Introduction to Software Engineering

Software Engineering is the systematic approach to the development, operation and maintenance of software.

The ultimate objective of software engineering is to produce quality software that is maintainable, delivered on time, within budget, and also satisfies its requirements.

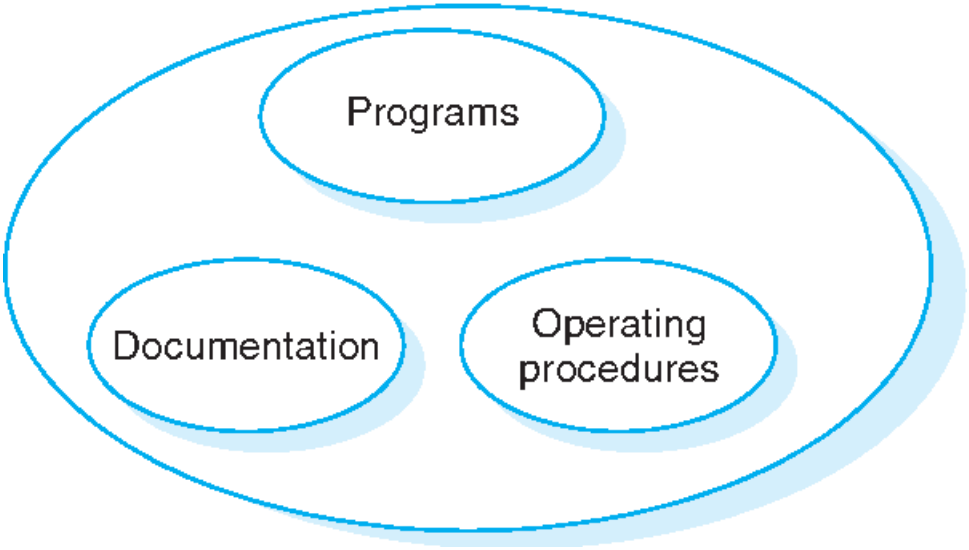
Definition

Fritz Bauer defined software engineering as "The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines".

Stephen Schach defined the same as "A discipline whose aim is the production of quality software software that is delivered on time, within budget, and that satisfies its requirements".

Program Vs Software

Software is more than programs. It consists of programs, documentation of any facet of the program and the procedures used to setup and operate the software system. The components of the software system are shown below.

