NARAYANA ENGINEERING COLLEGE:: NELLORE
SOFTWARE ENGINEERING

MODULE– 1	II B.Tech, II Semester, CSE-C	2024-25
Introduction: Evolution, Software development projects, Exploratory style of software developments, Emergence of software engineering, Notable changes in software development practices, Computer system engineering. **Software Life Cycle Models:** Basic concepts, Waterfall model and its extensions, Rapid application development, Agile development model, Spiral model.		

Software Engineering - Introduction

- **Software** is a set of instructions used to get inputs and process them to produce the desired output. It also includes a set of documents, such as the software manual, Source Code, Executables, Design Documents, Operations, and System Manuals and Installation and Implementation Manuals.
- **Engineering** is the application of scientific and practical knowledge to invent, design, build, maintain, and improve frameworks, processes, etc

Software products may be:
[1] Generic - That means developed to be sold to a range of different customers.
[2] Custom - That means developed for a single customer according to their specification.

What is software engineering?

A popular definition of software engineering is: "A systematic collection of good program development practices and techniques".
An alternative definition of software engineering is: "An *Engineering approach* to develop software".

A few important definitions given by several authors and institutions are as follows:
Definition:
Software Engineering is the application of a systematic, disciplined, scientific approach to the development, operation and maintenance of software.

OR

Software Engineering is a discipline whose aim is the production of fault free software that satisfies the user's needs and that is delivered on time and within budget.

Programs

- A program is an executable code, which serves some computational purpose.
- A program is a combination of source code and object code.
- A program is a subset of software and it becomes software only if documentation and operating procedure manuals are prepared.
- **Program = Source Code + Object Code**

Software Products

- Software, when made for a specific requirement is called a software product. The outcome of software engineering is an efficient and reliable software product.

- A software product consists not only of the program code but also of all the associated documents, such as the requirements specification documents, the design documents, the test documents, and the operating procedures, user manuals and operational manuals, etc.
- **Software=Program + Documentation + Operating Procedures**

Characteristics of Software:

- Software is developed or engineered; it is not manufactured in the classical sense.
- Software does not "wear out"
- Although the industry is moving toward component-based construction, most software continues to be custom built.
- Reusability of components. (How easily and effectively users can interact with and navigate through the software.)
- Flexibility of software.
- Maintainability of software.(How easily and cost-effectively software can be modified, updated, or extended.)
- Portability of software. (The ability of software to run on different platforms or environments without requiring significant modifications.)
- Reliability of Software.(The ability of the software to consistently perform its intended tasks without unexpected failures or errors.)

Is software engineering a science or an art?

- Writing good quality programs is an art.
- Past experiences have been systematically organized and wherever possible theoretical basis to the empirical observations have been provided.
- An appropriate solution is chosen out of the candidate solutions based on various trade- offs that need to be made on account of issues of cost, maintainability, and usability.
- Engineering disciplines such as software engineering make use of only well-understood and well-documented principles.

1.1 EVOLUTION—FROM AN ART FORM TO AN ENGINEERING DISCIPLINE

- Early programmers used an *ad hoc* programming style. This style of program development is now variously being referred to as *exploratory*, *build and fix*, and *code and fixes* styles.
- The exploratory programming style is an informal style in the sense that there are no set rules or recommendations that a programmer has to adhere to—every programmer himself evolves his own software development techniques.

a) Evolution of an Art into an Engineering Discipline

- ➢ Software engineering principles have evolved over the last sixty years with contributions from numerous researchers and software professionals.
- ➢ Over the years, it has emerged from a pure art to a craft, and finally to an engineering discipline.
- ➢ The early programmers used an ad hoc programming style.
- ➢ This style of program development is now variously being referred to as exploratory, build

and fix, and code and fixes styles.

- In a build and fix style, a program is quickly developed without making any specification, plan, or design.
- The exploratory programming style is an informal style in the sense that there are no set rules or recommendations that a programmer has to adhere to every programmer himself evolves his own software development techniques solely guided by his own intuition, experience, whims, and fancies.
- The exploratory style comes naturally to all first time programmers.

b) Evolution Pattern for Engineering Disciplines

- The evolution of the software development styles over the last sixty years, tells that it has evolved from an esoteric art form to a craft form, and then has slowly emerged as an engineering discipline.
- Every technology in the initial years starts as a form of art.
- Over time, it graduates to a craft and finally emerges as an engineering discipline.
- Those who knew iron making kept it a closely-guarded secret.
- This esoteric knowledge got transferred from generation to generation as a family secret.
- Slowly, over time technology graduated from an art to a craft form where tradesmen shared their knowledge with their apprentices and the knowledge pool continued to grow.
- In the early days of programming, there were good programmers and bad programmers.
- The good programmers knew certain principles (or tricks) that helped them write good programs, which they did not share with the bad programmers.
- Over the next several years, all good principles were organized into a body of knowledge that forms the discipline of software engineering.

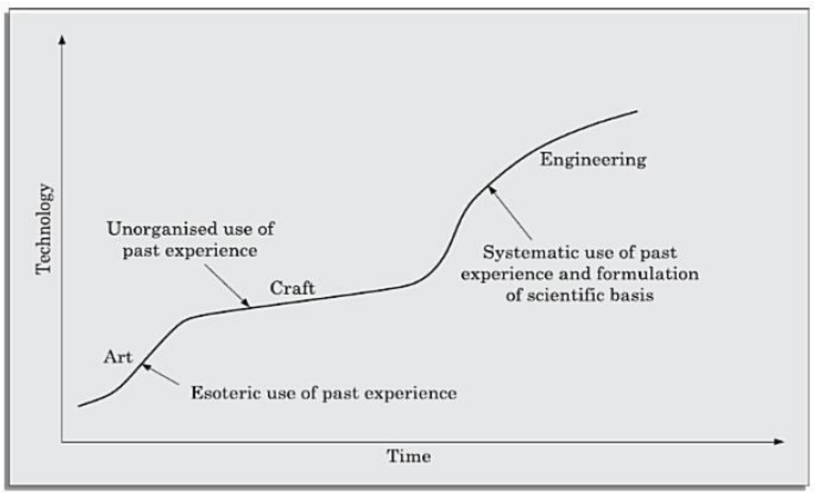

Technology
Engineering
Unorganised use of past experience
Craft
Systematic use of past experience and formulation of scientific basis
Art
Esoteric use of past experience
Time

Figure 1.1: Evolution of technology with time.

c) A Solution to the Software Crisis

- Software engineering is one option that is available to tackle the present software crisis.
- The expenses that organizations all over the world are incurring on software purchases as compared to the expenses incurred on hardware purchases have been showing an worrying trend over the years
- The trend of increasing software costs is probably the most vexing.
- Hardware Prices would become insignificant compared to software prices—when you buy any software product the hardware on which the software runs would come free with the software!!!

- Factors that contribute to the present software crisis are
    - ✓ Fail to meet user requirements.
    - ✓ Frequently crash.
    - ✓ Expensive.
    - ✓ Difficult to alter, debug, and enhance.
    - ✓ Often delivered late.
    - ✓ Use resources non-optimally.
    - ✓ Lack of adequate training in software engineering,
    - ✓ Increasing skill shortage,
    - ✓ Low productivity improvements.

What is the remedy?

It is believed that a satisfactory solution to the present software crisis can possibly come from a spread of software engineering practices among the developers, along with the further advancements. . Organizations are spending increasingly larger portions of their budget on software as compared to that on hardware.

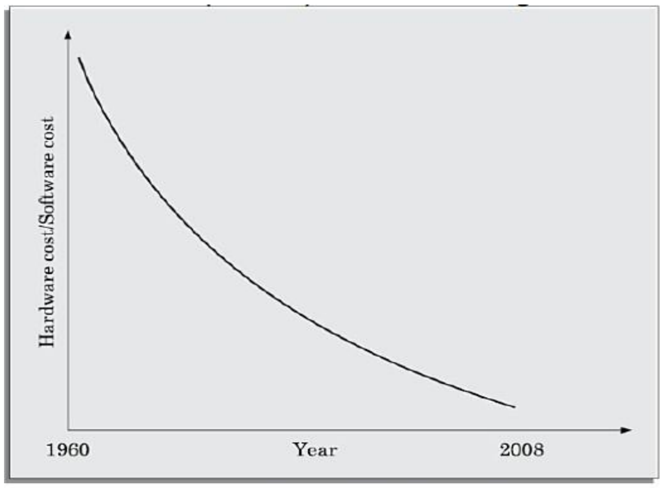


Figure 1.2: Relative changes of hardware and software costs over time.

1.2 SOFTWARE DEVELOPMENT PROJECTS

A. <u>Programs *versus* Products</u>

- Many **toy** software are developed by individuals such as students for their classroom assignments and for their personal use. These are usually small in size and support limited functionalities. The author of a program is usually the sole user of the software and himself maintains the code. These toy software lack good user-interface and proper documentation. It has poor maintainability, efficiency, and reliability. Since these toy software do not have any supporting documents such as users' manual, maintenance manual, design document, test documents, etc., we call these toy software as ***programs***.

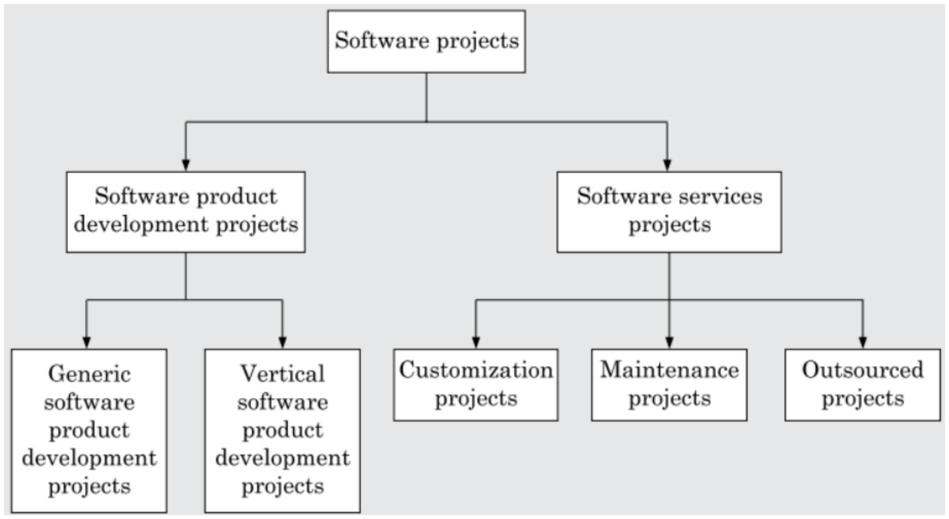
- In contrast, professional software usually has multiple users and, therefore, has good user-interface, proper users' manuals, and good documentation support. It is systematically designed, carefully implemented, and thoroughly tested. In addition, professionally written software usually consists not only of the program code but also of all associated documents such as requirements specification document, design document, test document, user manuals, etc. other difference is that professional software are often too large and complex to be developed by any single individual. It is usually developed by a group of developers working in a team.

➢ Professional software is developed by a group of software developers working together in a team.

