Software Engineering

Semester 6

Unit 1: Introduction

Introduction to Software Engineering

Last 40 years have been experiencing rapid changes in the nature and complexity of SOFTWARE.

Earlier

- Applications ran on single processor
- Produced alphanumeric outputs

•Received inputs from a linear/ single source

Today

- GUI's
- Client Server Architecture
- Multi processors, different operating systems, distributed computing and many more such complexities have evolved.

Key Challenges faced by Industry

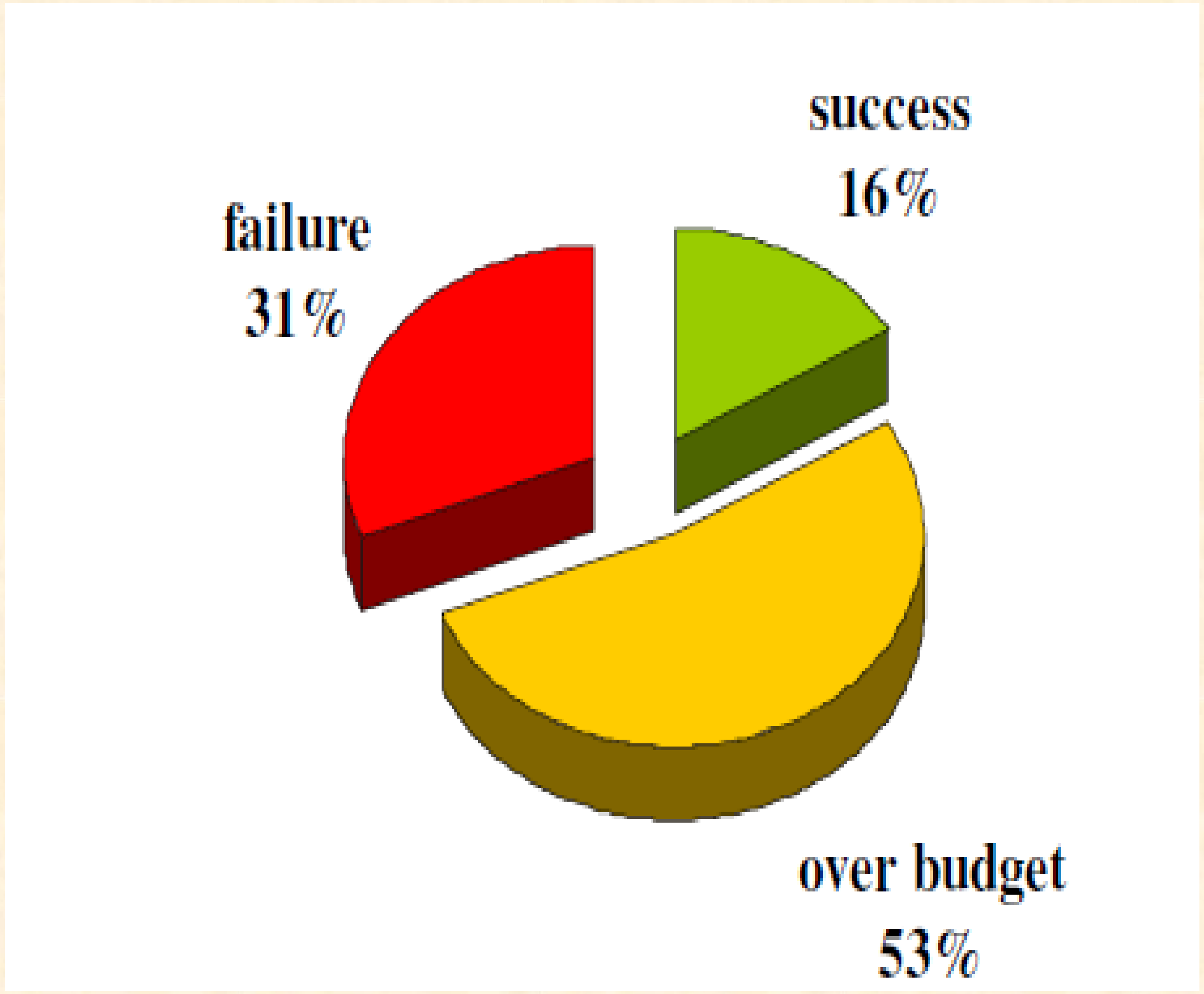
- Struggle to stay abreast with technological developments
- Dealing with accumulated development backlogs
- Coping up with issues related to stakeholders.

Signs of Evolution

- Concept of one ***GURU*** for a Project has vanished
- There has been an acceptance on ***need for change*** in existing processes

Need to adapt new Software Engineering Concepts, Dynamic strategies, practices to avoid conflict, largely to Improve Software Development Process to generate Quality Software on Time and in Budget

Software Crisis



An IBM Report says that

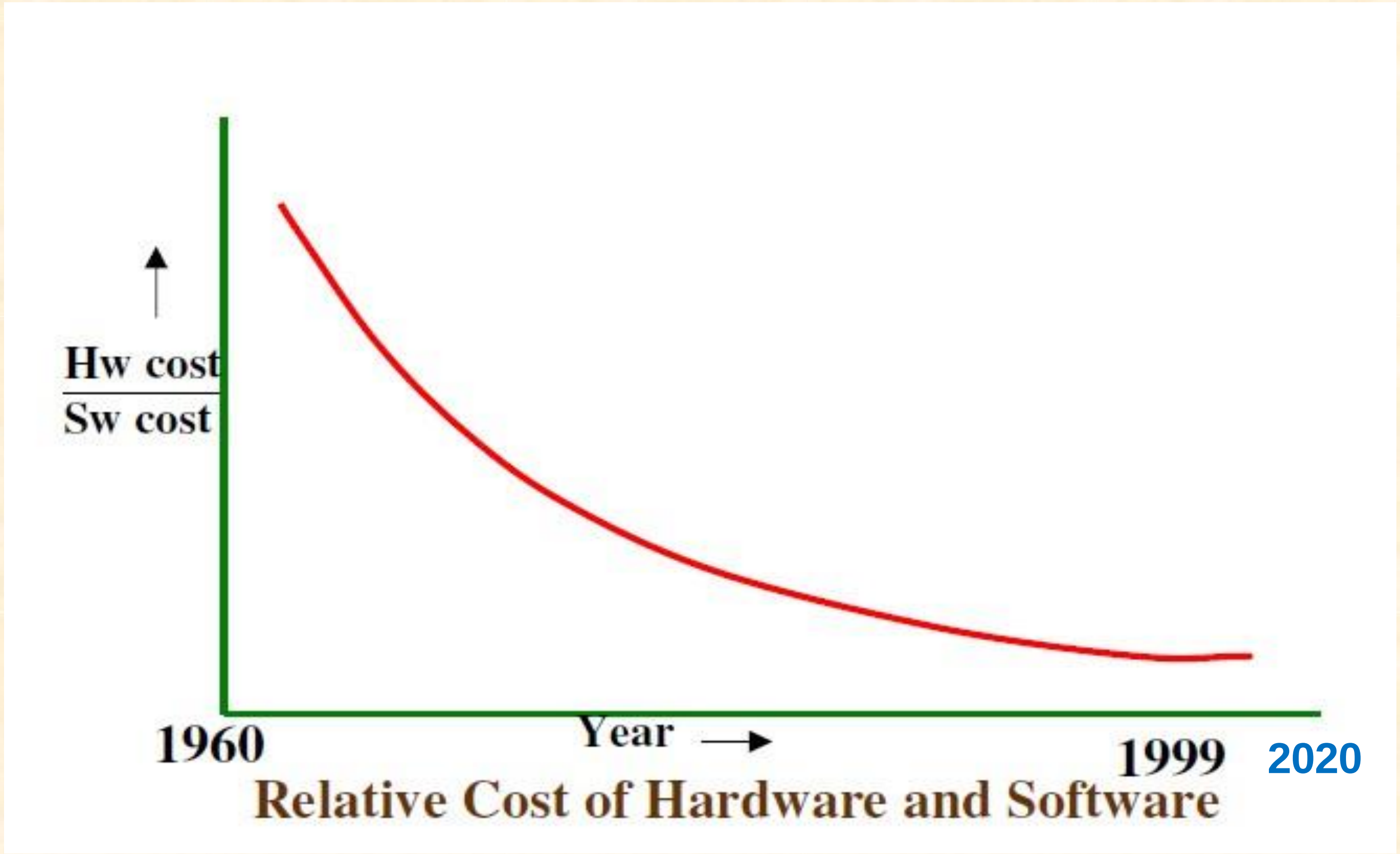
- 31% of projects are cancelled before they are completed.
- 53% are over budget by an average 189%.
- It implies there is a success of just 16% of all human efforts in this industry.