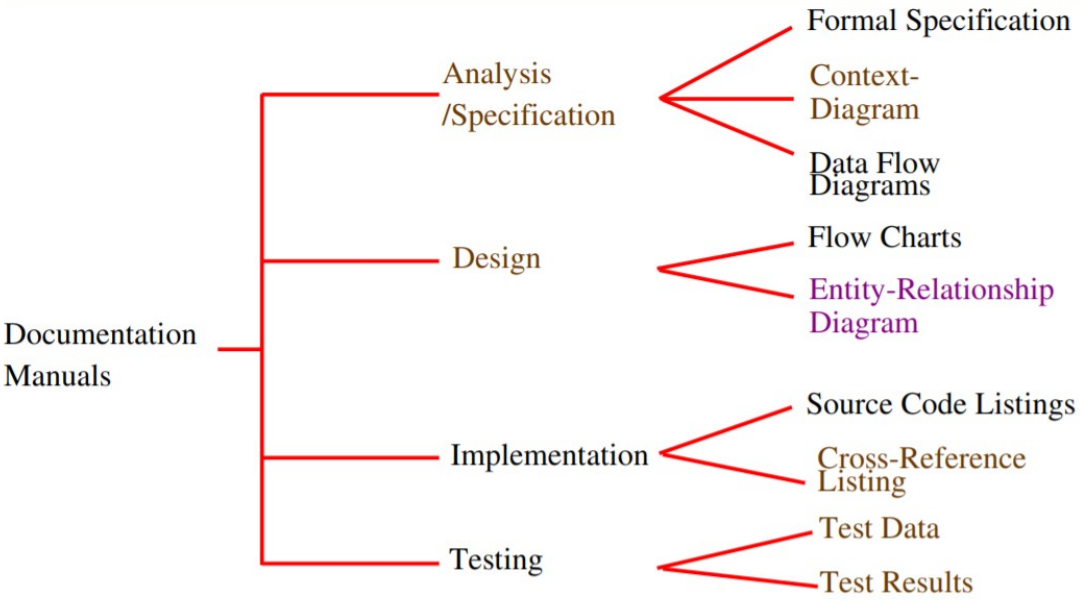


Software= Program + Documentation + Operating Procedures

Fig. 1.1: Components of Software

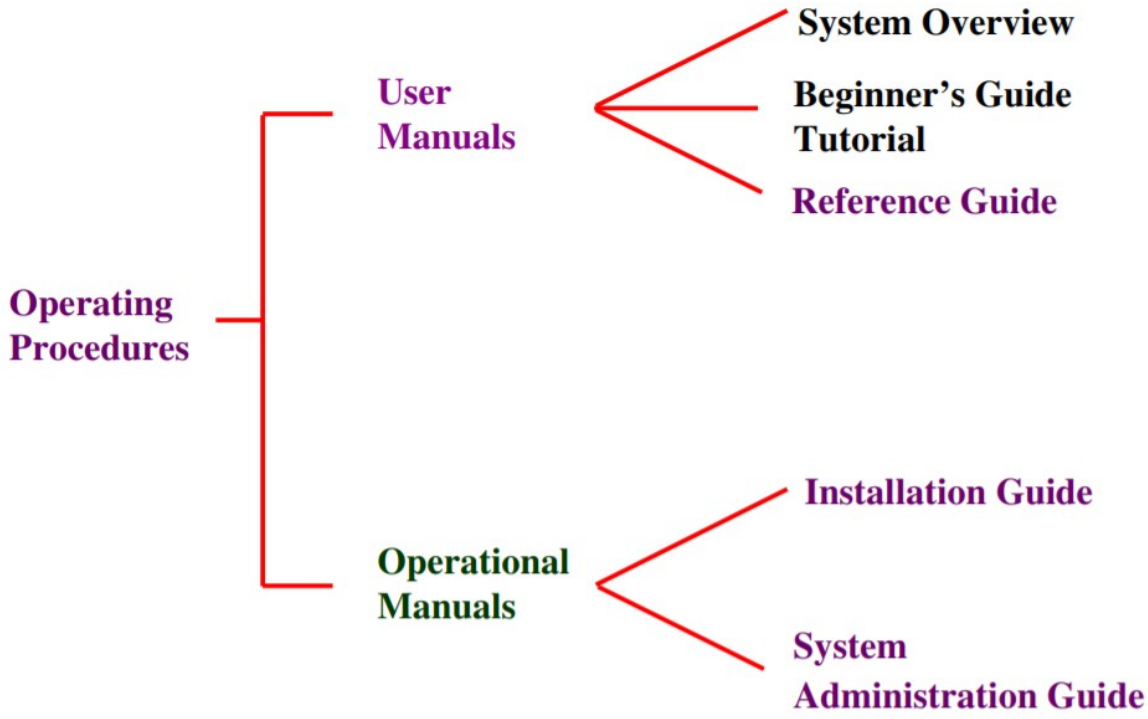
Program is a combination of source code and object code. Documentation consists of different types of manuals are shown below.

Documentation consists of different types of manuals are



List of documentation manuals

Operating procedures consist of instructions to setup and use the software system and instructions on how to react to system failure. List of operating procedure manuals/documents is given below.



List of operating procedure manuals.

Software process

The software process is the way in which we produce software. This differs from organization to organization. Many organizations are looking at software process improvement as a way to improve the quality, productivity and maintenance efforts. But it is difficult to improve software process because of

- Not enough time
- Lack of knowledge
- Wrong motivations
- Insufficient commitment

Not enough time - Unrealistic schedules leave insufficient time to do the essential project work. Customers and senior managers are demanding more software, of higher quality in minimum possible time. So there is always a shortage of time. One consequence is that software organizations may deliver release 1.0 on time, but then they have to ship release 1.01 almost immediately thereafter to fix the recently discovered bugs.

Lack of knowledge - Most of the developers are unaware of best known ways of software development. They may know many languages but do not spent time to improve their knowledge about industry best practices.

Wrong motivations – Software developers should be motivated by the prospect of meeting their commitments, improving customer satisfaction, and delivering excellent products that meet customer expectations. But most organizations launch process improvement initiatives for the wrong reasons. May be a contractor demanded that the development organization should achieve CMM level X by date Y. Or perhaps a senior manager learned just enough about the CMM and directed his organization to climb on the CMM bandwagon.

Insufficient commitment - Many times, process improvement fails due to lack of commitment. Management sets no expectations from the development community around process improvement, they devote insufficient resources, write no improvement plan, develop no roadmap, and pilot no new processes.

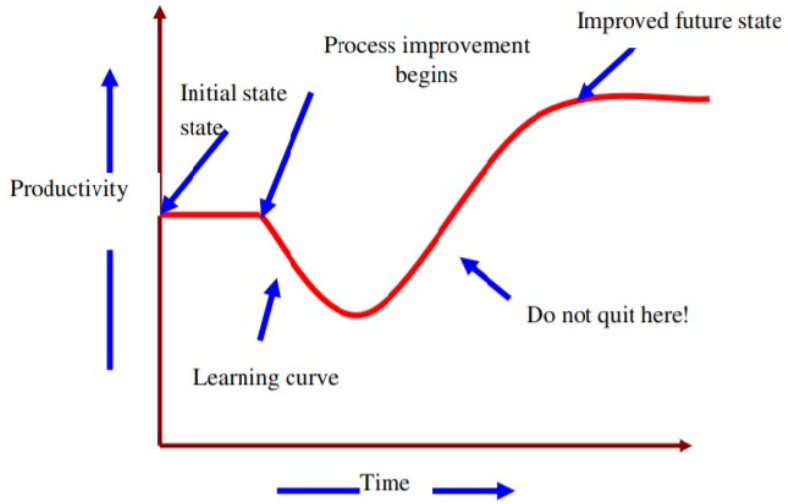


Fig. 1.4: The process improvement learning curve

The investment we make in process improvement will not have an impact on current productivity; because the time we spend developing better ways to work tomorrow is not available for today's assignment. Improvements will take place over time and organizations should not expect and promise miracles and should always remember the learning curve (See Figure 1.4).

Software Characteristics

Software is a logical entity rather than a physical system. So characteristics of software is different from hardware. Some of the important characteristics are:

- Software does not wear out
- Software is not manufactured
- Reusability of Components
- Software is flexible

Software does not wear out: There is a well-known “bath tub curve” in reliability studies for hardware products. The curve is given in Fig. 1.5. The shape of the curve is like “bath tub”; and is known as bath tub curve.