Program	Software product
Programs are developed by individuals for their personal use.	A software product is usually developed by a group of engineers working as a team.
Usually small in size	Usually large in size
Author himself is sole user i.e Single User	Large number of users
Single developer	Team of developers
Lacks proper user interface	Well-designed interface
Lacks proper documentation	Well documented & user-manual prepared
Ad hoc development.	Systematic development

B. <u>Types of Software Development Projects</u>

A software development company typically has a large number of on-going projects. Each of these projects may be classified into software product development projects or services type of projects.


Software projects
Software product development projects
Software services projects
Generic software product development projects
Vertical software product development projects
Customization projects
Maintenance projects
Outsourced projects

I. Software products

- Microsoft's Windows and the Office suite, Oracle DBMS, software accompanying a camcorder or a laser printer, etc. This software are available off-the-shelf for purchase and are used by a diverse range of customers. These are called ***generic software products.***

II. Software services

- Customized ***software*** is developed according to the specification drawn up by one or at most a few customers.
- Another type of software service is ***outsourced software***. Sometimes, it can make good commercial sense for a company developing a large project to outsource some parts of its development work to other companies.

C. <u>Software Projects Being Undertaken by Indian Companies</u>

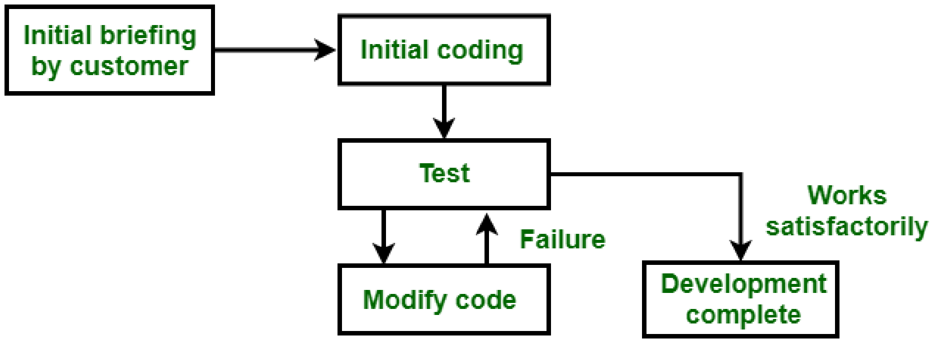
- Indian software companies have excelled in executing software services projects and have made a name for themselves all over the world. Of late, the Indian companies have slowly started to focus on product development as well.

- A company needs to invest upfront and there is substantial risks concerning whether the investments would turn profitable. Possibly, the Indian companies were risk averse.

1.3. EXPLORATORY STYLE OF SOFTWARE DEVELOPMENT

- **Exploratory program development style** refers to an **informal development style** or **builds and fixes the style** in which the programmer uses his own intuition to develop a program rather than making use of the systematic body of knowledge which is categorized under the software engineering discipline.
- This style of development gives complete freedom to programmers to choose activities which they like to develop software. This dirty program is quickly developed and bugs are fixed whenever it arises.
- This style does not offer any rules to start developing any software.

The following block diagram will clear some facts relating to this model :

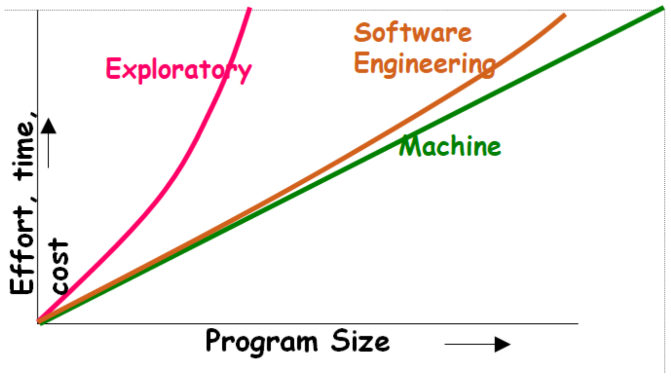


Exploratory program development

In the above diagram, the first block is the **initial briefing by the customer** i.e brief introduction of the problem by the customer. After the briefing, control goes to **initial coding** i.e. as soon as the developer or programmer knew about the problem he starts coding to develop a working program without considering any kind of requirement analysis. After this, the program will be **tested** i.e bugs found and they are getting fixed by programmers. This cycle continues until satisfactory code is not obtained. After finding satisfactory code, development gets completed.

What is wrong with the Exploratory Style?

- Can successfully be used for very small programs only.
- Does not work for Non-trivial projects.



- Exploratory style usually results in un maintainable code. It becomes very difficult to use the exploratory style in Team development environment.
- The approach completely breaks down when the size of the software become large.

An Interpretation Based on Human Cognition Mechanism.

- Human memory can be thought to be made up of two distinct parts.

-- Short term memory

-- Long term memory

Suppose I ask: "It is 10:10AM now,
how many hours are remaining today?"
Human Cognition Mechanism
10AM would be stored in the short-term memory.
"A day is 24 hours long." would be fetched from the long term memory into short term memory.
The mental manipulation unit would compute the difference (24-10).

Principles Deployed by Software Engineering to Overcome Human Cognitive Limitations..

Software engineering principles use **abstraction** and **decomposition** to reduce problem complexity and overcome human cognitive limitations:

<u>Abstraction</u>: Separates the behavior of software components from their implementation. This requires looking at software from two points of view: what it does and how it does it. Also called Model building.

- Simplifying a problem by omitting unnecessary details

- focus attention on only one aspect of the problem and
ignore other aspects and irrelevant details

- for more complex problems

- a single level abstraction is inadequate

- a hierarchy of abstraction may have to be constructed.

<u>Decomposition</u>: Breaks down complex problems into simpler independent parts.

-- the smaller parts are then taken up one by one and solved separately.

-- the idea is that each small part would be easy to grasp and therefore can be easily solved.

Ex: try to break a bunch of sticks tied together versus breaking them individually.

1.4 EMERGENCE OF SOFTWARE ENGINEERING

Software engineering techniques evolution is the result of a series of innovations and accumulation of experience about writing good quality programs. They are

a) **Early Computer Programming**

b) **High-level Language Programming**

c) **Control Flow-based Design**

d) **Data Structure-oriented Design**

e) **Data Flow-oriented Design**

f) **Object-oriented Design**

g) **Web based Design**

a) **<u>Early Computer Programming(1950s)</u>**

- Every programmer developed his own individualistic style of writing programs according to his intuition and used this style ***adhoc*** while writing different programs.
- Then designated this style of programming as the *build and fix* (or the *exploratory programming*) style.
- Early commercial computers were slow and elementary.
- It took lot of time for computation.
- Programs were very small in size and were written in assembly language.
- Programmers wrote it without proper plan, design etc.

b) <u>High-level Language Programming(early 1960s)</u>

- Computers became faster with the introduction of this semiconductor technologies.
- This helped to solve more complex problems.
- At this time, high level language BASIC, FORTRAN, COBOL, ALGOL were introduced.
  - This reduced software development efforts greatly.
- Software development style was still exploratory.
  - Typical program sizes were limited to a few thousands of lines of source code.

c) <u>Control Flow-based Design</u>

- Size and complexity of programs increased further:
  - exploratory programming style proved to be insufficient.
- Programmers found:
  - Very difficult to write cost
  - effective and correct programs.
  - They also found it difficult to understand and maintain program written by others.
  - So they started to pay attention to program's control flow structure.
  - Thus flow charting technique was developed.
- Using flow charting technique:
  - one can represent and design a program's control structure.
- Usually one understands a program:
  - by mentally simulating the program's execution sequence.
- Many programmers had extensively used assembly languages.

  →JUMP instructions are frequently used for program branching in assembly languages,

  →programmers considered use of GO TO statements inevitable.

- If the flowchart representation is simple, then the corresponding code should be simple.

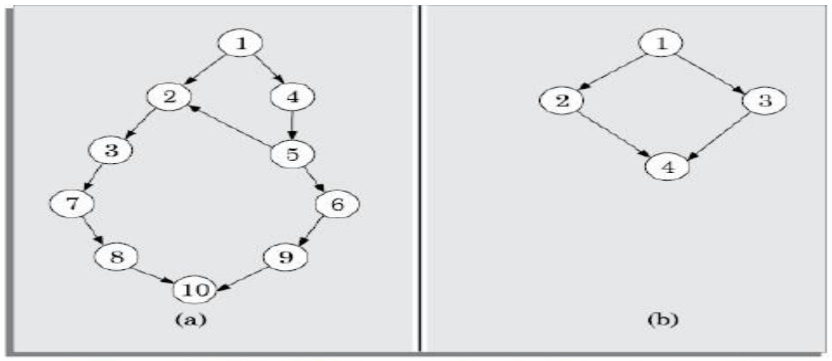


Figure 1.9: Control flow graphs of the programs of Figures 1.8(a) and (b).

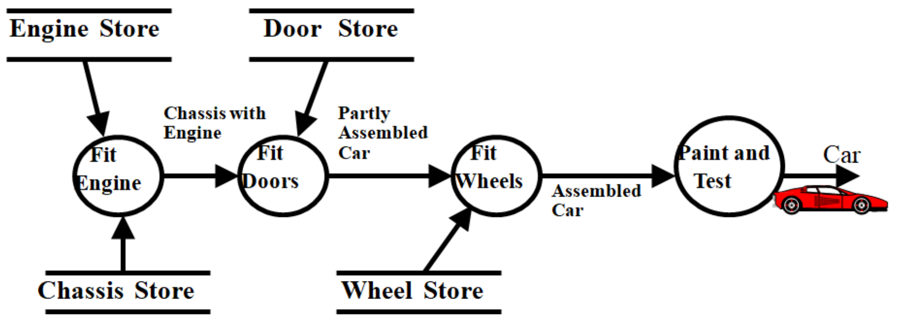
- For example, for the program of Fig 1.9(a) you would have to understand the execution of the program along the paths1-2-3-7-8-10,1-4-5-6-9-10,and1-4-5-2-3-7-8-10. A program having a messy control flow(i.e. flow chart) structure, would have a large number of execution paths. Consequently, It would become extremely difficult to determine all the execution paths.
- A program having a **messy flow chart** representation: difficult to understand and debug.
- **Structured programming**
  – A program is called structured when it uses only the sequence, selection, and iteration types of constructs and is modular.
  – Very soon several languages such as PASCAL, MODULA, C, etc., became available which were specifically designed to support structured programming.

d) **<u>Data Structure-oriented Design (Early 1970s)</u>**

- It is much more important to pay attention to the design of the important data structures of the program than to the design of its control structure.
- Designtechniquesbasedonthisprinciplearecalled*datastructure-oriented*design techniques.
- This helped to derive the program structure from its data structure representation.
- A program is called structured, When it uses only the following types of constructs
  - Sequence
  - Selection
  - Iteration
  - Easier to read, understand and maintain
  - Requires less effort and time for development, less buggy

e) **<u>Data Flow-oriented Design (Late 1970s)</u>**

- In this, the major data items handled must be identified and then the processing required on these data items to produce required output must be determined.
- The functions (processes) and the data items that are exchanged between the different function are represented as Data Flow Diagram (DFD)



Data Flow Model of a Car Assembly Unit

The functions (also called as *processes*)and the data items that are exchanged between the different functions are represented in a diagram known as a *data flow diagram* (DFD).

f) <u>Object-oriented Design</u>

- Data flow-oriented techniques evolved in to object-oriented design (OOD) techniques in the late seventies.
- Natural objects relevant to a problem are first identified and then the relationships among the objects such as composition, reference, and inheritance are determined.
- Each object essentially acts as a *data hiding*(also known as *data abstraction*)entity.
- Object-Oriented Techniques have gained wide acceptance:
    - Simplicity
    - Reuse possibilities
    - Lower development time and cost
    - More robust code
    - Easy maintenance

g) <u>Web based design</u>

- Many of the present day software are required to work in a client-server environment through a web browser-based access (called *web-based software*).