A few Software Failures in Past

- **Y2K Problem** (Year 2000)
- **Patriot Missiles** (Gulf War 1990, to hit Iraqi Scud Missiles)
- 1996 **US Consumer Database Failure** (development team agreed to all requirement, compromised to bugs for delivering on deadline)
- **British Airways "Technical Issue" (A**n IT system issue in 2019 delayed hundreds of flights in the UK, while dozens of flights were canceled completely. This failure affected three British airports and thousands of passengers who had to rebook their flights or check-in by using manual systems)
- **Facebook User Data Leak (2018 September):** *Even the data of facebook CEO was compromised by inherent vulnerabilities in code*

The list is endless…….

Relative Cost of Hardware and Software over time

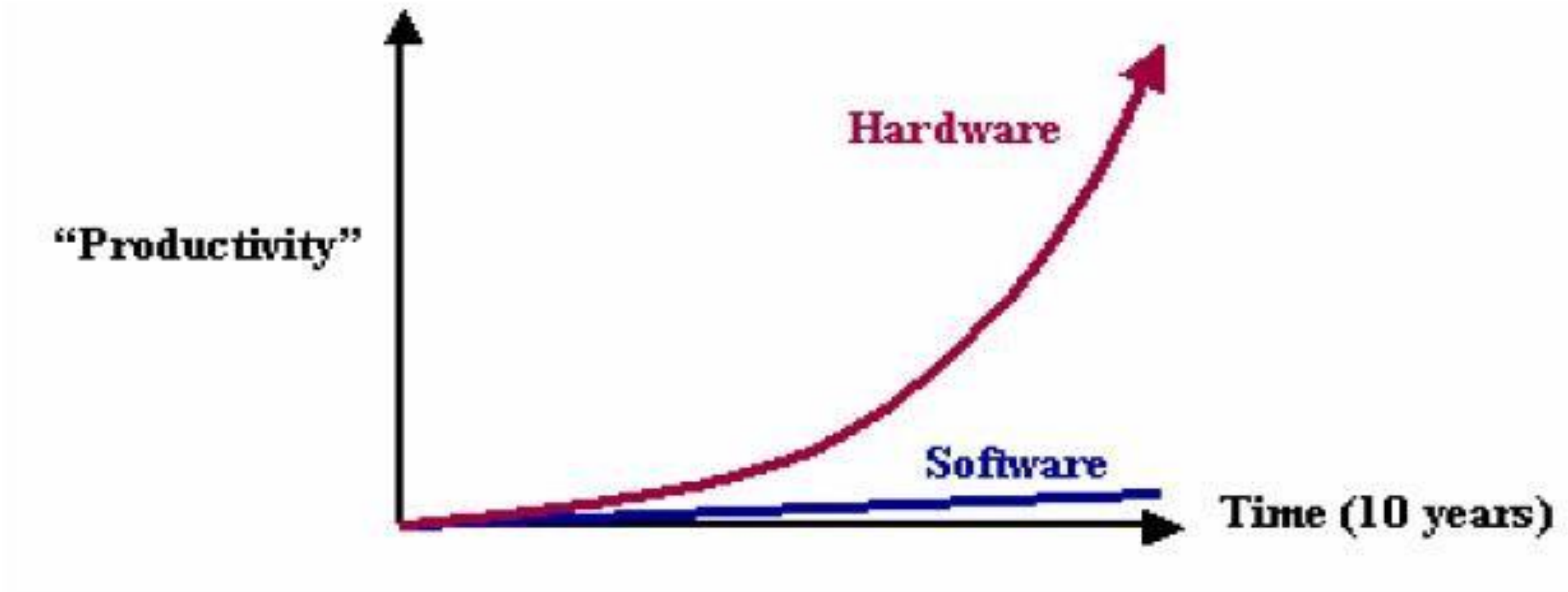


Relative Cost of Hardware and Software

Costs of Hardware and Software have gone significantly down over years

Hardware Vs Software

- Unlike Hardware
  - Moore's law: processor speed/memory capacity doubles every two years



Managers and Technical Persons are asked:

- Why does it take so long to get the program finished?
- Why are costs so high?
- Why can not we find all errors before release?
- Why do we have difficulty in measuring progress of software development?

Factors Contributing to the Software Crisis

- Larger problems
- Lack of adequate training in software engineering
- Increasing skill shortage
- Low productivity improvements
- Time Lines
- Communicational issues
- Intangibility
- Slow execution of software than expected
- Maintainability was difficult
- Increased actual cost of development than projected
- Low Traceability of customer identified errors

“No Silver Bullet”

- The **hardware cost** continues to decline drastically.
- However, there are desperate cries for a **Silver Bullet** something to make software costs drop as rapidly as computer hardware costs do.
- But as we look to the horizon of a decade, we see no silver bullet.
- There is no single development, either in technology or in management technique, that by itself promises even one order of magnitude improvement in productivity, in reliability and in simplicity.

