys. but may not be injamiliar with all ~~of the~~ aspects of the sys.

## Constructive or Estimation Model

Was proposed by Boehm in 1981 and as the model for estimating effort, cost, and schedule for s/w projects.

The estimation of s/w projects are done in 3 stages / models:

1. **Basic COCOMO**

Gives an approximate estimate of effort for software project- based on a single attribute and the size of the software expressed in terms of KLOC is the most important attribute for estimation of effort. According to BASIC COCOMO, the expressions of estimation of efforts required for s/w development is

$$Effort = A \times (Size)^B$$

where A & B are constants.

Effort estimation is expressed in units of PMs. The amount of effort estimated for s/w development is main

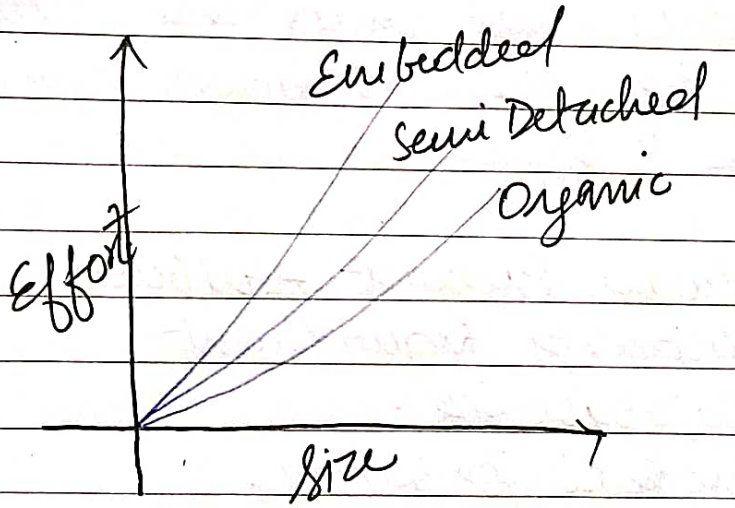
Date :
Page :

parameter for estimations of time.
Expressions for estimation of time required for S/W development is;

$$T = C \times (Effort)^D$$

where C & D are constants.

| Types of Software | A | B | C | D |
|---|---|---|---|---|
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semi detached | 3.0 | 1.12 | 2.5 | 0.38 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.38 |




Q- The size of the organic type is 50 KLOC, If unit cost of effort is Rs 20,000 per PM Estimate development time & cost of project

$$Effort = 2.4 \times (50)^{1.05}$$
$$= 145.925014879$$
$$Time = 2.5 \times (145.925014879)^{0.38}$$
$$= 16.6076931576 \text{ M}$$

Cost = Rs 2,918,500.29758

Date :
Page :

## 2. Intermediate COCOMO

The purpose of basic COCOMO is to get rough estimate of effort and time required for software development. It considers only one attribute of software, i.e., software size expressed in KLOC. However, beside the software size, there are various other factors that affect the amount of effort & time duration needed for completion of software project.

The intermediate COCOMO recognizes 15 factors / cost drivers that affect the software project. These 15 factors can be categorized into 4 types

### 1. Software Product Attributes

1.1 Reliability Requirement
1.2 Database Size.
1.3 Product Complexity.

### 2. Computer System Attributes

2.1 Execution Time Constraints.
2.2 Main Storage Constraints.
2.3 Virtual Machine Volatility.
2.4 Computer Turn around Time.

### 3. Human Resource Attribute

3.1 Analyst Capability
3.2 Virtual Machine Experience
3.3 Programmer capability

Date :
Page :

3.4 Programming Language Experience
3.5 Application Experience.

4. Software Product - Attributes.
4.1 Use of Modern Practices
4.2 Use of Software tools.
4.3 Required Development Schedule.

3. Complete COCOMO
Any large software system will consist of different components & sub systems. The complexity of these components may not be the same. For example, some subsystem may be too simple whereas some may be quite complex. Hence, some of the sub systems of large system may be considered as organic, as semi-detached, as embedded type.
Basic and Intermediate COCOMO are considered as software project as single homogeneous entity, however, complete COCOMO considers system to be heterogeneous in nature to apply complete COCOMO. The software is broken down into subsystem. The sub system are categorized into organic, semi-detached or embedded type. The sub systems are estimated separately and their added up to get the estimates of complete project.