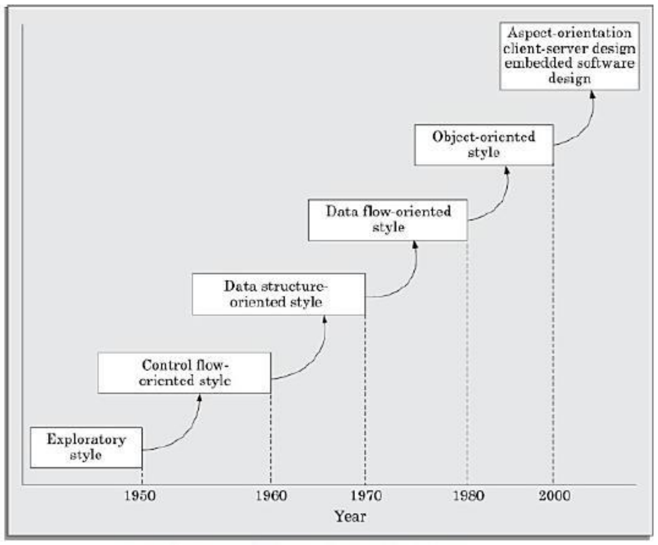

Aspect-orientation client-server design embedded software design
Object-oriented style
Data flow-oriented style
Data structure-oriented style
Control flow-oriented style
Exploratory style
1950
1960
1970
1980
2000
Year

Figure 1.12: Evolution of software design techniques.

1.5. Notable Changes in Software Development Practices

It is worthwhile to examine the glaring differences that you would notice when you observe an exploratory style of software development and another development effort based on modern software engineering practices.

- ✓ An important difference is that the exploratory software development style is based on error correction (build and fix) while the software engineering techniques are based on the principles of error prevention.
- ✓ In the exploratory style, coding was considered synonymous with software development. For instance, this naive way of developing a software believed in developing a working system as quickly as possible and then successively modifying it until it performed satisfactorily. Exploratory programmers literally dive at the computer to get started with their programs even before they fully learn about the problem!!!
- ✓ A lot of attention is now being paid to requirements specification. Significant effort is being devoted to develop a clear and correct specification of the problem before any development activity starts. Unless the requirements specification is able to correctly capture the exact customer requirements, large number of rework would be necessary at a later stage. Such rework would result in higher cost of development and customer dissatisfaction.
- ✓ Now there is a distinct design phase where standard design techniques are employed to yield coherent and complete design models
- ✓ Periodic reviews are being carried out during all stages of the development process. The main objective of carrying out reviews is phase containment of errors, i.e. detect and correct errors as soon as possible.
- ✓ Today, software testing has become very systematic and standard testing techniques are available.
- ✓ There is better visibility of the software through various developmental activities.
- ✓ Now, projects are being thoroughly planned.
- ✓ Several metrics (quantitative measurements) of the products and the product development activities are being collected to help in software project management and software quality assurance.

1.6 Computer Systems Engineering

In all the discussions so far, we assumed that the software being developed would run on some general-purpose hardware platform such as a desktop computer or a server. But, in several situations it may be necessary to develop special hardware on which the software would run. Examples of such systems are numerous, and include a robot, a factory automation system, and a cell phone. In a cell phone, there is a special processor and other specialized devices such as a speaker and a microphone. It can run only the programs written specifically for it.

Development of such systems entails development of both software and specific hardware that would run the software.

Computer systems engineering addresses development of such systems requiring development of both software and specific hardware to run the software. Thus, systems engineering encompasses software engineering.

The general model of systems engineering is shown schematically in Figure. One of the important stages in systems engineering is the stage in which decision is made regarding the parts of the problems that are to be implemented in hardware and the ones that would be implemented in software. This has been represented by the box captioned hardware-software partitioning in the Fig. While partitioning the functions between hardware and software, several trade-offs such as flexibility, cost, speed of operation, etc., need to be considered.

The functionality implemented in hardware run faster. On the other hand, functionalities implemented in software is easier to extend.

Further, it is difficult to implement complex functions in hardware. Also, functions implemented in hardware incur extra space, weight, manufacturing cost, and power overhead

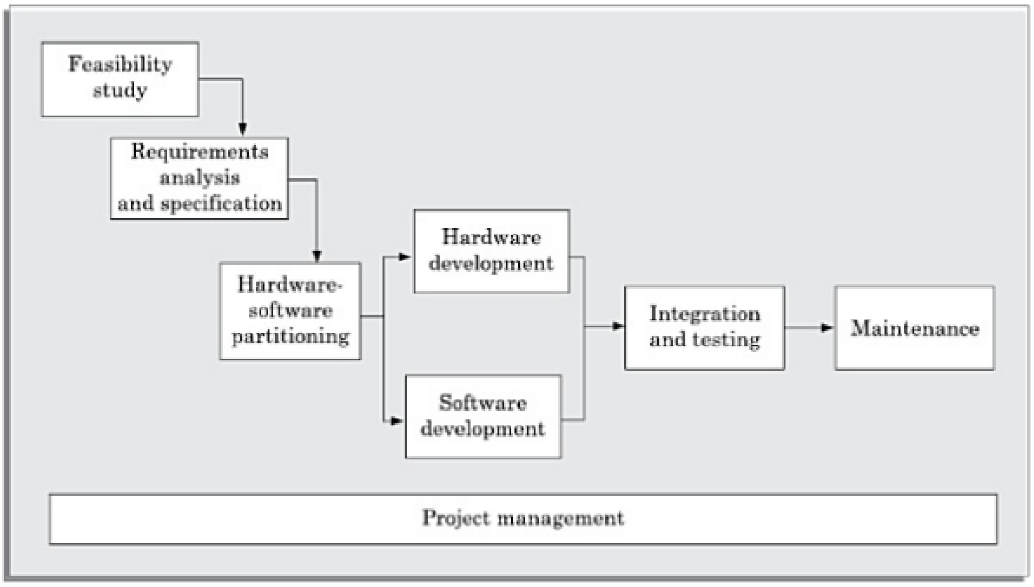


Figure 1.13: Computer systems engineering.

1.7 SOFTWARE LIFE CYCLE MODELS

In the software engineering approaches emphasize software development through a well-defined and ordered set of activities. These activities are graphically modeled (represented) as well as textually described and are variously called as ***software life cycle model***, ***software development lifecycle (SDLC) model***, and ***software development process model***.

a) A FEW BASIC CONCEPTS

Software life cycle

- The software life cycle has been defined to imply the different stages (or phases) over which a software evolves from an initial customer request for it, to a fully developed software, and finally to a stage where it is no longer useful to any user, and then it is discarded.
- The life cycle of every software starts with a request for it by one or more customers.
- This stage where the customer feels a need for the software and forms rough ideas about the required features is known as the **inception stage**.

- A software evolves through a series of identifiable stages(also called phases)on account of the development activities carried out by the developers, until it is fully developed and is released to the customers.
- Once installed and made available for use, the users start to use the software. This signals the start of the operation (also called **maintenance**) phase.
- The maintenance phase usually involves continually making changes to the software to accommodate the bug-fix and change requests from the user.
- The **operation phase** is usually the longest of all phases and constitutes the useful life of a software.
- Finally the software is retired, when the users do not find it any longer useful.
- The life cycle of software represents the series of identifiable stages through which it evolves during its life time.

<u>Software development lifecycle(SDLC)model</u>

- A **software development life cycle** (SDLC) model (also called software life cycle model and software development process model) describes the different activities that need to be carried out for the software to evolve in its life cycle.
- The terms **software development life cycle (SDLC)** and software development process are interchangeable.
- An SDLC graphically depicts the different phases through which software evolves. It is usually accompanied by a textual description of the different activities that need to be carried out during each phase.

<u>Process *versus* methodology</u> is at time used interchangeably, there is a subtle difference between the two. First, the term **process** has a broader scope and addresses either all the activities taking place during software development, or certain coarse grained activities such as design (e.g. design process), testing (test process), etc. Further, a software process not only identifies the specific activities that need to be carried out, but may also prescribe certain **methodology** for carrying out each activity.

<u>A software development process</u> has a much broader scope as compared to a **software development methodology**. A **process** usually describes all the activities starting from the inception of a software to its maintenance and retirement stages, or at least a chunk of activities in the life cycle. It also recommends specific methodologies for carrying out each activity. A **methodology**, in contrast, describes the steps to carry out only a single or at best a few individual activities.

<u>Software development organizations</u> have realized that adherence to a suitable life cycle model helps to produce good quality software and that helps minimize the chances of time and cost overruns.

- **Programming-in-the-small** refers to development of a toy program by a single programmer.
- While development of a software of the former type could succeed even while an individual programmer uses a **build and fix** style of development,
- **Programming-in-the-large** refers to development of professional software through team

effort.

- Use of a suitable SDLC is essential for a professional software development project involving team effort to succeed.
- A **documented development process** forms a common understanding of the activities to be carried out among the software developers and helps them to develop software in a systematic and disciplined manner.
- A **documented development process model**, besides preventing the misinterpretations that might occur when the development process is not adequately documented, also helps to identify inconsistencies, redundancies, and omissions in the development process.

Phase entry and exit criteria

- A good SDLC besides clearly identifying the different phases in the life cycle, should unambiguously define the entry and exit criteria for each phase.
- The phase entry(or exit)criteria is usually expressed as a set of conditions that needs to be be satisfied for the phase to start (or to complete).
- As an example, the phase exit criteria for the software requirements specification phase, can be that the *software requirements specification* (SRS) document is ready, has been reviewed internally, and also has been reviewed and approved by the customer.

Only after these criteria are satisfied, the next phase can start.

SDLC Models

- ➢ There are various software development life cycle models defined and designed which are followed during the software development process.
- ➢ These models are also referred as “Software Development Process Models".
- ➢ Each process model follows a Series of steps unique to its type to ensure success in the process of software development.
- ➢ Following are the most important and popular SDLC models followed in the industry:
  - ✓ Waterfall Model (**Classical Waterfall Model)**
  - ✓ Iterative Model
  - ✓ Evolutionary Model
  - ✓ Incremental Model
  - ✓ V-Model
  - ✓ Prototype Model
  - ✓ Spiral Model
  - ✓ RAD Model
  - ✓ Agile Model

1.8 WATERFALL MODEL AND ITS EXTENSIONS

- ➢ The Waterfall Model (1970s)was the first Process.
- ➢ It is also referred to as a linear-sequential life cycle model.
- ➢ In this model whole application is developed in a sequential approach.
- ➢ It is very simple to understand and use.
- ➢ In this model, each phase must be completed before the next phase can begin and there is no

overlapping in the phases.