“No Silver Bullet”

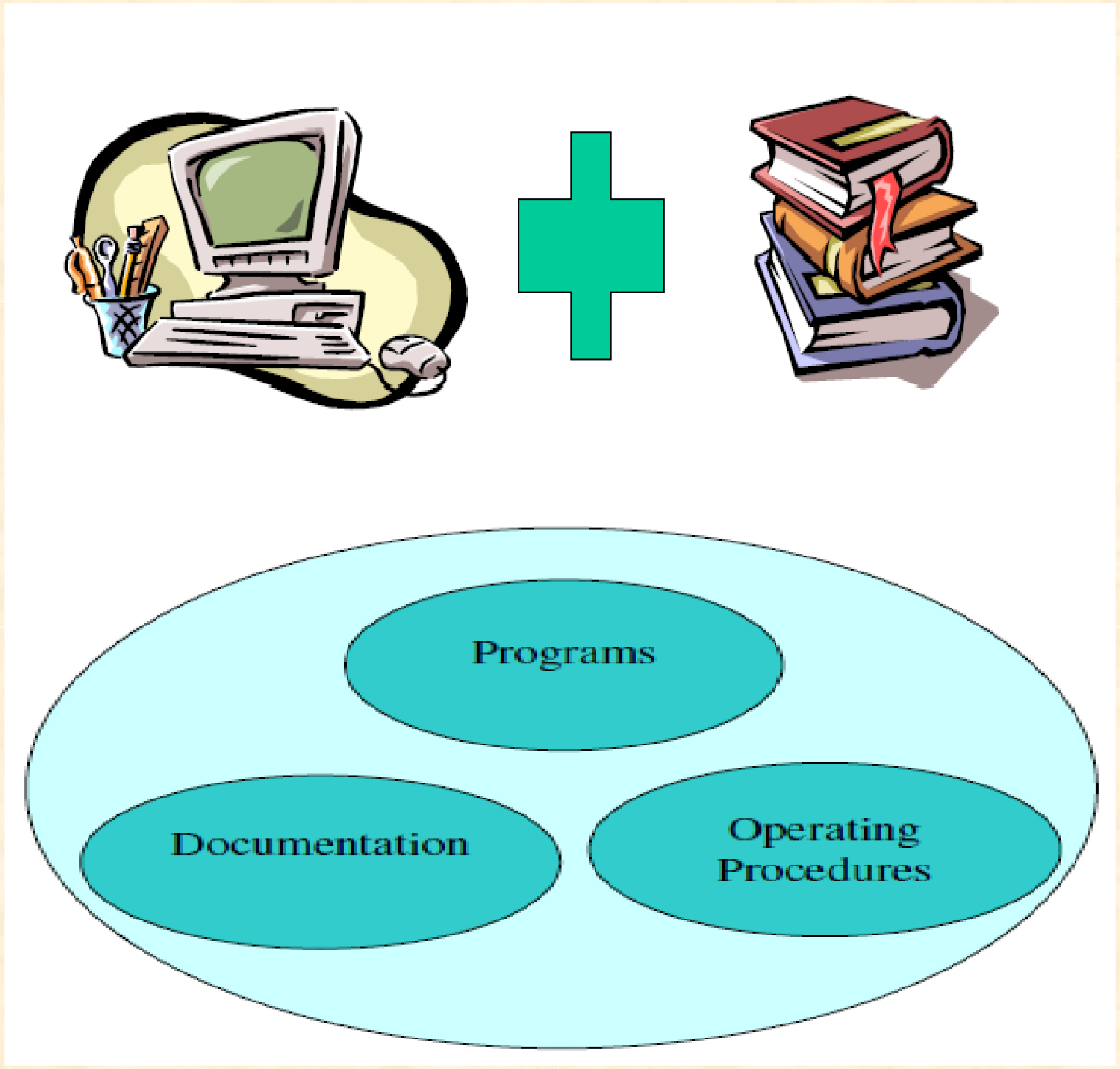
- The hard part of building software is the specification, design and testing of this conceptual construct, not the labour of representing it and testing the correctness of representation.
- While there is no royal road, there is a path forward.
- Is **reusability (and open source)** the new silver bullet?
- We still make **syntax errors**, to be sure, but they are **trivial as compared to the conceptual errors** (logic errors) in most systems.
- That is why, building software is always hard and there is inherently no silver bullet.

The Bug Blame Game

The blame for software bugs belongs to:

- Software companies
- Software developers
- Legal system
- Universities

What is a Software????