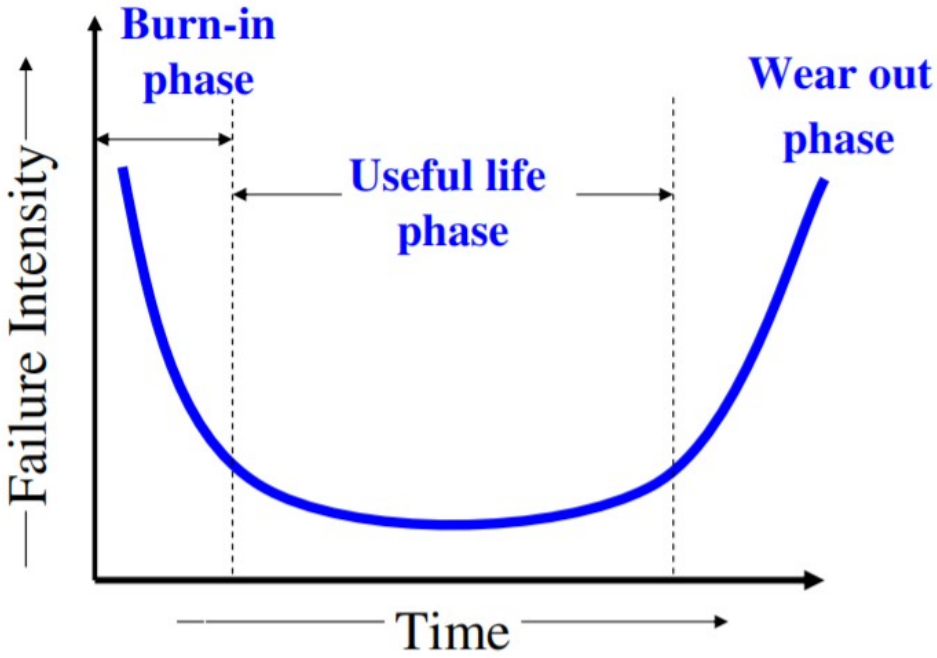


Fig. 1.5: Bath tub curve

There are three phases for the life of a hardware product. Initial phase is burn-in phase, where failure intensity is high. It is expected to test the product in the industry before delivery. Due to testing and fixing faults, failure intensity will come down initially and may stabilise after certain time. The second phase is the useful life phase where failure intensity is approximately constant and is called useful life of a product. After few years, again failure intensity will increase due to wearing out of components. This phase is called wear out phase. We do not have this phase for the software as it does not wear out. The curve for software is given in Fig. 1.6.

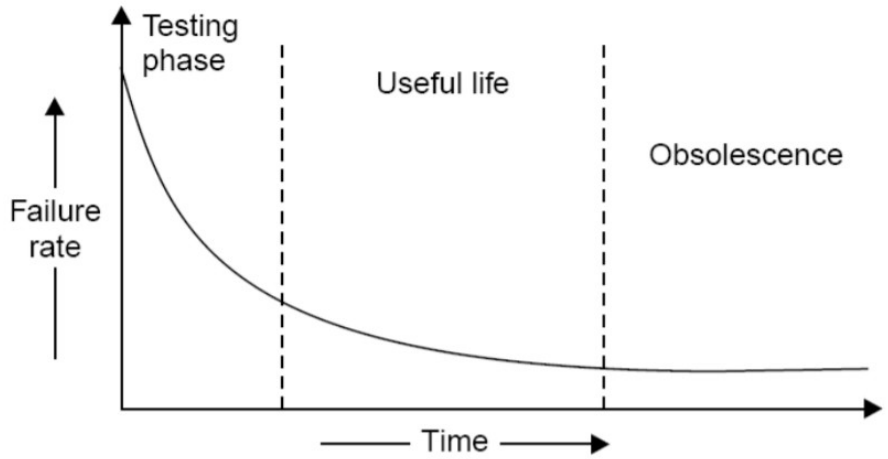


Fig. 1.6: Software curve

Important point is software becomes reliable overtime instead of wearing out. It becomes obsolete, if the environment for which it was developed, changes. Hence software may be retired due to environmental changes, new requirements, new expectations, etc.

(ii) **Software is not manufactured:** The life of a software is from concept exploration to the retirement of the software product. It is one time development effort and continuous maintenance effort in order to keep it operational. However, making 1000 copies is not an issue and it does not involve any cost. In case of hardware product, every product costs us due to raw material and other processing expenses. We do not have assembly line in software development. Hence it is not manufactured in the classical sense.

(iii) **Reusability of components:** If we have to manufacture a TV, we may purchase picture tube from one vendor, cabinet from another, design card from third and other electronic components from fourth vendor. We will assemble every part and test the product thoroughly to produce a good quality TV. We may be required to manufacture only a few components or no component at all. We purchase every unit and component from the market and produce the finished product. We may have standard quality guidelines and effective processes to produce a good quality product.

In software, every project is a new project. We start from the scratch and design every unit of the software product. Huge effort is required to develop a software which further increases the cost of the software product. However, effort has been made to design standard components that may be used in new projects. Software reusability has introduced another area and is known as component based software engineering. Eg. pull-down menus, graphics windows used in GUI.

(iv) **Software is Flexible:** Software can be developed to do almost anything. Sometimes, this characteristic helps us to accommodate changes but most of the times, this makes software development difficult to plan, monitor and control.

Deliverables and Milestones

Different deliverables are generated during software development. The examples are source code, user manuals, operating procedure manuals etc.

The milestones are the events that are used to ascertain the status of the project. Finalisation of specification is a milestone. Completion of design documentation is another milestone. The milestones are essential for project planning and management.

Product and Process

What is delivered to the customer, is called a product. It may include source code, specification document, manuals, documentation etc. Basically, it is nothing but a set of deliverables only.

Process: Process is the way in which we produce software. It is the collection of activities that leads to (a part of) a product. An efficient process is required to produce good quality products.