1. **Classical Waterfall Model**
2. **Iterative Waterfall Model**
3. **V-Model**
4. **Prototyping Model**
5. **Incremental Development Model**
6. **Evolutionary Model**

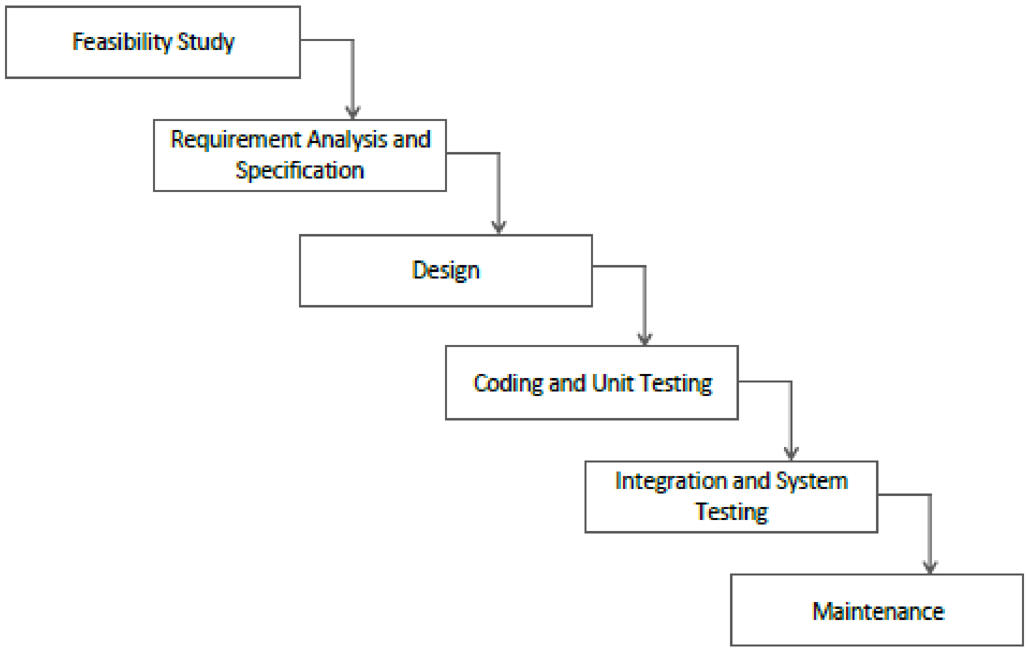
1. CLASSICAL WATERFALL MODEL

✓Classical waterfall model provides a systematic and sequential approach to software development and is better than the build and fix approach.
✓It is easy to understand, implement and easy to maintain.
✓It clearly divides the life cycle into phases that may be performed independently.

The different phases of this model are

•Feasibility Study
• Requirements Analysis and Specification
• Design
• Coding and Unit Testing
• Integration and System Testing
• Maintenance.

•Phases of the **Classical Waterfall Model**:


Feasibility Study
Requirement Analysis and Specification
Design
Coding and Unit Testing
Integration and System Testing
Maintenance

Phases of the classical waterfall model

- The different phases are feasibility study, requirements analysis and specification, design, coding and unit testing, integration and system testing, and maintenance.
- The phases starting from the feasibility study to the integration and system testing phase are

known as the development phases.

- A software is developed during the development phases, and at the completion of the development phases, the software is delivered to the customer.

- After the delivery of software, customers start to use the software signaling the commencement of the **operation** phase.
- As the customers start to use the software, changes to it become necessary on account of bug fixes and feature extensions, causing maintenance works to be undertaken.
- Therefore, the last phase is also known as the **maintenance.**
- An activity that spans all phases of software development is **project management**. Since it spans the entire project duration, no specific phase is named after it.
- In the waterfall model, different lifecycle phases typically require relatively different amounts of efforts to be put in by the development team.
- On the average, about 60 percent of the total effort put in by the development team in the entire lifecycle is spent on the maintenance activities alone.
- However, among the development phases, the integration and system testing phase requires the maximum effort in a typical development project.

I. Feasibility study

➢ A feasibility study is a detailed analysis that determines if a proposed project is likely to be successful.

➢ It assesses many factors, including technical, economic, legal, operational, and scheduling aspects.

➢ The study provides information to help guide the next steps for the project.

An abstract problem definition:

- Only important requirements of customers are collected others are ignored.
- Formulation of the different strategies for solving the problem.
- Evaluation of different solution strategies. i.e. estimates of resource required, cost, time, etc.

II. Requirement analysis and specification

➢ It has two phase

- Requirement gathering and analysis
- Requirement specification

Requirement gathering and analysis

- The goal of requirement gathering is to collect are relevant information from the *customer with a clear view*

Requirement specification

Both analysis and gathering activity are organized into Software Requirement Specification (*SRS*) document. The three important contents of this documents are

✓ Functional requirement
✓ Non function requirement
✓ Goals of implementation

The SRS serves as a contract between development team and the customer

III. Design

- Goal of design is to transform the requirements in *SRS document into a structure* suitable for implementation. i.e During design phase, Software architecture is derived from the SRS document. There are two design approaches.

a. *Traditional design approach:*

- It is based on data – flow oriented design approach.
- Structured analysis is carried out followed by structured design activity.
- Data Flow Diagram (DFD) are used to perform structured analysis.
- Structured design has two activities i.e. *architectural design* and *detailed design*

b. *Object Oriented design approach:*

- Various objects that occur in the problem domain and solution domain are identified.
- The relationship between these objects are identified.
- It is further refined to obtain detailed design.

IV. Coding and Unit testing

- The purpose of this phase is to translate the software design into source code.
- Each component of design is implemented as a program module.
- After coding is completed, *each module is unit tested*.
- Each module is *documented*.
- The main objective of unit testing is to determine the correct working of individual modules.

V. Integration and System testing

- During this phase, the different modules are integrated.
- It is carried out incrementally over a no. of steps.
- *After integrating all modules system testing is carried out*.

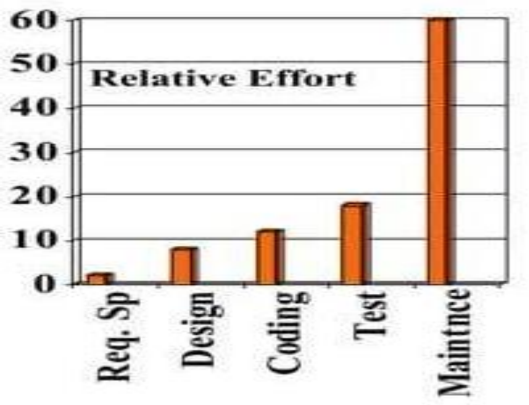
  There are three types of system testing.
  - α-testing - testing performed by the development team.
  - β-testing – testing performed by a friendly set of customer.
  - Acceptance testing – performed by the customer after product delivery to find whether to accept or reject it.

VI. Maintenance

- Maintenance requires more effort. It is roughly in 40:60 ratio. There are three kinds of activities.
- *Corrective maintenance :*It involves in correcting the errors found during product development phase.
- *Perfective maintenance :*It involves in improving and enhancing the functionalities of the system.
- *Adaptive maintenance :*It is required for porting the software to work in a new environment.

<u>Relative Effort for Phases</u>

- Phases between feasibility study and testing known as *development phases*.
- Among all life cycle phases *maintenance phase* consumes maximum effort.
- Among development phases, *testing phase* consumes the maximum effort.



Limitations:

- Oldest model
- Has sequential development requirements → maintenance
- No reverse direction(water fall)
- Only when we are clear about requirements
  - ✓ Low budget projects, So it is not suitable for the large projects
  - ✓ In waterfall model, a working model of software is not available.
- No customer involvement
- Will not allow any changes

Shortcomings of the classical waterfall model

The classical waterfall model is a very simple and intuitive model. However, it suffers from several short comings.

- **No feedback paths**
  - ✓ Just as water in a waterfall after having flowed down cannot flow back (no reverse direction), once a phase is complete, the activities carried out in it and this phase are considered to be final and are closed for any rework.
  - ✓ This requires that all activities during a phase are flawlessly carried out.
  - ✓ The classical waterfall model incorporates no mechanism for error correction.
  - ✓ Programmers are humans and as the old adage says *to err is human.*
  - ✓ *The cause for errors can be many—oversight,* wrong interpretations, use of incorrect solution scheme, communication gap, etc.
  - ✓ These defects usually get detected much later in the life cycle like in coding or testing.
  - ✓ Once a defect is detected at a later time, the developers need to redo some of the work done during that phase.
  - ✓ Therefore, it becomes impossible to strictly follow the classical waterfall model of software development.

- **Difficult to accommodate change requests:**
  - This model assumes that all customer requirements can be completely and correctly defined at the beginning of the project.
  - The customers' requirements usually keep on changing with time.
  - But, in this model it is difficult to accommodate the requirement change requests made by the customer after the requirements specification phase is complete.

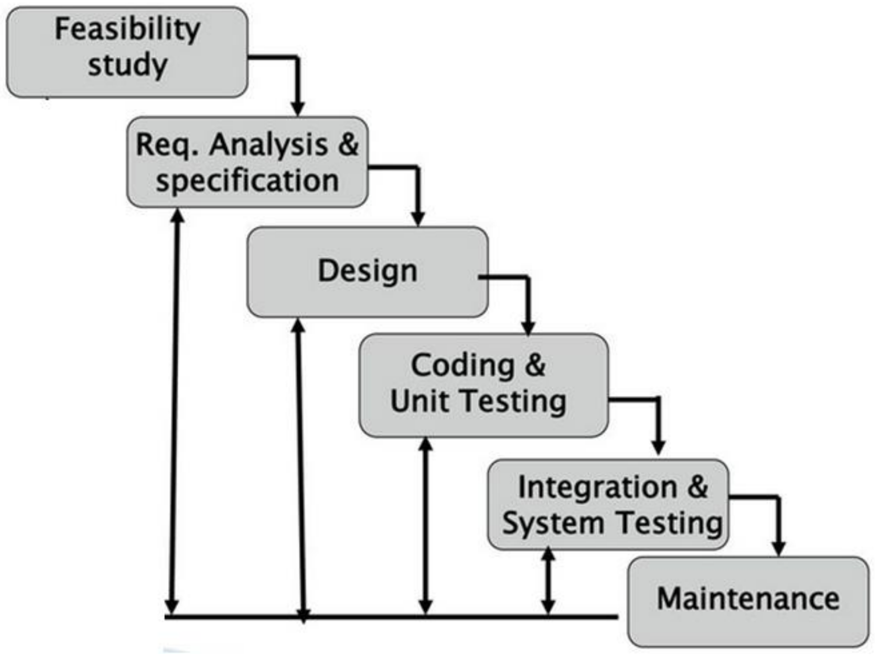
- **Inefficient error corrections:**
  - This model defers integration of code and testing tasks until it is very late when the problems are harder to resolve.