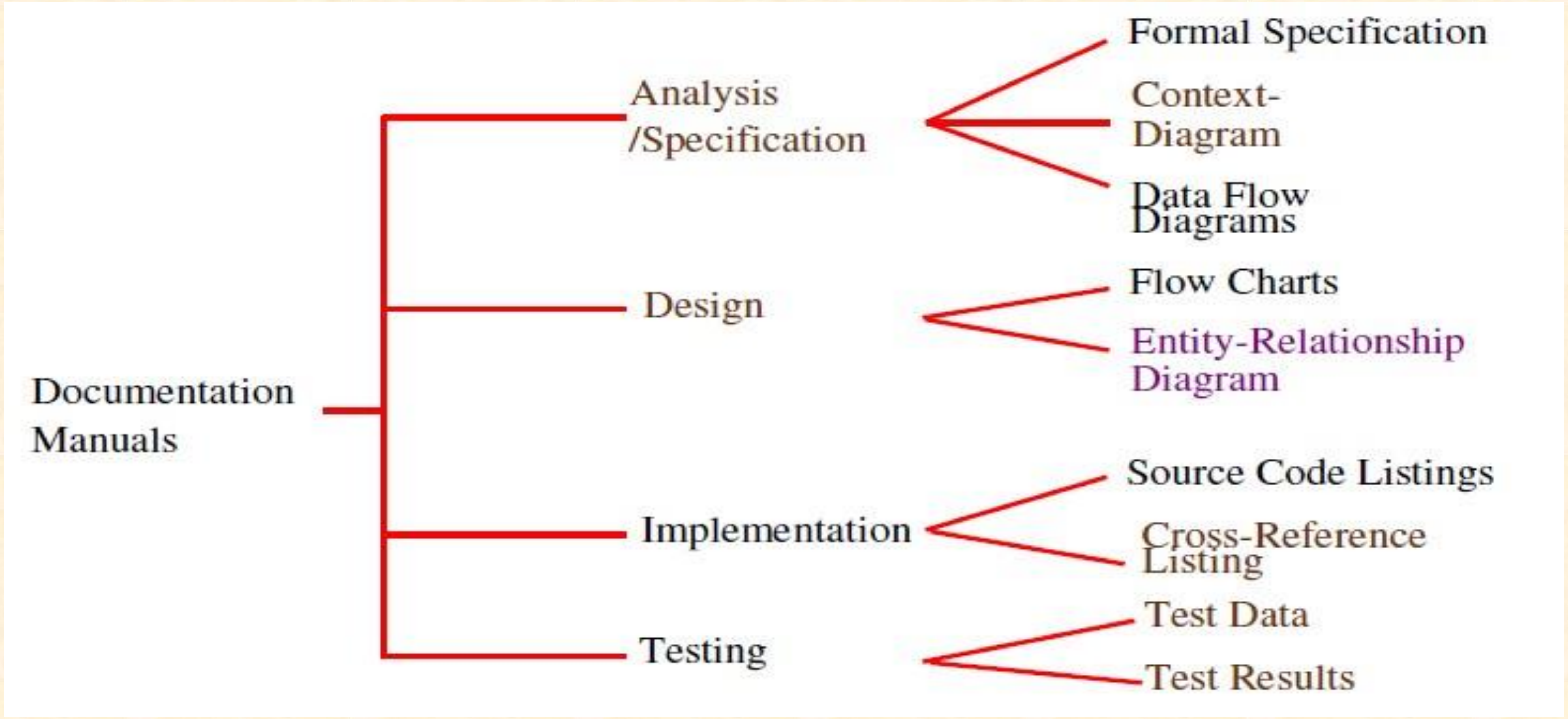


A Computer Program with all Relevant Documentation

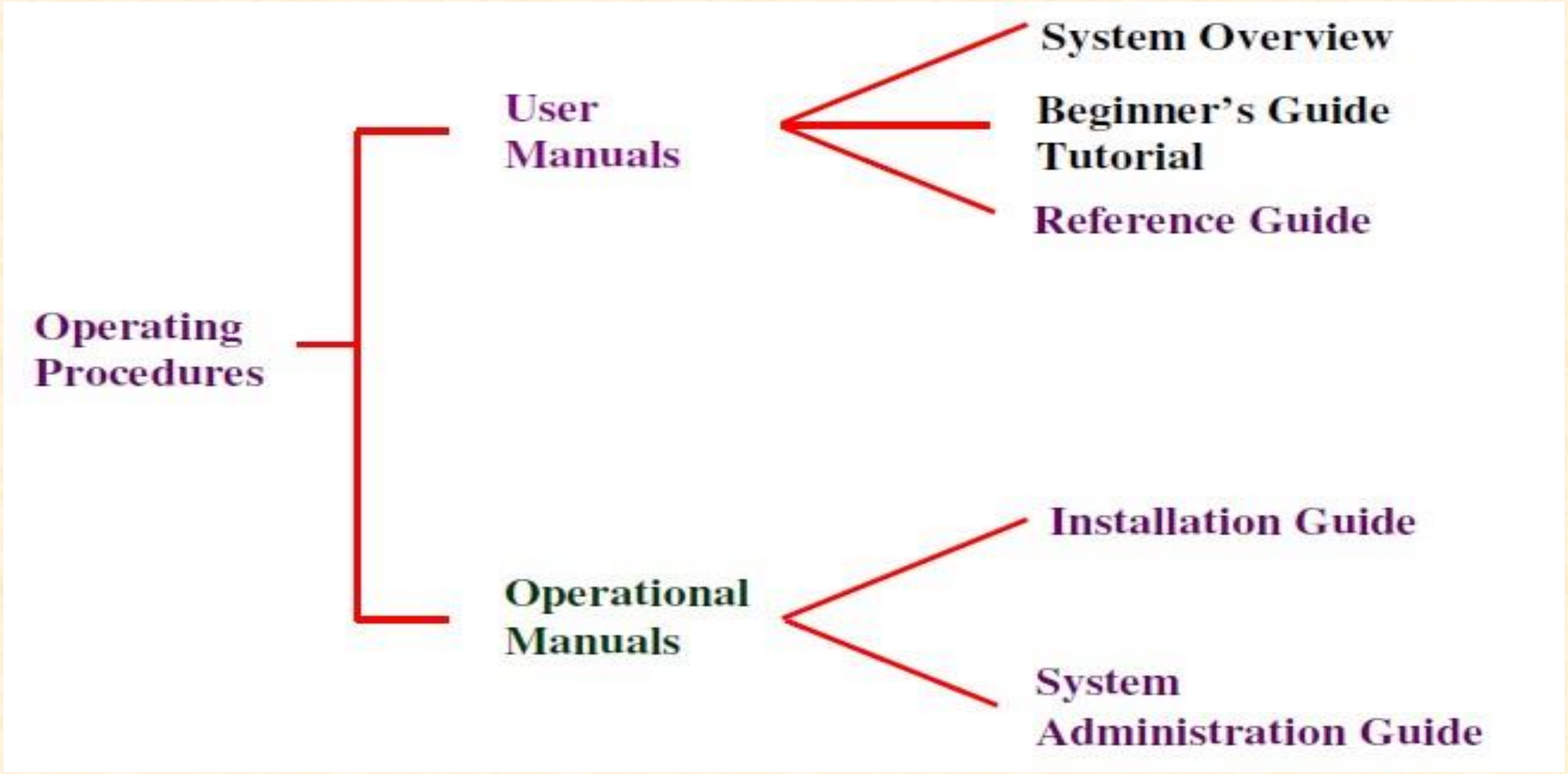
Or

Software = Program+ Documentation+ Operating Procedures

Software Documentation



Operating Procedures



Software Product

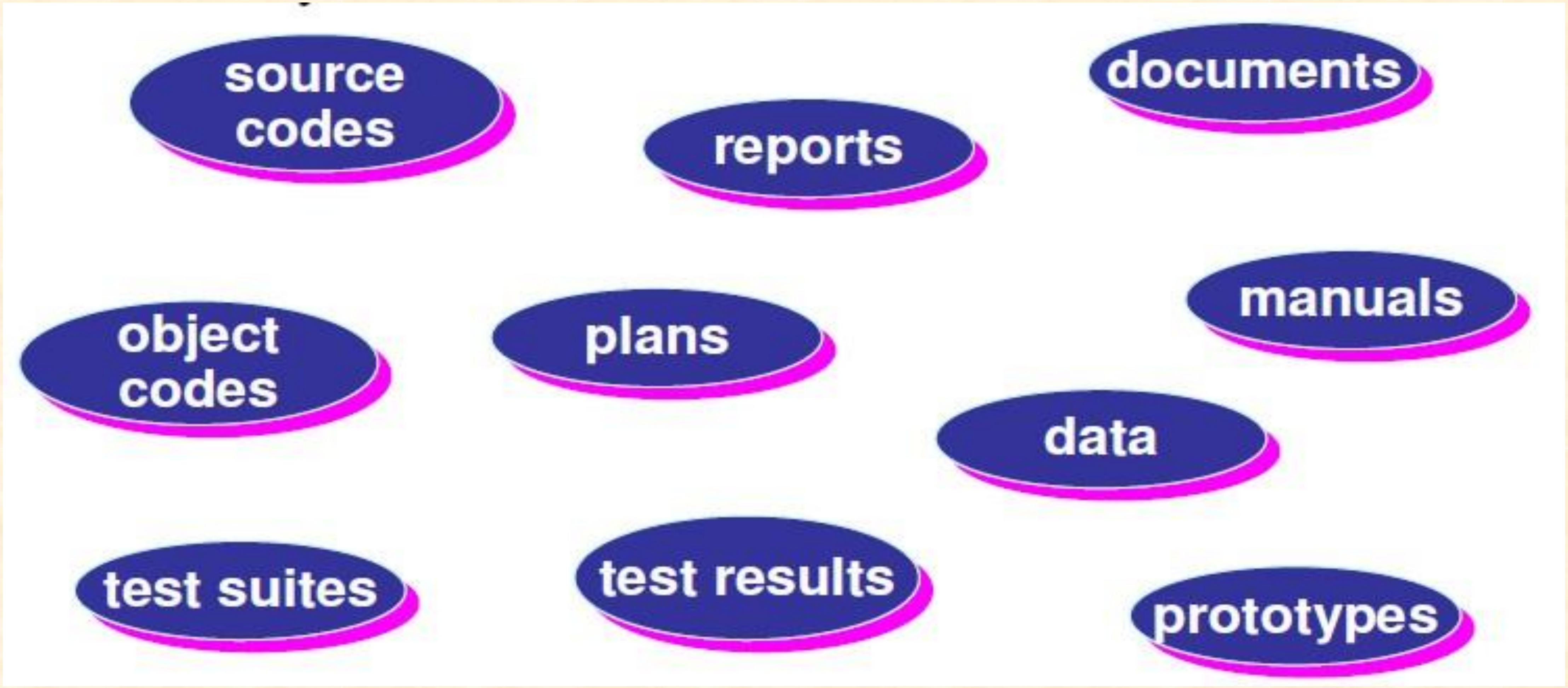
Software products may be developed for a particular customer or may be developed for a general market