Software Process and Product Metrics

Software metrics are used to quantitatively characterise different aspects of software process or software products. Process Metrics quantify the attributes of software development process and environment. Examples of process metrics include productivity, quality, failure rate, efficiency etc. Product Metrics measures the characteristics of software product. Examples of product metrics are size, reliability, complexity, functionality etc.

Software life cycle models

Life cycle of the software starts from concept exploration and ends at the retirement of the software.

IEEE definition of software life cycle is: "The period of time that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes a requirement phase, design phase, implementation phase, test phase, Installation and check out phase, operation and maintenance phase, and sometimes retirement phase".

A software life cycle model is a particular abstraction that represents a software life cycle. A software life cycle model is often called a software development life cycle (SDLC).

Some of the well known models are:

- Waterfall model
- Increment process models
    - Iterative Enhancement Model
    - RAD Model
- Evolutionary process models
    - Prototyping model
    - Spiral model

Waterfall Model

This is the simplest model and this model is named "waterfall model" because its diagrammatic representation resembles a cascade of waterfalls. This model has five phases: requirement analysis and specification, design, implementation and unit testing, integration and system testing and operation and maintenance. The phases always occur in this order and do not overlap. The developer must complete each phase before the next phase begins.

Requirement analysis and specification phase: - The goal of this phase is to understand the exact requirements of the customer and to document them properly. This activity is usually executed together with the customer, as the goal is to document all functions, performance and interfacing requirements for the software. The requirements describe the 'what' of a system not the 'how'. This phase produces a large document, written in a natural language, contains description of what the system will do without describing how it will be done. The resultant document is known as software requirement specification (SRS).

SRS acts as a - contract between the developer and customer, reference document for designers, reference document for validation.

Design phase: - The goal of this phase is to transform the requirements specification into a structure that is suitable for implementation in some programming language. Here, overall software architecture is defined, and high level and detailed design work is performed. This work is documented and known as software design description (SDD).

Implementation and unit testing phase:- During this phase design is implemented using the information in SDD. During unit testing, small modules are tested in isolation from the rest of the product. The problems associated with unit testing are solved by writing some overhead code.

Integration and system testing phase:- Integration testing mainly tests interface between modules and system testing tests the entire system. Effective testing will contribute to the delivery of higher quality software, more satisfied users, lower maintenance costs and more accurate and reliable results. It is a very expensive activity and consumes one-third to one half of the cost of a development project.

Operation and maintenance phase:- Release of software inaugurates the operation and maintenance phase. The time spent and effort required to keep the software operational after release is very important. Maintenance is a broad activity that includes error correction, enhancement of capabilities and optimization.

Merits - This model is simple and and easy to understand and reinforces the notion of "define before design" and "design before code".

Demerits

1. The model expects complete & accurate requirements early in the process, which is unrealistic.
2. This model is not suitable for accommodating any change.
3. A working version of the system is not seen until late in the project's life.