- **No overlapping of phases:**
  - This model recommends that the phases be carried out sequentially—new phase can start only after the previous one completes.
  - For example, for efficient utilization of manpower, the testing team might need to design the system test cases immediately after requirements specification is complete.
  - In this case, the activities of the design and testing phases overlap.
  - Consequently, it is safe to say the different phases need to overlap for cost and efficiency reasons.

2. ITERATIVE WATERFALL MODEL:

- This model was developed to remove the shortcomings of waterfall model.
- This is also referred as ***Modified Waterfall Life-Cycle Model***.
- The classical waterfall model is an idealistic model because it assumes that development error is ever committed by the engineers during any phase of life cycle of software development.
- But in practical environment of development, engineers do commit large number of errors in almost every phase of life cycle of software development.
- Once defect is detected, the engineers need to go back to the phase where the defects had occurred and redo some work done during that phase and the subsequent phases to correct the defects and its effect on the latter phases.
- Therefore a ***feedback*** path is needed in the classical waterfall model from every phase to its preceding phase.
- The principle of detecting errors as close to their points of introduction as possible is called ***phase containment error.***

- The main change brought about by the iterative waterfall model to the classical waterfall model is in the form of providing feedback paths from every phase to its preceding phases.



- In spite of best efforts put to detect errors in the same phase some errors do escape detection and may get noticed in the later phase.
- Thus practically several iterations through the waterfall stages are normally necessary to develop the final products.
- In this model, the phase of software development remain the same, but the construction and delivery is done in the iterative mode.

- The feedback paths allow for correction of the errors committed during a phase, as and when errors are detected in later phases i.e. it allows to correct the errors found in that phase.
- But there is no feedback path to the feasibility stage.

- **Phase Containment of Errors:**
  - Though errors cannot be avoided, it is desirable to detect the errors in the same phase in which they occur.
  - This can reduce the effort required for correcting bugs.
  - Eg. If a problem is found in design phase, it must be identified and corrected in that phase itself.
  - The errors should be detected as early as possible.
  - The principle of detecting errors as close to their point of introduction as possible is known as phase containment of errors.

- **How can phase containment of errors be achieved?**
  - An important technique is frequently used to conduct review
  - after every milestone.
  - In spite of best effort to detect error in the same phase, still
  - some errors can escape.
  - So rework of already completed phase is required.
  - Thus cannot complete phase at specified time.
  - This makes the different life cycle phase overlap in time.

- **Shortcomings of the iterative waterfall model**
  - The iterative waterfall model is a simple and intuitive
  - software development model.
  - It was used satisfactorily during 1970s and 1980s.
  - The projects are now shorter, and involve Customized
  - software development.
  - Software was earlier developed from scratch.
  - Now reuse of code is possible.
  - The software services (customized software) are poised to become the dominant types of projects.

- **Difficult to accommodate change requests:**
  - A major problem with the waterfall model is that the requirements need to be frozen before the development starts.
  - Accommodating even small change requests after the
  - development activities are difficult.
  - Once requirements have been frozen, the waterfall model
  - provides no scope for any modifications to the requirements.
  - Requirement changes can arise due to a variety of reasons including the following—requirements were not clear to the customer, requirements were misunderstood, business process of the customer may have changed after the SRS document was signed off, etc.
  - In fact, customers get clearer understanding of their requirements only after working on a fully developed and installed system.

- **Incremental delivery not supported:**
  - In the iterative waterfall model, the full software is completely developed and tested before it is delivered to the customer.
  - There is no provision for any intermediate deliveries to occur.
  - This is problematic because the complete application may take several months or years to be completed and delivered to the customer.
  - By the time the software is delivered, installed, and becomes ready for use, the customer's business process might have changed substantially.
  - This makes the developed application a poor fit to the customer's requirements.
- **Phase overlap not supported:**
  - For most real life projects, it becomes difficult to follow the rigid phase sequence prescribed by the waterfall model.
  - By the term a rigid phase sequence, we mean that a phase can start only after the previous phase is complete in all respects.
  - Strict adherence to the waterfall model creates blocking states.

- **Error correction unduly expensive:**
  - In waterfall model, validation is delayed till the complete development of the software.
  - As a result, the defects that are noticed at the time of validation incur expensive rework and result in cost escalation and delayed delivery.
- **Limited customer interactions:**
  - This model supports very limited customer interactions.
  - It is generally accepted that software developed in isolation from the customer is the cause of many problems.
  - Interactions occur only at the start of the project and at
  - project completion.
  - As a result, the developed software usually turns out to be a misfit to the customer's actual requirements.

- **Heavy weight:**
  - The waterfall model over emphasizes documentation.
  - A significant portion of the time of the developers is spent in preparing documents, and revising them as changes occur over the life cycle.
  - Heavy documentation though useful during maintenance
  - and for carrying out review, is a source of team inefficiency.
- **No support for risk handling and code reuse:**
  - It becomes difficult to use the waterfall model in projects that are susceptible to various types of risks, or those involving significant reuse of existing development artifacts.

3. V-Model

- V –model means ***Verification*** and ***Validation*** model.

<u>Verification:</u>

"Are we building the product right".

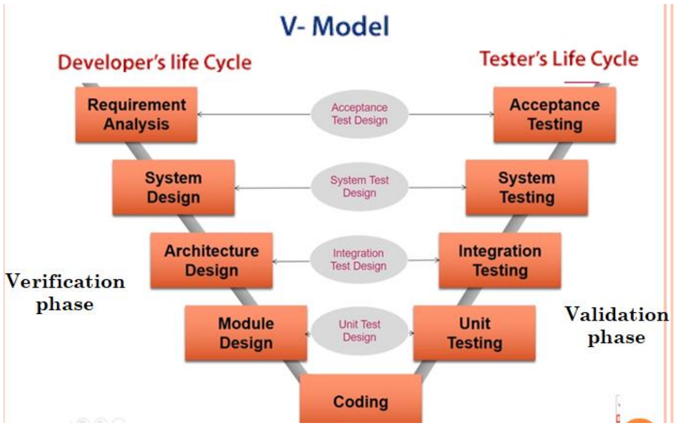
The software should conform to its specification.

<u>Validation:</u>

"Are we building the right product".

The software should do what the user really wants.

- V-shaped life cycle is a sequential path of execution of processes.
- Each phase must be completed before the next phase starts.
- Testing of the product is planned in ***parallel*** with a corresponding phase of development.
- The V-Shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.



There are the various phases of Verification Phase of V-model:

1. **Requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

There are the various phases of Validation Phase of V-model:

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client's business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the

compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

Merits

- ✓ Simple and easy to use.
- ✓ Testing activities like planning, test designing happens well before coding.
- ✓ This saves a lot of time. Hence higher chance of success over the waterfall model.
- ✓ Proactive defect tracking - that is defects are found at early stage.
- ✓ Avoids the downward flow of the defects.
- ✓ Works well for small projects where requirements are easily understood.

Demerits

- ✓ Very rigid and least flexible.
- ✓ Software is developed during the implementation phase, so no early prototypes of the software are produced.
- ✓ If any changes happen in midway, then the test documents along with requirement documents has to be updated.

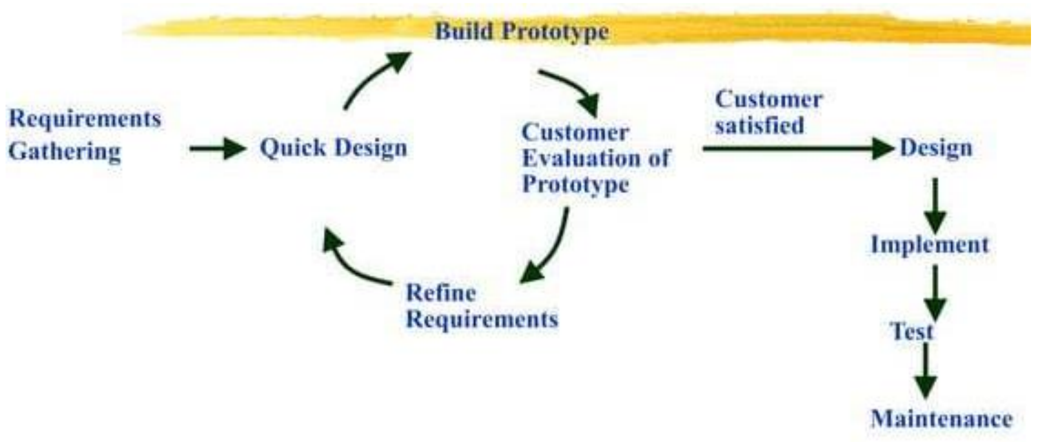
4. PROTOTYPING MODEL

The prototype model suggests building a working prototype of the system, before development of the actual software.

A prototype is a toy and crude implementation of a system.

- • Before starting actual development,
  - ✓ a working prototype of the system should first be built.
- • A prototype is a **toy** implementation of a system:
  - ✓ limited functional capabilities,
  - ✓ low reliability,
  - ✓ inefficient performance.
- • The reason for developing a prototype is:
  - - it is impossible to “get it right” the first time,
  - - we must plan to throw away the first product
  - * if we want to develop a good product.
- • The developed prototype is submitted to the customer for his evaluation:
  - -Based on the user feedback, requirements are refined.
  - -This cycle continues until the user approves the prototype.
- • The actual system is developed using the classical waterfall approach.

For example, tools based on fourth generation languages (4GL) may be used to construct the prototype for the GUI parts.



• Requirements analysis and specification phase becomes redundant:

→final working prototype (with all user feedbacks incorporated) serves as an *animated requirements specification.*

• Design and code for the prototype is usually thrown away:

→However, the experience gathered from developing the prototype helps a great deal while developing the actual product.