- **Software products** may be

  Generic - developed to be sold to a range of different customers

  Bespoke (Custom) - developed for a single customer according to their specification

Software Product is a product designed for delivery to a customer: Artifacts



Engineering

- Engineering is the process of designing and building something that serves a particular purpose and find a cost-effective solution to problems.
- Engineering is the science, skill, and profession of acquiring and applying scientific, economic, social, and practical knowledge, in order to design and also build structures, machines, devices, systems, materials and processes.

While

- *Manufacturing is the production of goods for use or sale using labour and machines, tools, chemical and biological processing, or formulation.*

What is Software Engineering?

- **Software Engineering** is an engineering discipline which is concerned with all aspects of software production.
- **Software engineers** should
  - adopt a systematic and organized approach to their work
  - use appropriate tools and techniques depending on
    - the problem to be solved
    - the development constraints
  - use the resources available

Definitions of Software Engineering

- At the first conference on software engineering in 1968, 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".

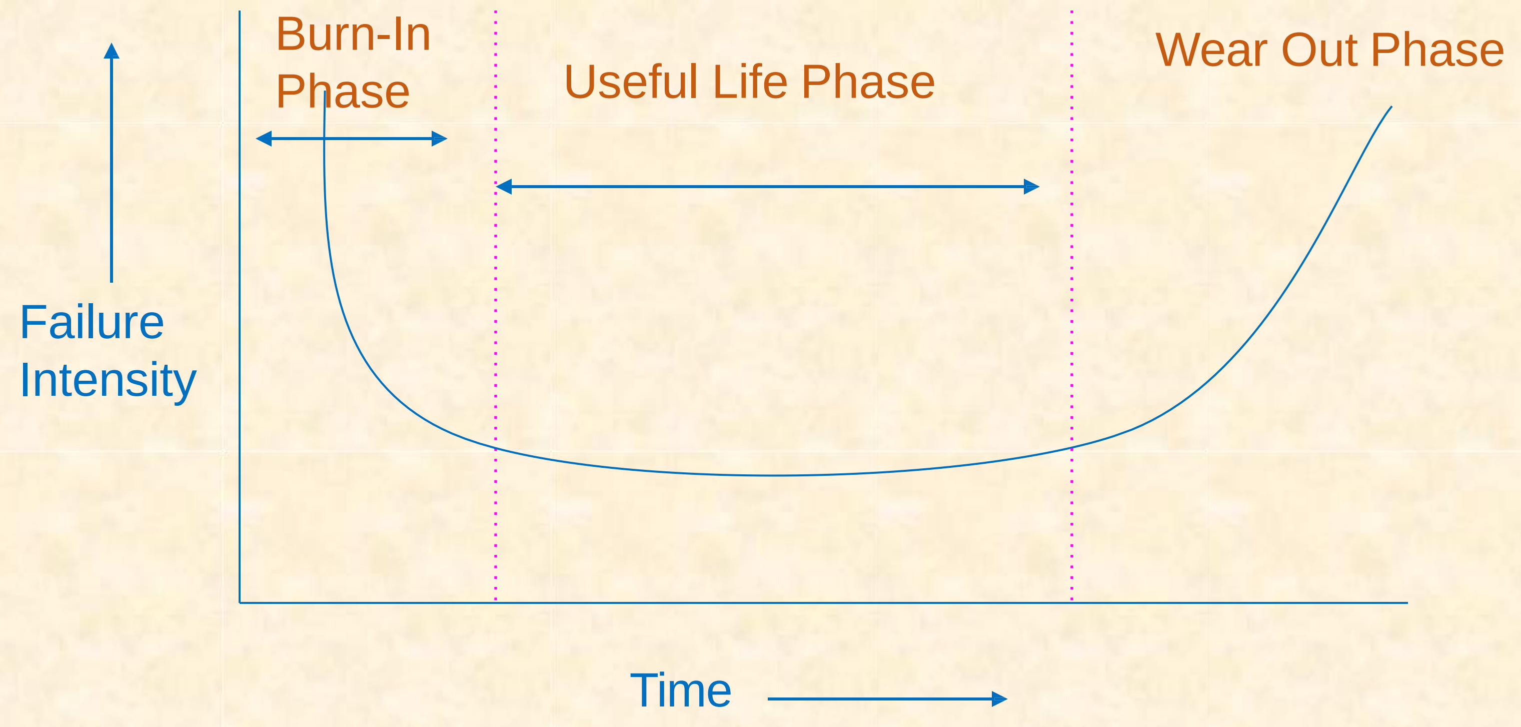
According to IEEE -

The application of systematic, disciplined and quantifiable approach to the development, operation, and maintenance of software.

- Due to increase in cost of maintaining software, *objective is now shifting to produce quality software that is maintainable, delivered on time, within budget, and also satisfies its requirements.*