4. It does not scale up well to large projects.
5. Real projects are rarely sequential.

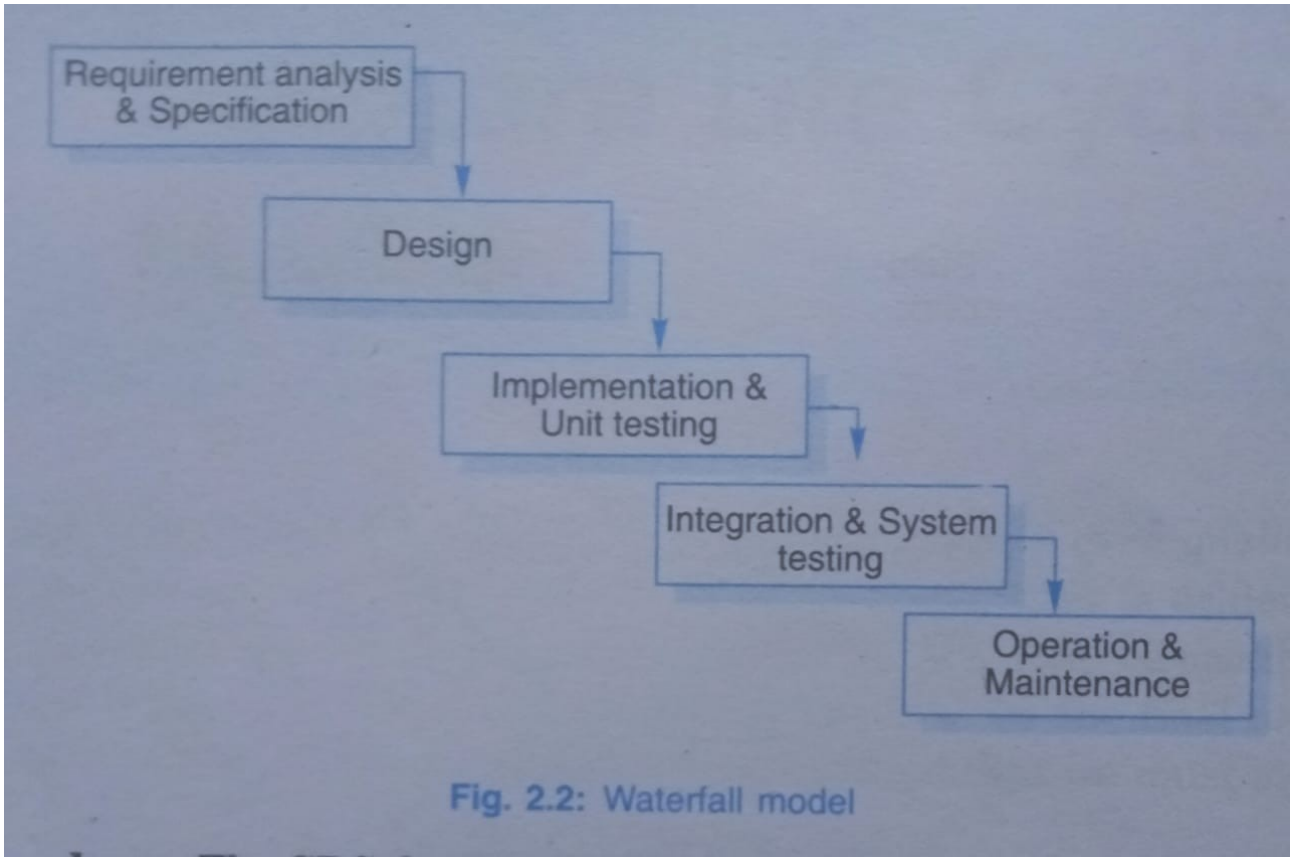


Fig. 2.2: Waterfall model

INCREMENT PROCESS MODELS

Increment process models are effective in the situations where requirements are defined precisely and there is no confusion about the functionality of the final product. After every cycle, a useable product is given to the customer. For example, in the university automation software library automation module may be delivered in the first phase and examination automation module in the second phase and as so on. Every new cycle will have an additional functionality. Increment process models are popular particularly when we have to quickly deliver a limited functionality system.

Iterative Enhancement Model

This model has the same phases as the waterfall model, but with fewer restrictions. Generally the phases occur in the same order as in the waterfall model, but these may be conducted in several cycles. A useable product is released at the end of the each cycle, with each release providing additional functionality.

During the first requirements analysis phase, customers and developers specify as many requirements as possible and prepare a SRS document. Developers and customers then prioritize these requirements. Developers implement the specified requirements in one or more cycles of design, implementation and test based on the defined priorities. The model is given in Fig. 2.3.