• Even though construction of a working prototype model involves additional cost - -

✓ overall development cost might be lower for:

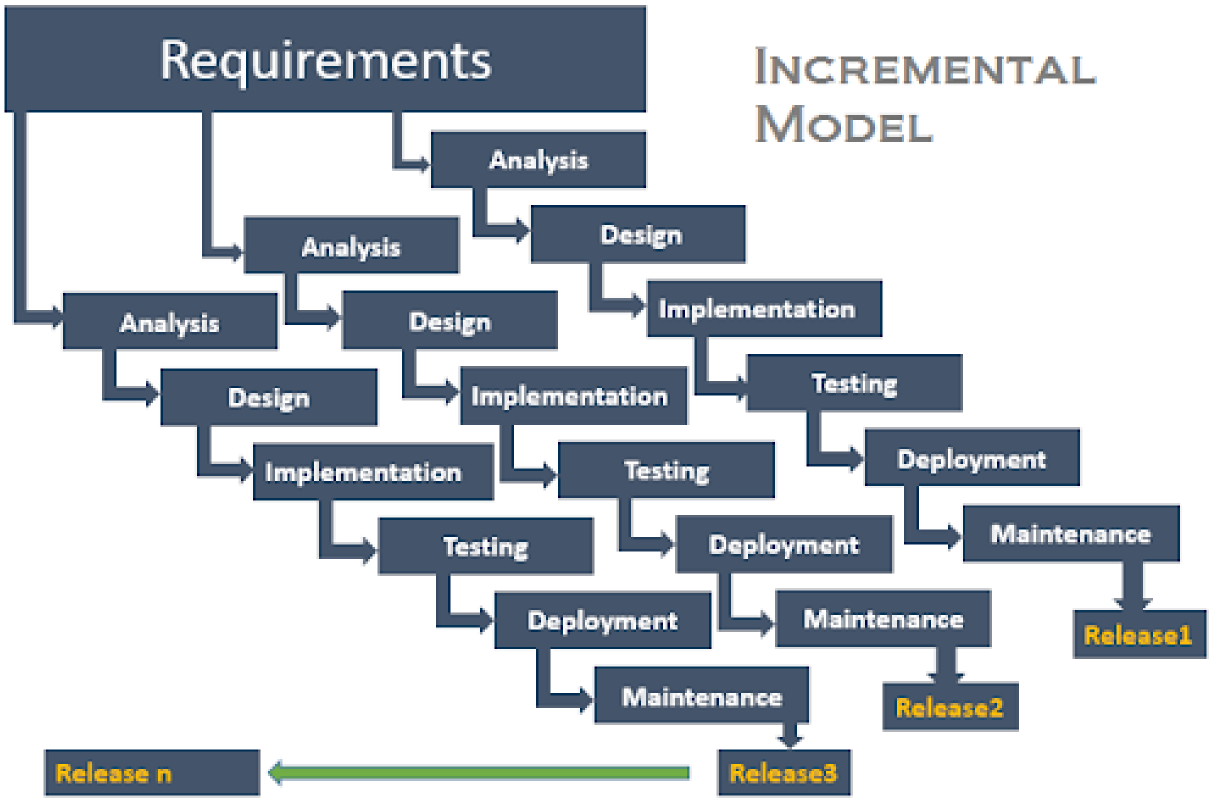
- systems with unclear user requirements,
- systems with unresolved technical issues.

• Many user requirements get properly defined and technical issues get resolved:

- these would have appeared later as change requests and resulted in incurring massive redesign costs.

5. INCREMENTAL MODEL

- Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle.
- Each iteration passes through the requirements, design, coding and testing phases.
- Typical product takes from 5 to 25 builds (iterations).
- Waterfall and rapid prototyping models
  - Deliver complete product at the end
- Incremental model
  - Deliver portion of the product at each stage



Advantages

- ✓ The software will be generated quickly during the software life cycle.
- ✓ It is flexible and less expensive to change requirements and scope.
- ✓ Throughout the development stages changes can be done.
- ✓ This model is less costly compared to others.
- ✓ A customer can respond to each building.
- ✓ Errors are easy to be identified.

Disadvantages:

- ✓ It requires a good planning designing.
- ✓ Problems might arise due to system architecture as not all requirements collected up front for the entire software lifecycle.
- ✓ Each iteration phase is rigid and does not overlap each other.
- ✓ Correcting a problem in one unit requires correction in all the units and consumes a lot of time.

When to use Incremental models?

- ✓ Requirements of the system are clearly understood
- ✓ When demand for an early release of a product arises
- ✓ When software engineering team are not very well skilled or trained
- ✓ When high-risk features and goals are involved
- ✓ Such methodology is more in use for web application and product based companies

6. EVOLUTIONARY MODEL

• Evolutionary model:

→The system is *broken down into several modules* which can be incrementally implemented and delivered.

• First develop the core modules of the system.

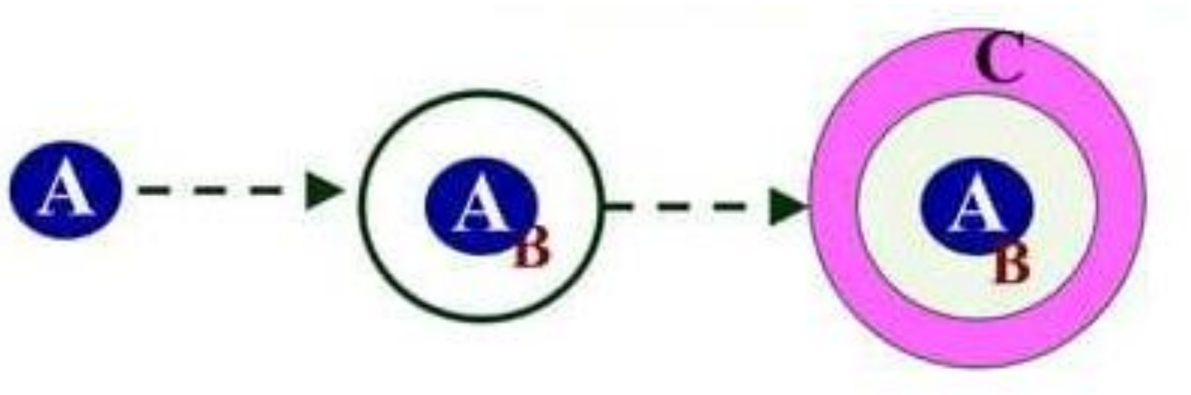
• The initial product skeleton is refined into increasing levels of capability:

→by adding new functionalities in successive versions.

• Successive version of the product:

- functioning systems capable of performing some useful work.
- A new release may include new functionality:

* also existing functionality in the current release might have been enhanced.



Advantages of Evolutionary Model

• Users get a chance to experiment with a partially developed system:

→ much before the full working version is released,

• Helps finding exact user requirements:

→ much before fully working system is developed.

• Core modules get tested thoroughly:

→reduces chances of errors in final product.

Disadvantages of Evolutionary Model

• Often, difficult to subdivide problems into functional units:

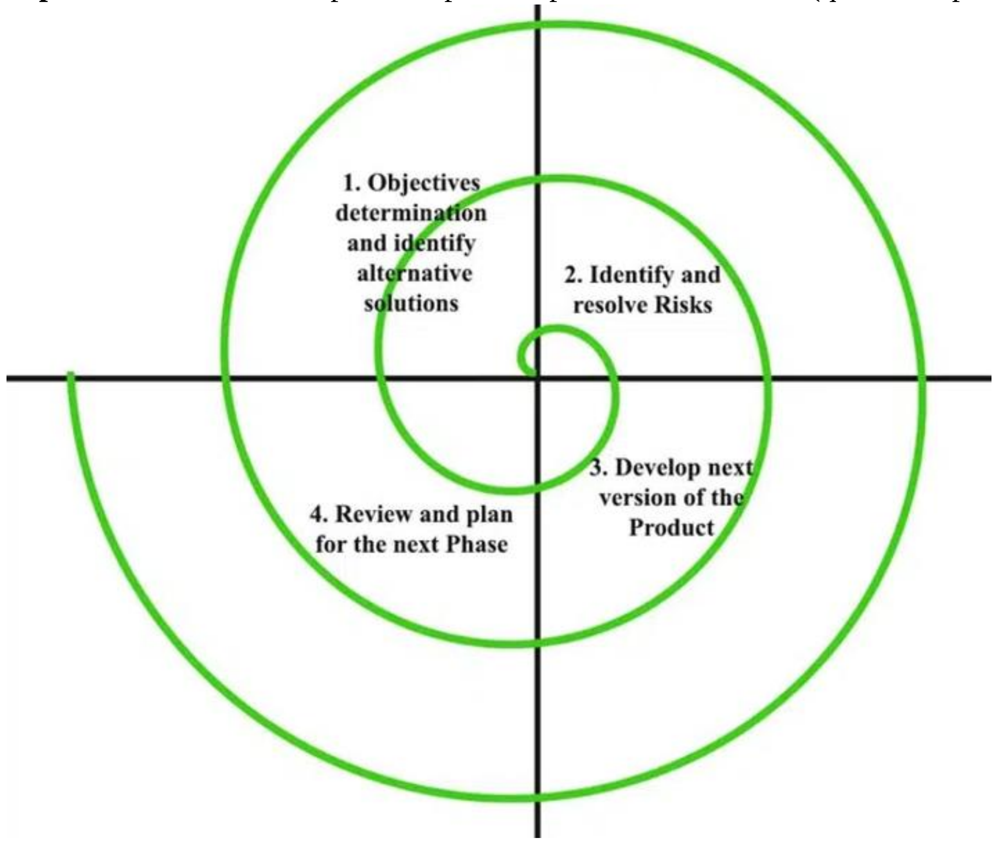
- which can be incrementally implemented and delivered.
- evolutionary model is useful for very large problems,

* where it is easier to find modules for incremental implementation.

7. SPIRAL MODEL

- The Spiral Model is a **Software Development Life Cycle (SDLC)** model that provides a systematic and iterative approach to software development by Boehm in 1988.
- In its diagrammatic representation, looks like a spiral with many loops.
- Each loop of the spiral represents a phase of the software development process.
- The exact number of loops of the spiral is unknown and can vary from project to project.
- It is based on the idea of a spiral, with each iteration of the spiral representing a complete software development cycle, from requirements gathering and analysis to design, implementation, testing, and maintenance.

Quadrants in spiral model : Each loop in the spiral is split into four sectors (quadrants/phases).


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

.

The functions of these four quadrants are discussed below:

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the ***Prototype is built*** for the best possible solution.
3. **Develop the next version of the Product:** During the third quadrant, the identified features are ***developed and verified through testing***. At the end of the third quadrant, the next version of the software is available.
4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so-far developed version of the software. In the end, ***planning for the next phase*** is started.

Advantages :

- Risk Identification at early stage.
- Good for large projects.
- Iterative and Incremental Approach.
- Flexibility for adding functionality.

Disadvantages:

- Costly.
- Risk dependent.
- Not suitable for smaller projects.
- Difficult to meet budget.

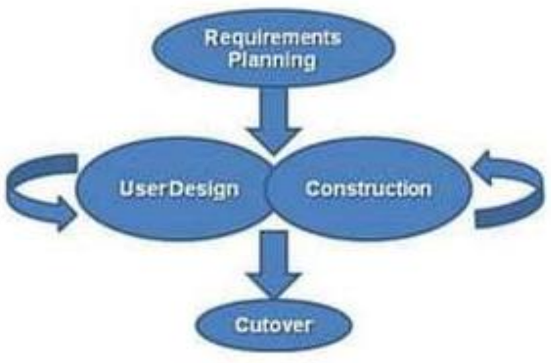
8. Rapid Application Development (RAD) Model

- IBM first proposed the Rapid Application Development or RAD Model in the 1980s. James martin formally developed the RAD model in 1991.
- Rapid Application Development(RAD) is an incremental software model that a short development cycle.
- The RAD process enables a development team to create a fully functional system within a very short time period.

- The **RAD (Rapid Application Development)** model is based on ***prototyping and iterative*** development with no specific planning involved.
- Rapid Application Development focuses on gathering ***customer requirements*** through **workshops** or **focus groups**, early testing of the prototypes by the customer using ***iterative*** concept, ***reuse of the existing prototypes*** (components), ***continuous integration*** and ***rapid delivery***.
- In the RAD model, the functional modules are developed in ***parallel as prototypes*** and are ***integrated to make the complete product for faster product delivery***.

- The prototype is refined based on the customer feedback.
- The development team almost always includes a customer representative to clarify the requirements.
- This is intended to make the system tuned to the exact customer requirements and also to bridge the communication gap between the customer and the development team.
- The development team usually consists of about five to six members, including a customer representative.