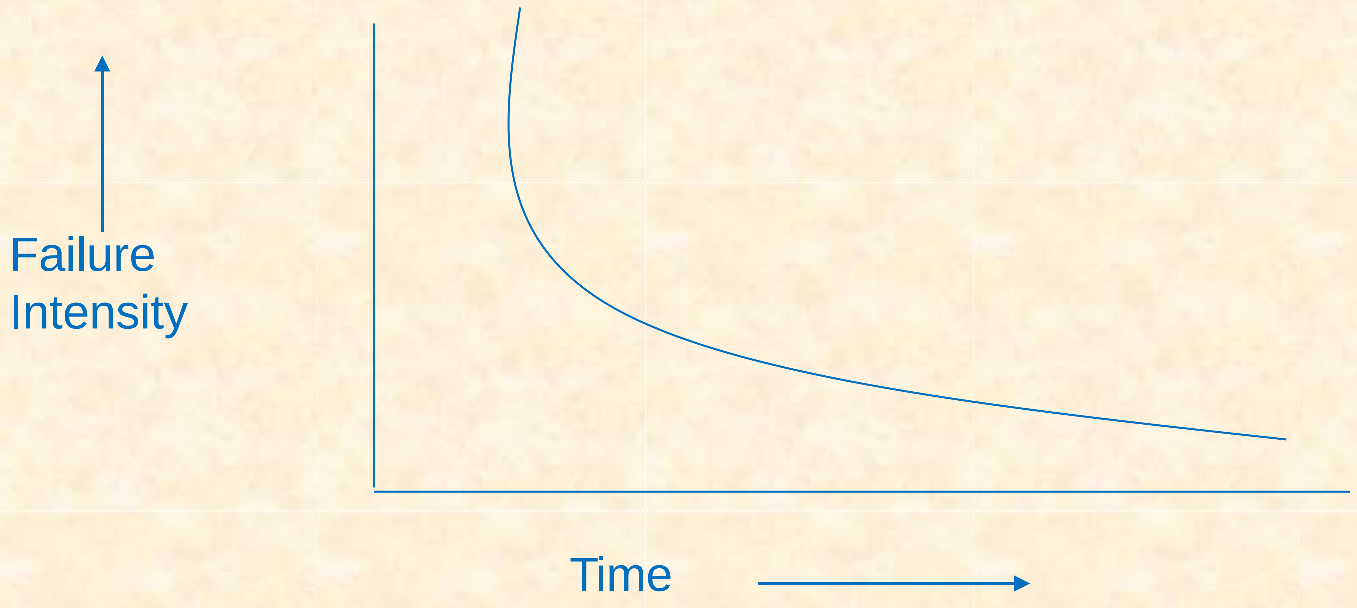
Software Characteristics

1. Software does not wear out – Like hardware



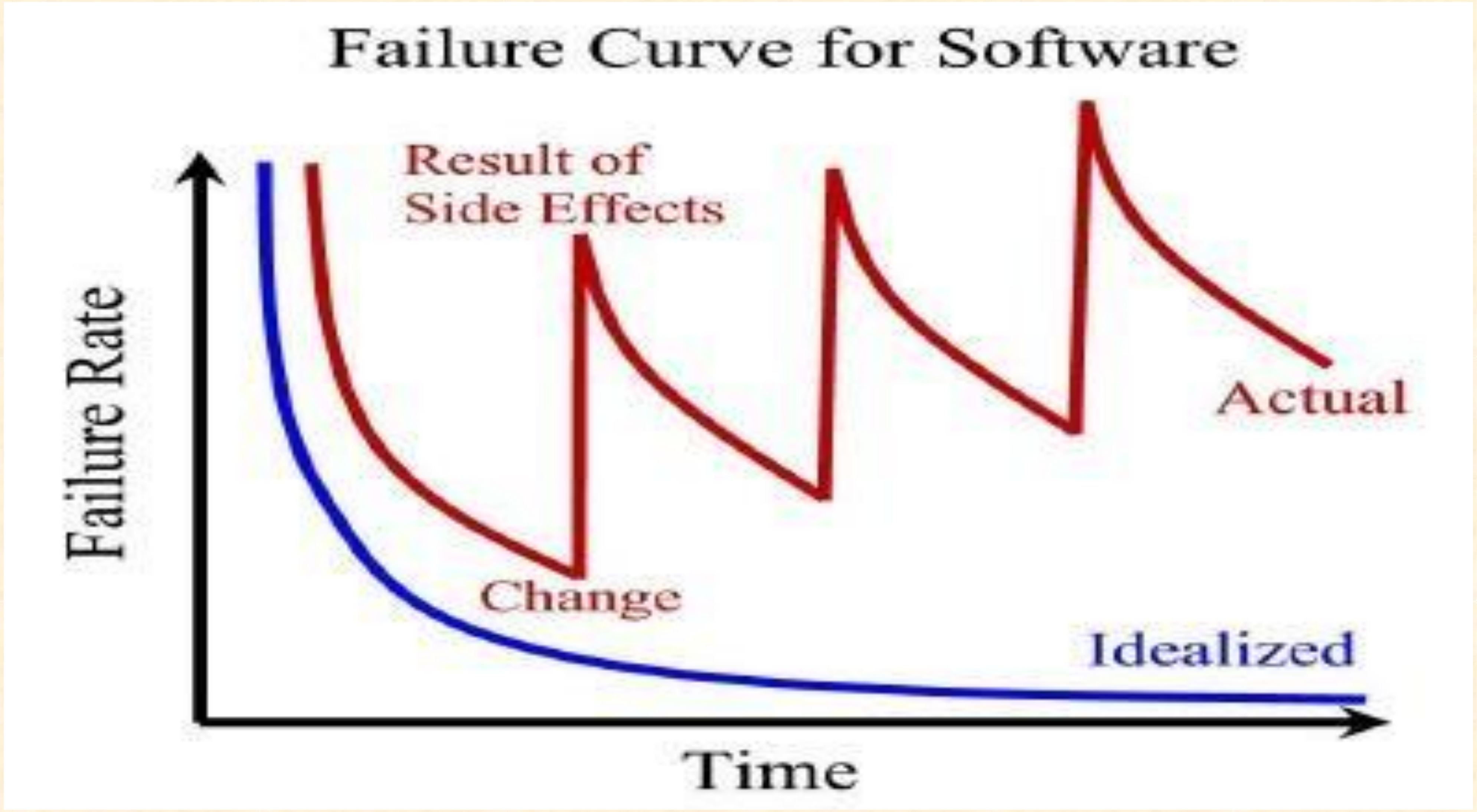
Bathtub Curve For Hardware

Software Characteristics



Ideal Software Failure Curve

Software may be retired due to environmental changes , new requirements, new expectations


Failure Curve for Software
Result of Side Effects
Failure Rate
Actual
Change
Idealized
Time

Software Characteristics

2. **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 continues 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 products, every product costs us due to raw material and other processing expenses.

Hence it is not manufactured in the classical sense.

Software Characteristics

3. Software is logical rather than physical.
4. Reusability of components.
5. Software is flexible.
6. Most software is custom built, rather than being assembled from existing components.

Software Myths

The Management Myths

Myth – We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality – The books very well exist

- But is it used?
- Are the practitioners aware of its existence?
- Is it complete?
- Is it adaptable?

In many cases the answer to all is no.

Software Myths

Myth – If we get behind schedule, we can add more programmers and catch up.

Reality – S/W development is not a mechanistic process like manufacturing.

- Adding people to a late s/w project makes it later .
  - spend time educating the newcomers, there by reducing the amount of time spent on productive development effort.
  - People can be added but only in a planned and well coordinated manner.

Software Myths

Myth – If I decide to outsource the s/w project to a 3rd party, I can just relax and let that firm build it.

Reality - If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsource s/w projects.

Software Myths

Customer myths

Myth – A general statement of objectives is sufficient to begin writing programs – we can add the details later.

- A formal and detailed description of the information domain, function, behaviour, performance, interfaces, design constraints, and validation criteria is essential.
- These characteristics can be determined only after thorough communication between customer and developer.