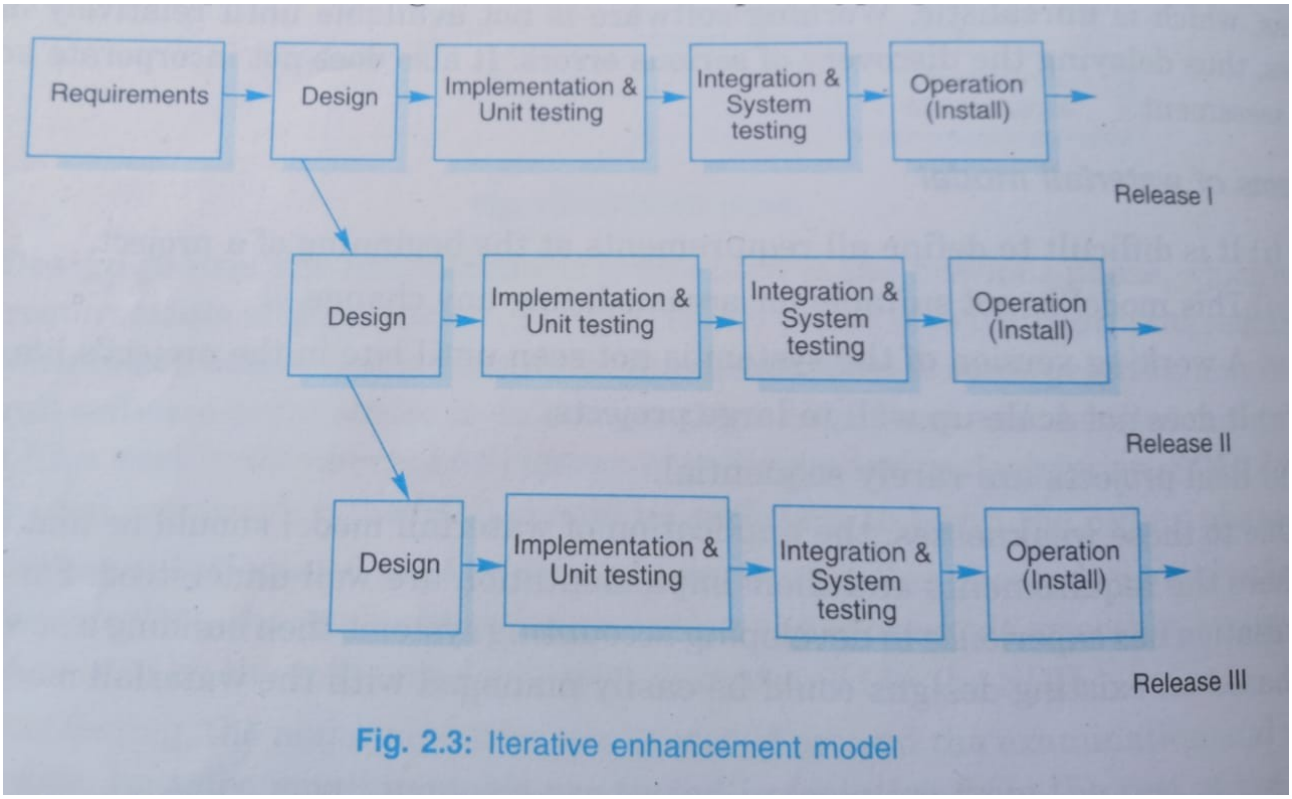


Fig. 2.3: Iterative enhancement model

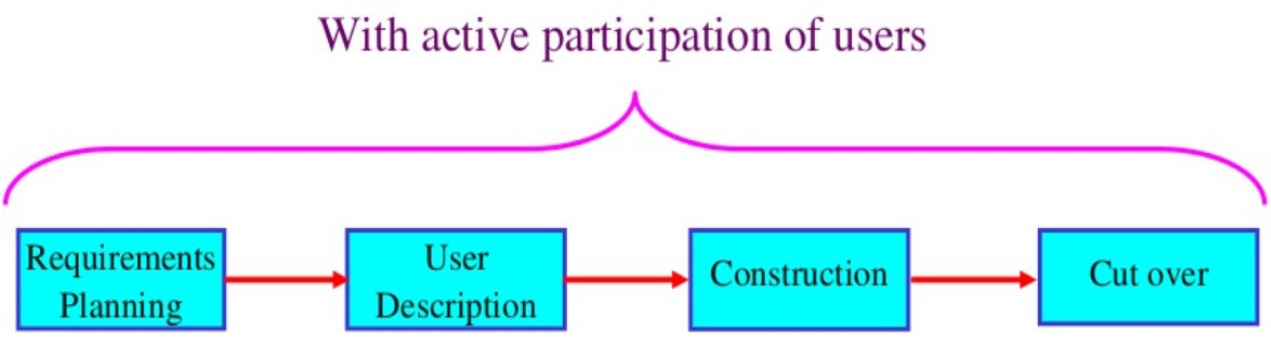
The complete product is divided into releases, and the developer delivers the product release by release. At each release, customer has an operational quality product that does a portion of what is required. With this model, first release may be available within few weeks or months, whereas the customer generally waits months or years to receive a product using the waterfall and prototyping model.

The Rapid Application Development (RAD) Model

This model is an incremental process model and was developed by IBM. Unique features of this model are:

- Continuous user involvement from requirement to delivery of the product.
- Maximum use of tools.

The process is started with building a rapid prototype and is given to user for evaluation. The user feedback is obtained and prototype is refined. The process continues, till the requirements are finalized. There are four phases in this model and these are shown in fig below.



Requirements Planning phase: Requirements are collected using any elicitation technique.

User Description: Joint team of developers and users are constituted to prepare, understand and review the requirements. The team may use automated tools to capture information from the other users.

Construction phase: This phase combines the detailed design, coding and testing phase of waterfall model. Here, we release the product to customer. It is expected to use code generators, screen generators, and other types of productivity tools.

Cut over phase: This phase incorporates acceptance testing by the users, installation of the system and user training.

Merits:

- Quick initial views about the product are possible due to delivery of rapid prototype.
- The development time is reduced due to the use of powerful tools.
- Use of CASE tools increases productivity.
- Involvement of users may increase the acceptability of the product.

Demerits:

- Not an appropriate model in the absence of user participation.
- Reusable components are required to reduce development time.
- Highly specialized & skilled developers are required and such developers are not easily available.

Evolutionary process models

Evolutionary process model resembles iterative enhancement model. The same phases as defined for the waterfall model occur here in a cyclical fashion. This model differs from iterative enhancement model in the sense that this does not require a useable product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

For eg, in a simple database application, one cycle might implement the graphical user interface: another file manipulation; another queries; and another updates. All four cycles must complete before there is working product available. GUI allow users to interact with the system; file manipulation allows data to be saved and retrieved; queries allow users to get data out of the system: and updates allow users to put data into the system. In contrast, an iterative enhancement model would start by developing a very simplistic, but usable database.

This model is useful for projects using new technology that is not well understood. This is also used for complex projects where all functionality must be delivered at one time, but the requirements are unstable or not well understood at the beginning.

Prototyping Model

In this model, a working prototype is developed first instead of developing actual software. It is developed as per current available requirements in minimum time. It has limited functional capabilities, low reliability and quality. Developers use this prototype to refine requirements and prepare final SRS. Because the working prototype has been evaluated by the customer, it is reasonable to expect that SRS will be correct. Moreover, the experience gathered helps in developing the actual system. The development of a prototype might involve extra cost, but overall cost might turnout to be lower than that of an equivalent system developed using the waterfall model.

The prototype may be a usable program but is not suitable as the final software product. The code for the prototype is thrown away after determining customer's real needs and actual system is developed using waterfall approach. Thus it is used as an input to waterfall model and produce maintainable and good quality software.