- Rapid Application Development is a **lightweight approach** to development. It is divided into four phases:
  - 1. Requirements Planning Phase
  - 2. User Design Phase
  - 3. Construction Phase
  - 4. Cutover Phase



RAD Model Design

Following are the various phases of the RAD Model −

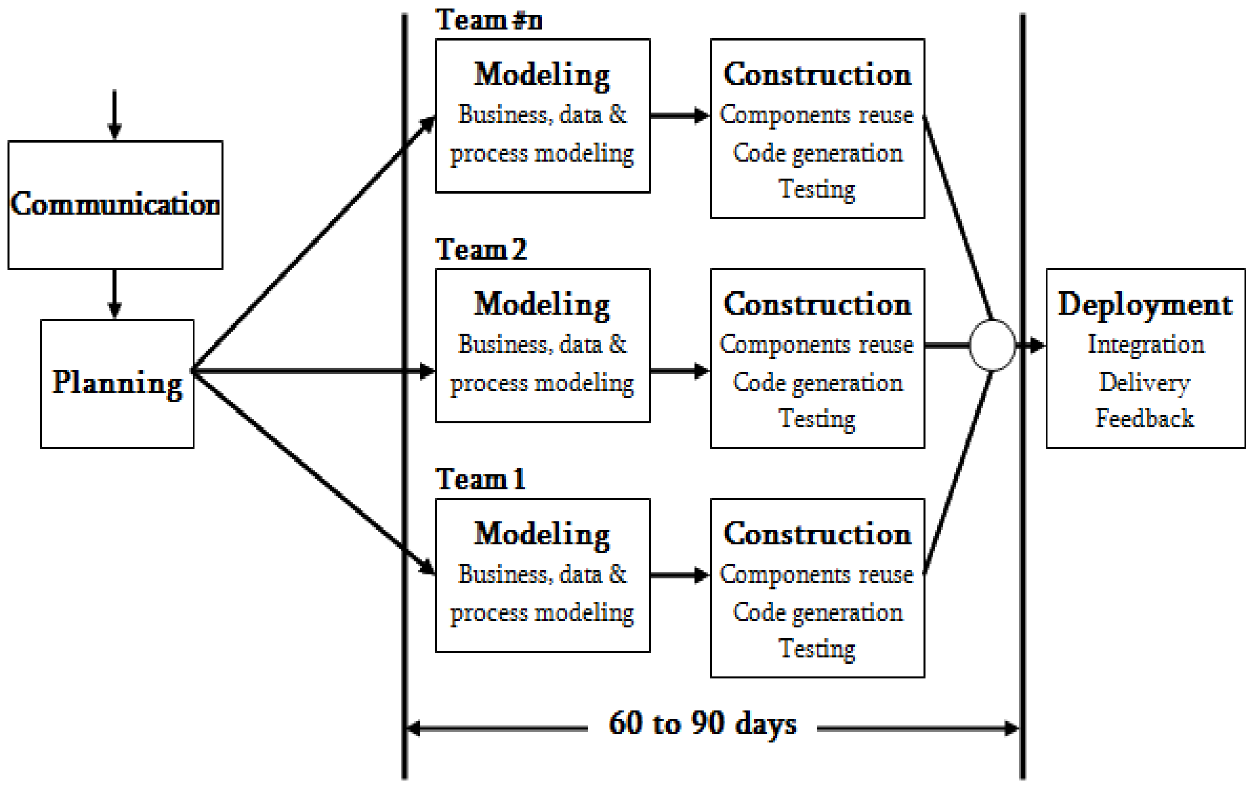
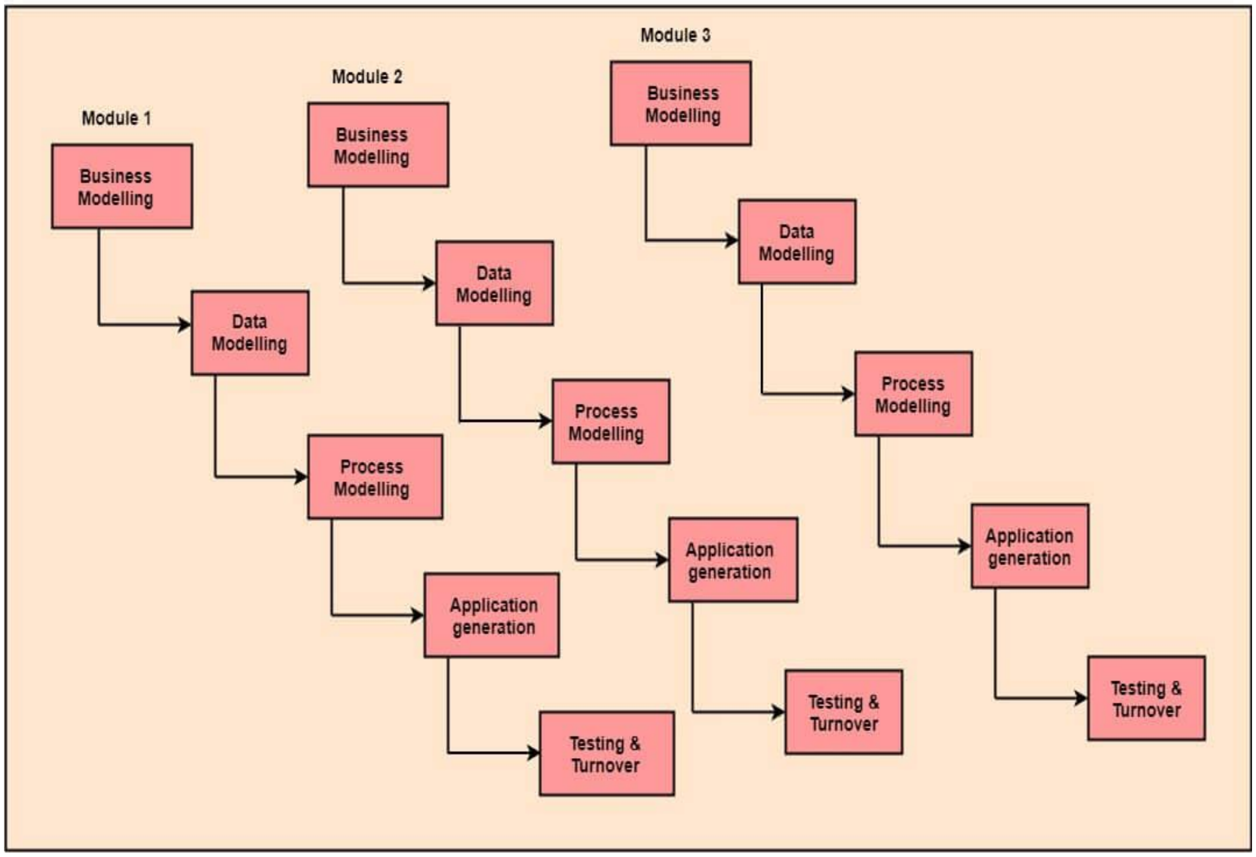


Figure : Flowchart of RAD model

Fig: RAD Model



The various phases of RAD are as follows:
1.Business Modeling: The information flow among ***business functions*** is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.
2. Data Modeling: The data collected from business modeling is refined into a set of ***data objects*** (entities) that are needed to support the business. The ***attributes*** (character of each entity) are identified, and the relation between these data objects (entities) is defined.
3. Process Modeling: The information object defined in the data modeling phase are transformed to achieve the ***data flow*** necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.
4. Application Generation: Automated tools are used to facilitate ***construction of the software***; even they use the 4th GL techniques.
5. Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

Advantages:
• Reduced development time.
- Increases reusability of components.
- Quick initial reviews occur.
- Parallel development.
- Encourages customer feedback.

• Efficient Documentation.
Disadvantages:

- User may not like fast activities.
    - Requires highly skilled developers/ designers.
- On the high technical risk, it's not suitable.
    - For smaller projects, we cannot use the RAD model.

The major goals of the RAD model are as follows:

- To decrease the time taken and the cost incurred to develop software systems.
- To limit the costs of accommodating change requests.
- T o reduce the communication gap between the customer and the developers.

WHEN TO USE RAD MODEL?

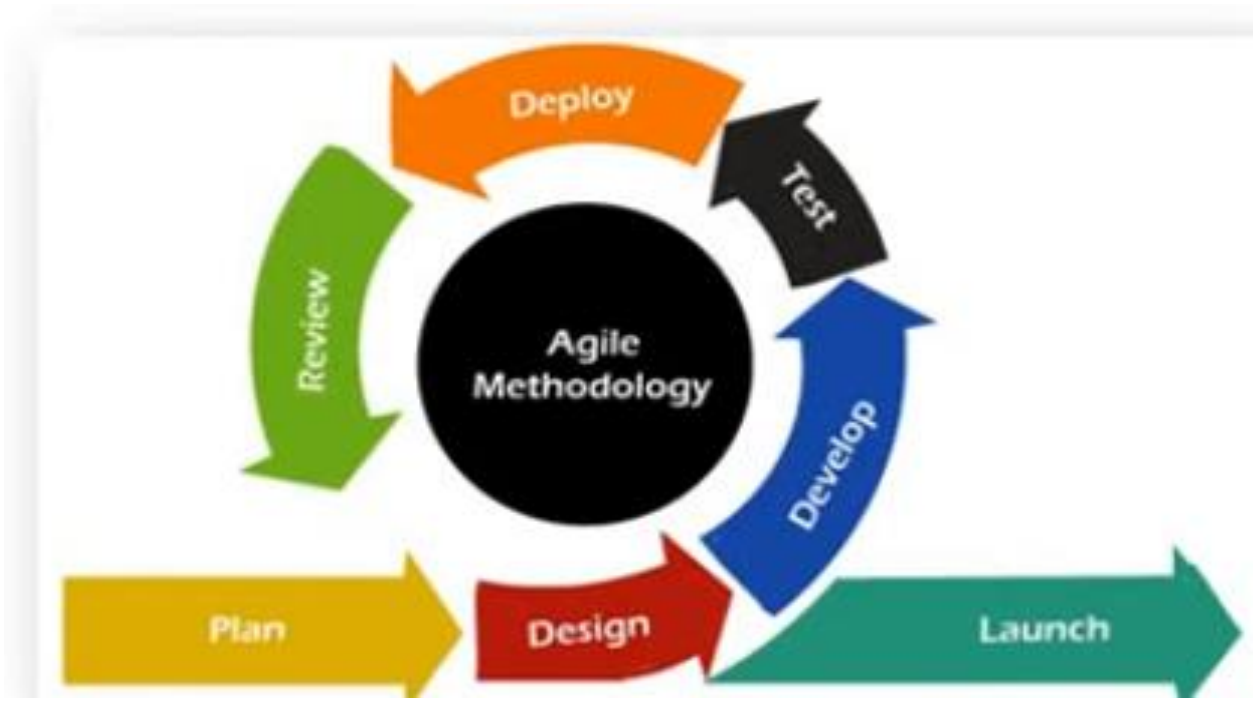
- When the system should need to create the project that modularizes in a short span time (2-3 months).
- When the requirements are well-known.
- When the technical risk is limited.
- When there's a necessity to make a system, which modularized in 2-3 months of period.
- It should be used only if the budget allows the use of automatic code generating tools.