Software Myths

Myth: **Project requirements continually change, but change can be easily accommodated because software is flexible.**

- Reality: It is true that software requirements change, but the **impact of change varies with the time at which it is introduced.**
- If serious attention is given to up-front definition, **early requests for change** can be accommodated easily.
- When changes are requested during software design, the cost impact grows rapidly.
- Changes in function, performance, interface, or other characteristics during implementation (code and test) have a severe impact on cost.
- Change, when requested after software is in production, can be over an order of magnitude more expensive than the same change requested earlier.

Software Myths

Practitioner's Myth

Myth – Once we write the program and get it to work, our job is done.

Reality

– Someone once said that the sooner you begin writing code, the longer it will take to get done.

- Industry data indicate that between 60 to 80% of all effort expended on s/w will be expanded after it is delivered to the customer for the first time.

Software Myths

Myth – Until I get the program running, I have no way of accessing its quality

Reality - One of the most effective software quality assurance mechanisms can be applied from the inception of a **project—the *formal technical review***

Myth – The only deliverable work product for a successful project is the working program.

Reality - working program is only one part of a *software configuration that includes* many elements. Documentation provides a foundation for successful engineering and, more important, guidance for software support.

Software Myths

Myth – SE will make us create voluminous and unnecessary documentation and will invariably slows us down.

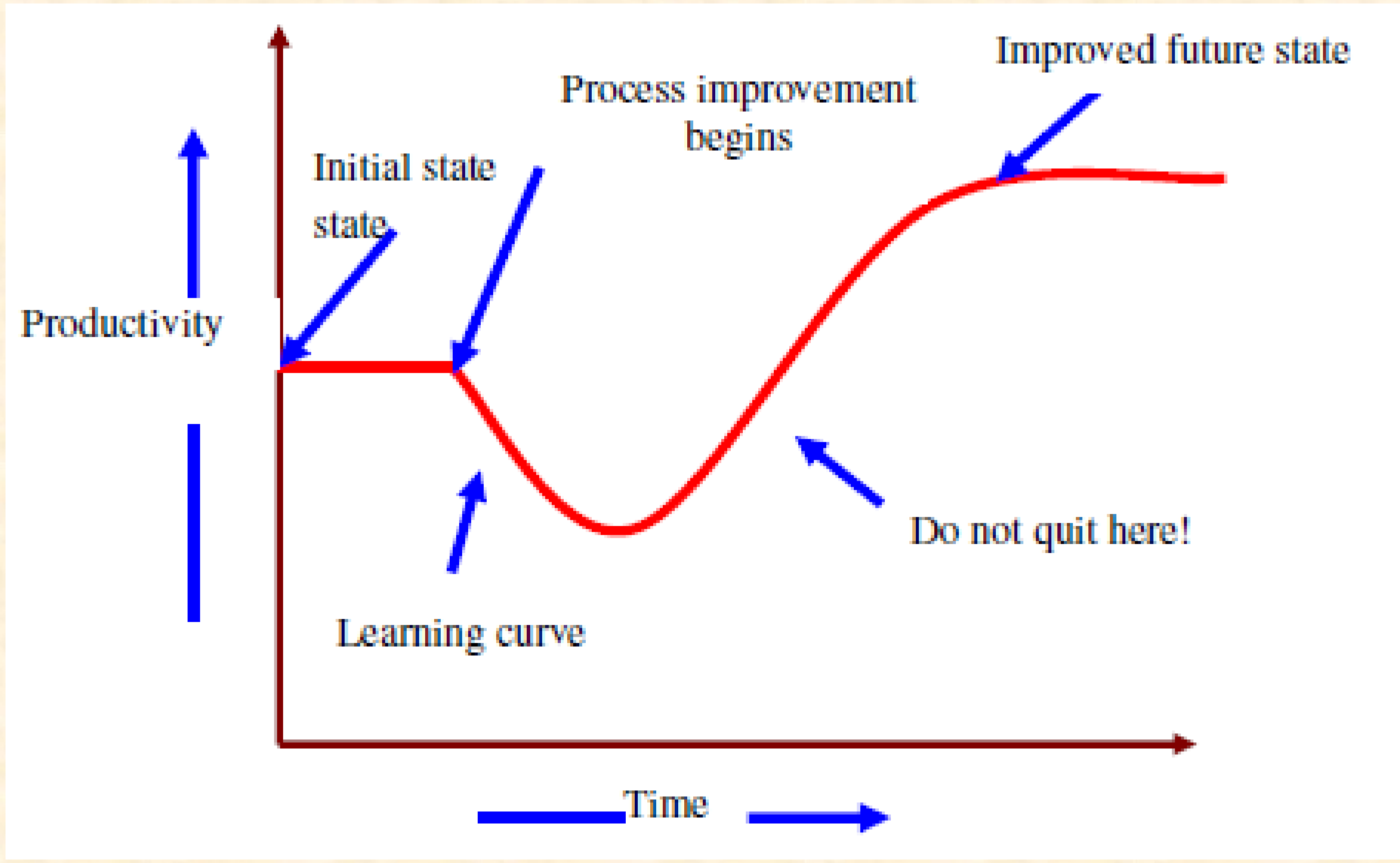
Reality – SE is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Other Myths

- Computers provide more reliability than systems they replace
- Reusing software increases safety
- Testing Software can remove all bugs
- Software will work right first time
- Software may be designed so thoroughly that it may not face integration problems
- Software with more features is a better software

Software Process

- Software process is a way in which we produce software.
- Why is it difficult to improve software process ?
    - Not enough time
    - Lack of knowledge
    - Insufficient commitment



Software Process

- **(Webster)** A system of operations in producing something; a series of actions, changes, or functions that achieve an end or a result.
- **(IEEE)** A sequence of steps performed for a given purpose

According to Pressman

- Software Engineering process is the glue that holds the Technology layers together and enables rational and timely development of Computer Software.
- Process defines a framework for a set of ***KEY PROCESS AREAS*** that must be established for effective delivery of software engineering technology.

Software Process

- A common process framework is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity.

- **Several Task Sets— Each A Collection Of Software Engineering**
    - Work Tasks
    - Project Milestones
    - Work Products
    - Quality Assurance Points

- **Process Enables** the framework activities to be **adapted to the characteristics of the software project** and the requirements of the project team.