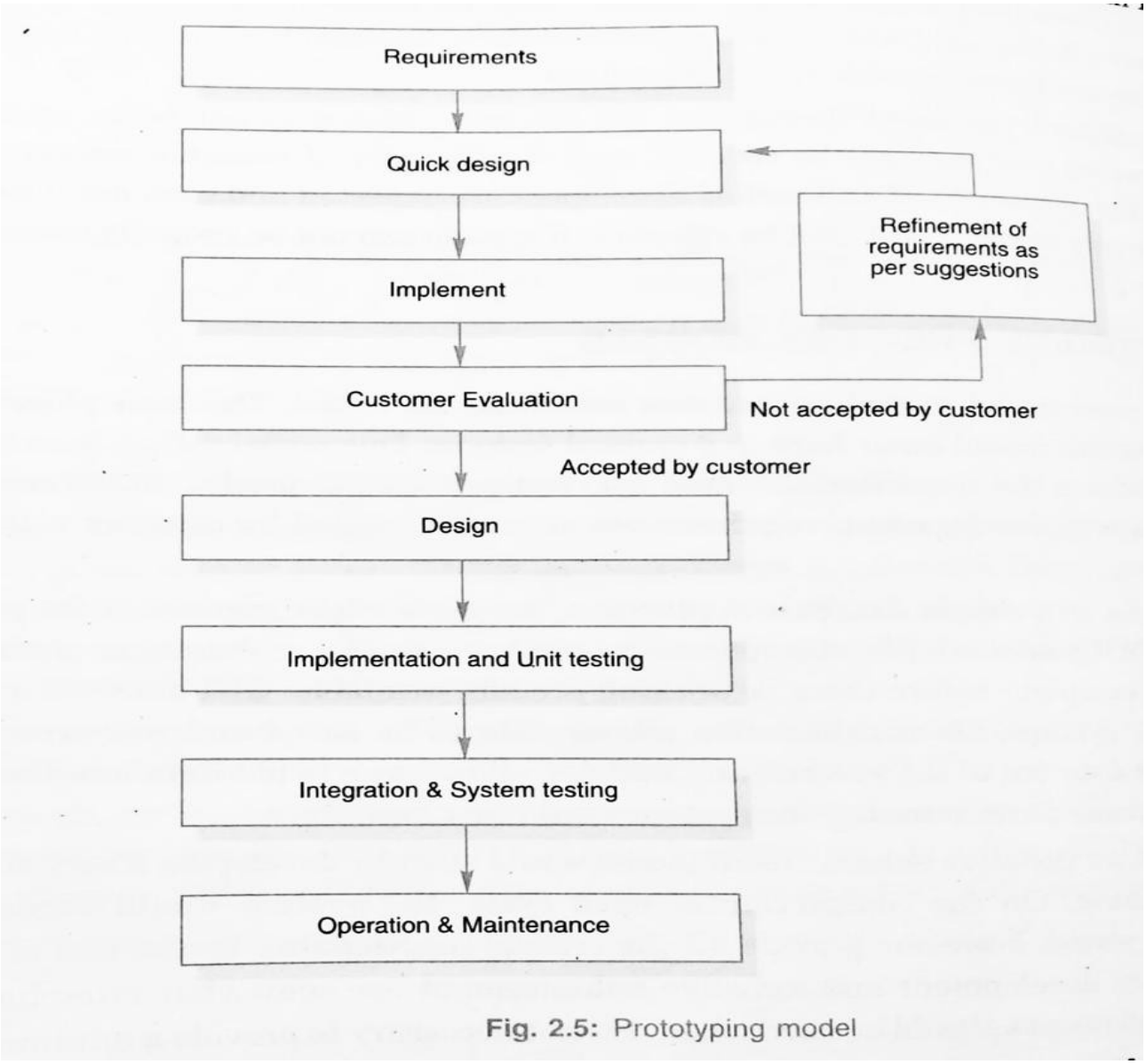


Fig. 2.5: Prototyping model

Spiral Model

Traditional process models do not deal with uncertainty which is inherent to software projects. Important software projects have failed because project risks were neglected & nobody was prepared when something unforeseen happened. Barry Boehm recognized this and tried to incorporate the “project risk” factor into a life cycle model. The result is the spiral model, which was presented in 1986.