9. AGILE MODEL

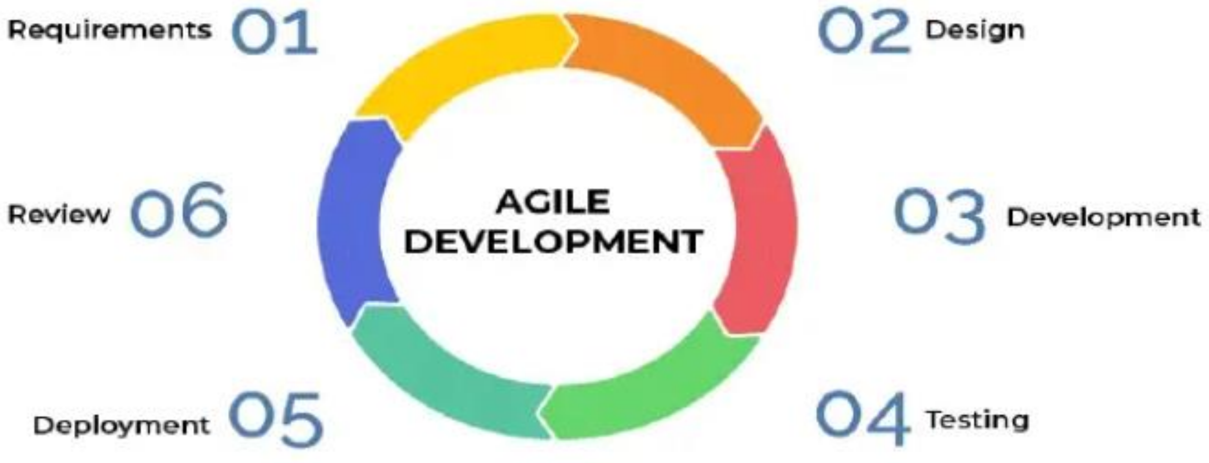
- The meaning of Agile is ***swift*** or ***versatile***.
- Mostly used model in today's digital era.
- Agile means "The ability to respond to changes from requirements, technology & people“
- Agile able to move quickly & easily.
- So, the main aim of the Agile model is to facilitate quick project completion.
- It is a combination of iterative and incremental process models.
- The main aim is to help a project to adapt to change requests quickly to facilitate quick project completion by removing unnecessary activities that waste time and effort.

Working with Example:

- Divides requirements into multiple iterations & provide specific functionality for the release.
- Delivers multiple software requirements.
- Each iterations are lasts from two to three weeks.
- Direct collaboration with customers.
- Rapid project development.

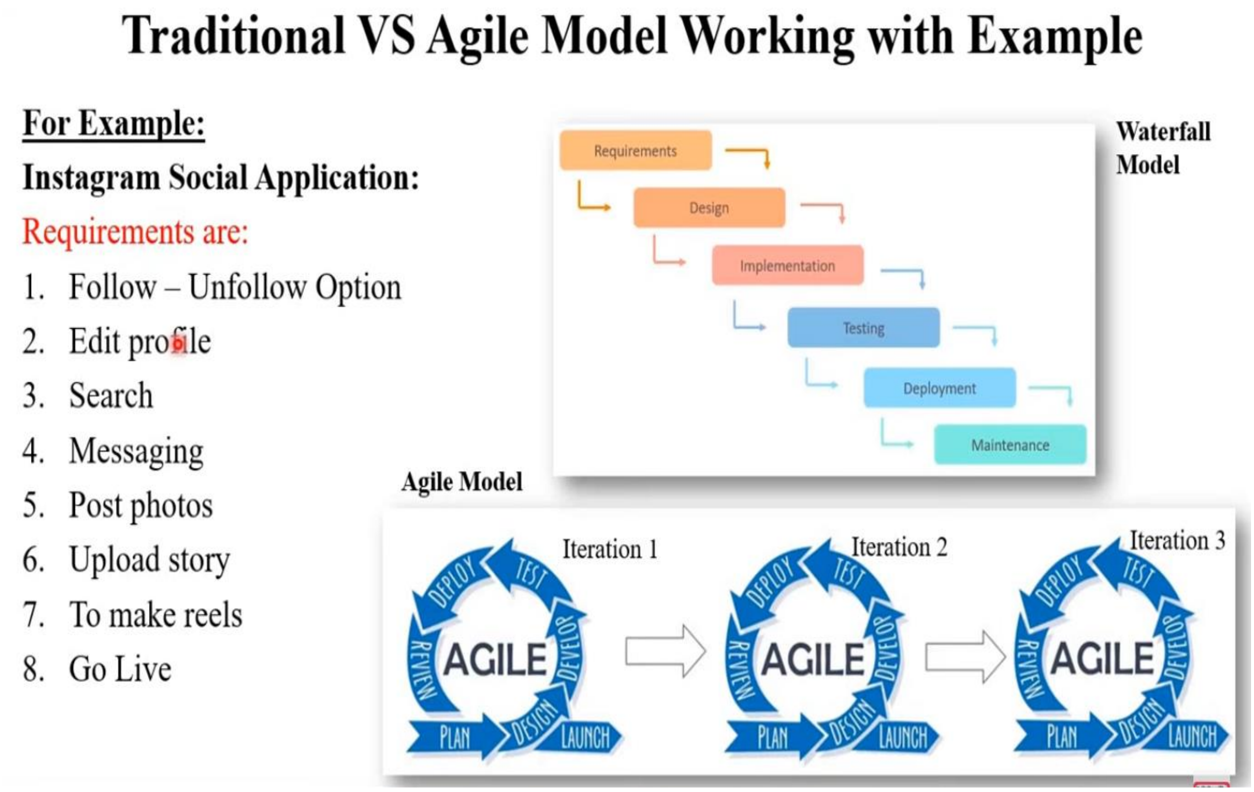


- In the Agile model, the requirements are decomposed into many small parts that can be incrementally developed.
- The Agile model adopts Iterative development.
- Each incremental part is developed over an iteration.
- Each iteration is intended to be small and easily manageable and can be completed within a couple of weeks only.
- At a time one iteration is planned, developed, and deployed to the customers.
- 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: 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: 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. Development: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product.

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

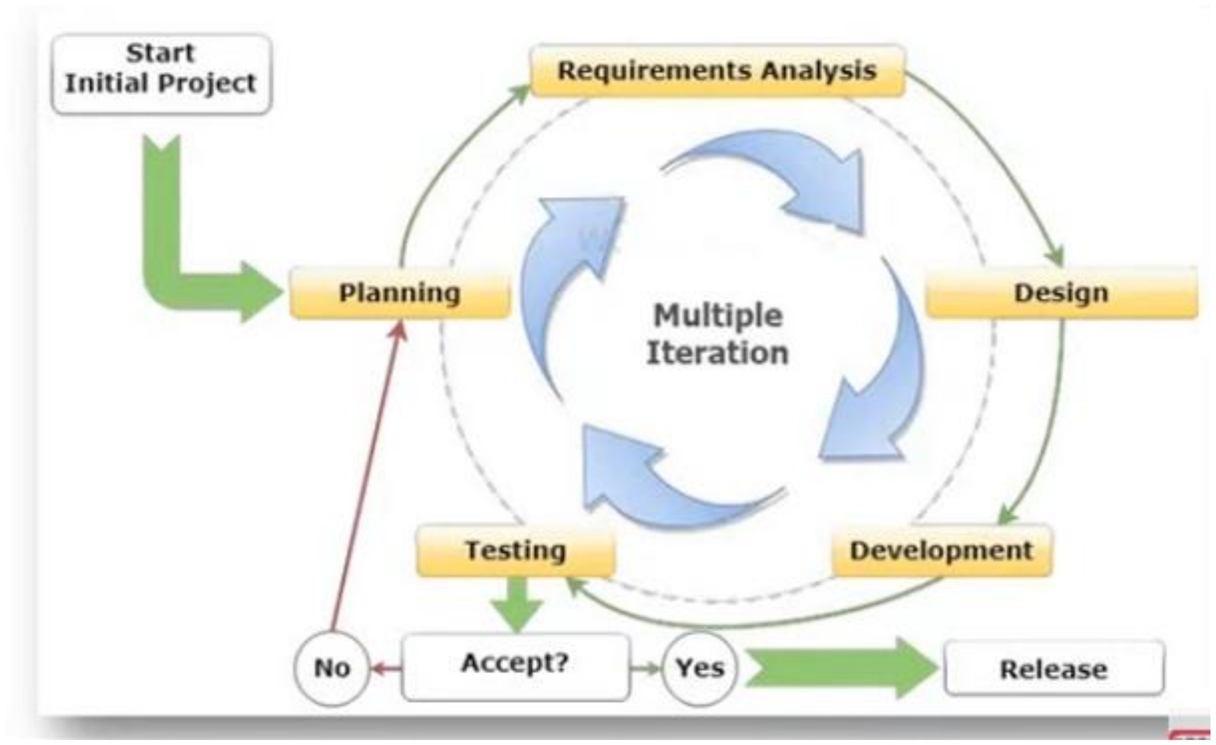


Traditional VS Agile Model Working with Example

For Example:
Instagram Social Application:
Requirements are:
1. Follow – Unfollow Option
2. Edit profile
3. Search
4. Messaging
5. Post photos
6. Upload story
7. To make reels
8. Go Live

When to use the Agile Model?

1. When project size is large.
2. When frequent changes are required.
3. When a highly qualified and experienced team is available.
4. When a customer is ready to have a meeting with a software team all the time.
5. Projects with flexible timelines and budget.



Agile Principles

1. Highest priority is to satisfy the customers to early & continue delivery of software.
2. Being flexible about changing requirements at any point of development.
3. Working on frequent and short deliveries like couple of weeks or months with preference.
4. Transparency between business people and developers and requires them to work together.
5. By providing a better productive environment and providing them with all the support, motivation it leads to better productivity.
6. Face to face communication as the most effective way to communicate between customer & development team.
7. Continuous attention towards effective designing and technical excellence through following optimal code standard.

8. It promotes sustainable development by developers, users and sponsors should work together.
9. Continuous attention to technical excellence and good design enhances agility. 。
10. Simplicity is the art of maximum result & less hard work by removing unnecessary tasks and prioritizing activities.
11. The best architectures, requirements, and designs emerge from self-organizing & experience teams.
12. For developing effective software, regular analysis and work on improving the overall delivery or the development process.

Advantages of Agile Model

1.Supports customer involvement and customer satisfaction
2.Strong communication of the software team with the customer.
3. Little planning required
4.Efficient design and fulfils the business requirement.

5.Anytime changes are acceptable.
6.Provides a very realistic approach to software development
7.Updated versions of functioning software are released every week. Can adapt to rapidly changing requirements and respond faster.
8. It reduces total development time.
9.Deployment of software is quicker.
10. Helps in getting immediate feedback which can be used to improve the software in the next iteration.

Disadvantages of Agile Model

1. Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.
2. Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
3. Not suitable for handling complex dependencies.
4. Depends heavily on customer interaction.

Agile Methodologies
There are several Agile Methodologies that support Agile Development.
The Agile Methodologies include -

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