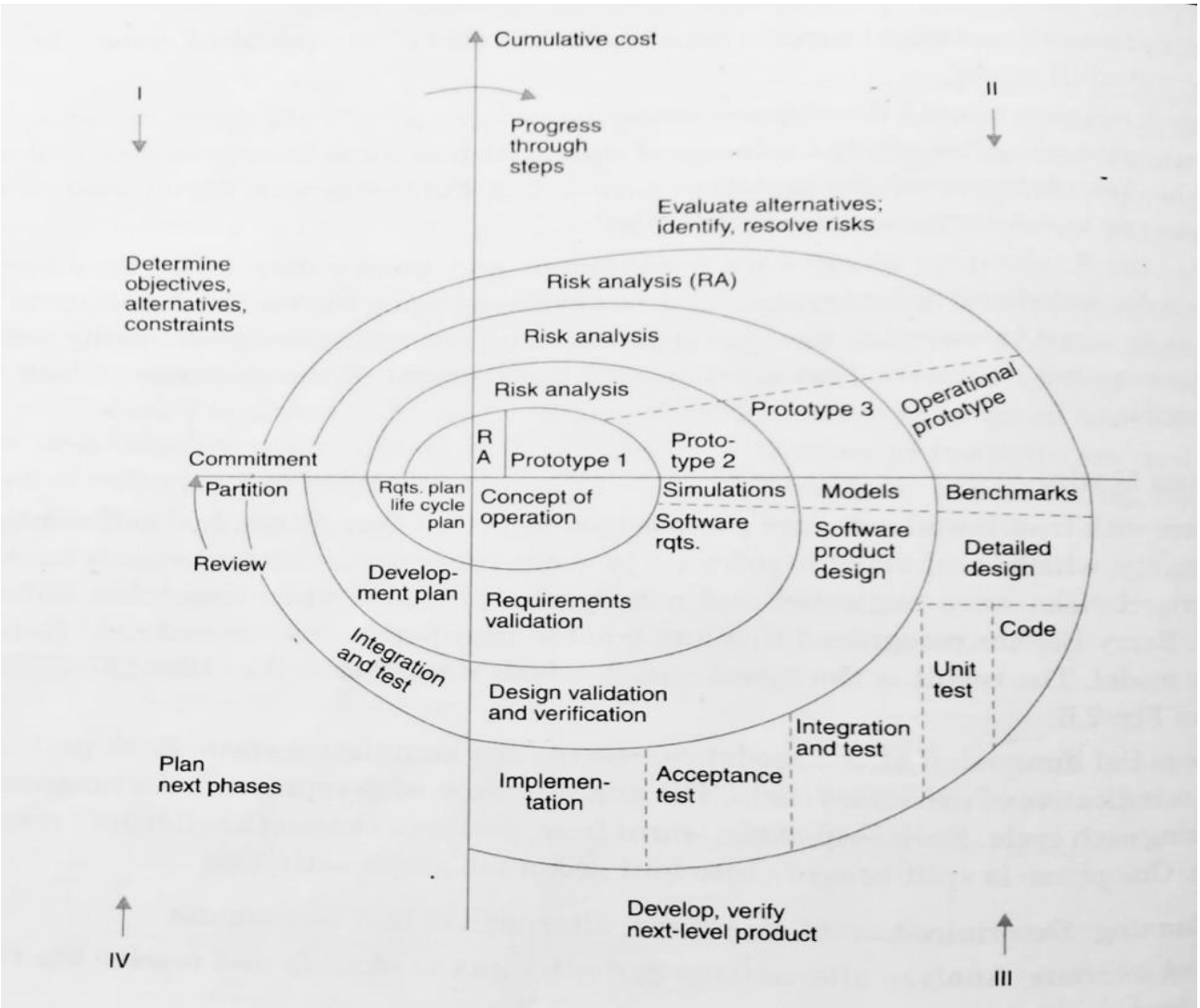


Fig. 2.6: Spiral model

The radial dimension of the model represents the cumulative costs. Each path around the spiral is indicative of increased costs. The angular dimension represents the progress made in completing each cycle. Each loop of the spiral from X-axis clockwise through 360° represents one phase. One phase is split roughly into four sectors of major activities:

Planning: Determination of objectives, alternatives and constraints

Risk analysis: Analyze alternatives and attempts to identify and resolve the risks involved

Development: Product development and testing product

Assessment: Customer evaluation

During the first phase, planning is performed, risks are analyzed, prototypes are built, and customers evaluate the prototype. During the second phase, a more refined prototype is built, requirements are documented and validated, and customers are involved in assessing the new

prototype. By the time the third phase begins, risks are known, and a somewhat more traditional development approach is taken.

An important feature of the spiral model is that each phase is completed with a review by the people concerned with the project (designers and programmers). This review consists of a review of all the products developed up to that point and includes the plans for the next cycle.

Merits:

- The main advantage of this model is the wide range of options to accommodate the good features of other life cycle models.
- It becomes equivalent to another life cycle model in appropriate situations.
- It also incorporates quality goals into software development.
- The risk analysis and validation steps eliminate errors in the early phases of development.
- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

Demerits:

The spiral model has some difficulties that need to be resolved before it can be a universally applied life cycle model. These difficulties include,

- Lack of explicit process guidance in determining objectives, constraints, alternatives;
- Requires highly specific expertise in risk analysis.
- provides more flexibility than required for many applications.
- Not suitable for small or low risk projects and could be expensive for small projects.

SELECTION OF A LIFE CYCLE MODEL

The selection of a suitable model is based on the following characteristics/categories:

- Requirements
- Development team
- Users
- Project type and associated risk.

Characteristics of Requirements

Requirements are very important for the selection of an appropriate model. There are number of situations and problems during requirements capturing and analysis. The details are given in Table 2.1.

Based On Characteristics Of Requirements

Requirements	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Are requirements easily understandable and defined?	Yes	No	No	No	No	Yes
Do we change requirements quite often?	No	Yes	No	No	Yes	No
Can we define requirements early in the cycle?	Yes	No	Yes	Yes	No	Yes
Requirements are indicating a complex system to be built	No	Yes	Yes	Yes	Yes	No

Table 2.1

Status of Development Team

The status of the development team in terms of availability, effectiveness, knowledge, intelligence, team work etc., is very important for the success of the project. If we know the above mentioned parameters and characteristics of the team, then we may choose an appropriate life cycle model for the project. Some of the details are given in Table 2.2.

Based On Status Of Development Team

Development team	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Less experience on similar projects?	No	Yes	No	No	Yes	No
Less domain knowledge (new to the technology)	Yes	No	Yes	Yes	Yes	No
Less experience on tools to be used	Yes	No	No	No	Yes	No
Availability of training if required	No	No	Yes	Yes	No	Yes

Table 2.2

Involvement of Users

Involvement of users increases the understandability of the project. Hence user participation, if available, plays a very significant role in the selection of an appropriate life cycle model. Some issues are discussed in Table 2.3.

Based On User's Participation

Involvement of Users	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
User involvement in all phases	No	Yes	No	No	No	Yes
Limited user participation	Yes	No	Yes	Yes	Yes	No
User have no previous experience of participation in similar projects	No	Yes	Yes	Yes	Yes	No
Users are experts of problem domain	No	Yes	Yes	Yes	No	Yes

Table 2.3

Type of Project and Associated Risk

Very few models incorporate risk assessment. Project type is also important for the selection of a model. Some issues are discussed in Table 2.4.

Based On Type Of Project With Associated Risk

Project type and risk	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Project is the enhancement of the existing system	No	No	Yes	Yes	No	Yes
Funding is stable for the project	Yes	Yes	No	No	No	Yes
High reliability requirements	No	No	Yes	Yes	Yes	No
Tight project schedule	No	Yes	Yes	Yes	Yes	Yes
Use of reusable components	No	Yes	No	No	Yes	Yes
Are resources (time, money, people etc.) scare?	No	Yes	No	No	Yes	No

Table 2.4

An appropriate model may be selected based on options given in four Tables .Firstly, we have to answer the questions presented for each category by circling a yes or no in each table. Rank the importance of each category in terms of the project for which we want to select a model. The total number of circled responses for each column in the tables decide an appropriate model. We may also use the category ranking to resolve the conflicts between models if the total in either case is close or the same.