Difference between Software Engineering and Traditional Engineering

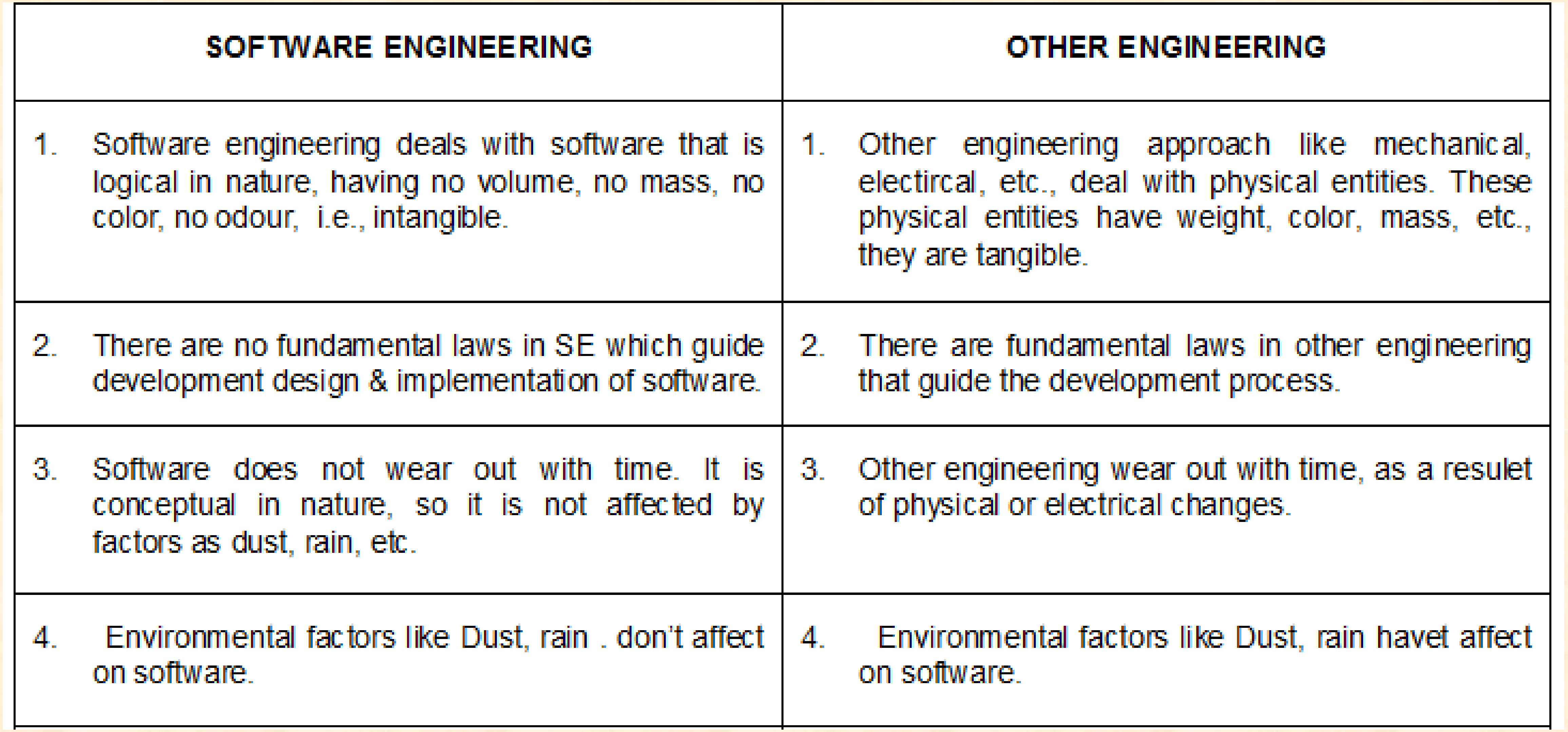
- Software is intangible
- Software doesn’t degrade with time as hardware does.
- Software failures are caused by design and implementation error, not by degradation over time

While In classical engineering disciplines

Engineer is equipped with tools and the mathematical maturity to specify the properties of the product separately from those of design.

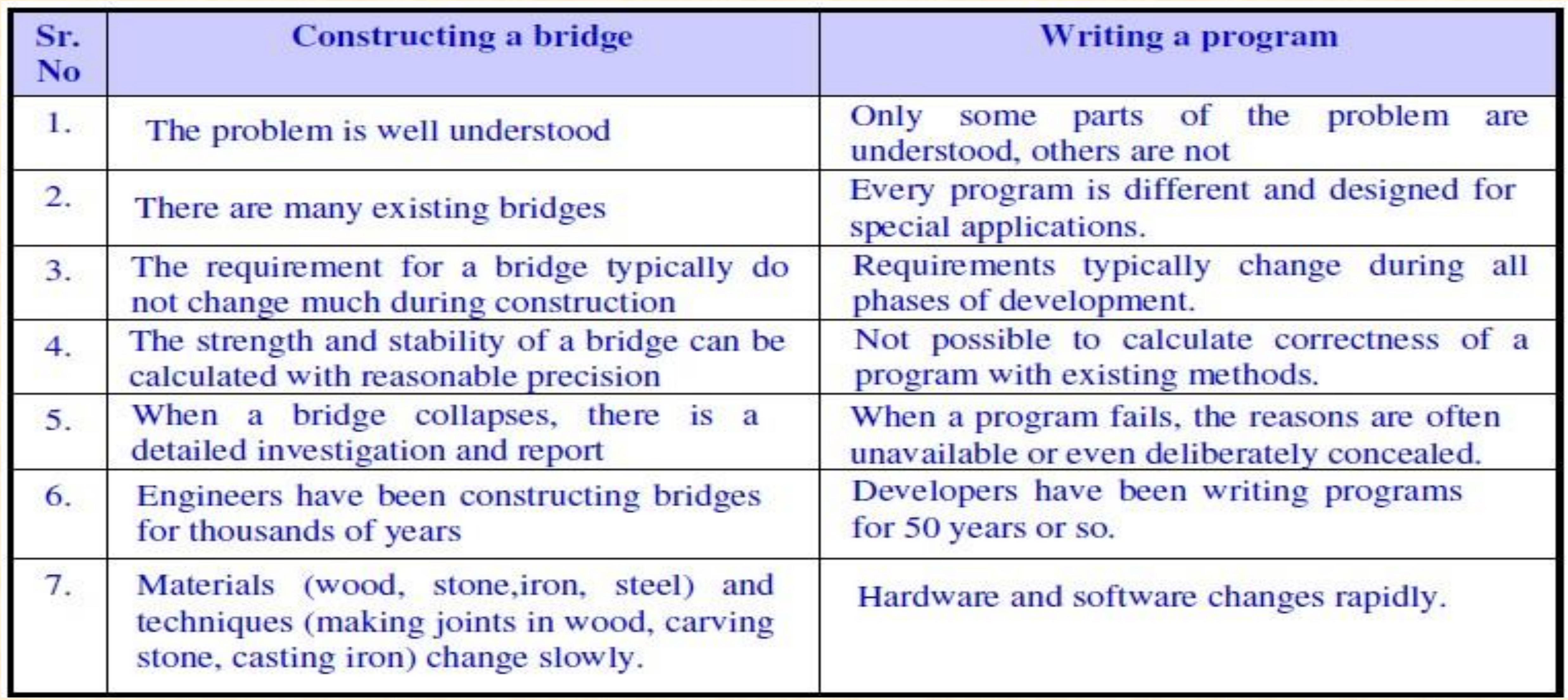
The typical software engineering relies much more on experience and judgment rather than mathematical formula.

Difference between Software Engineering and Traditional Engineering



SOFTWARE ENGINEERING	OTHER ENGINEERING
1. Software engineering deals with software that is logical in nature, having no volume, no mass, no color, no odour, i.e., intangible.	1. Other engineering approach like mechanical, electircal, etc., deal with physical entities. These physical entities have weight, color, mass, etc., they are tangible.
2. There are no fundamental laws in SE which guide development design & implementation of software.	2. There are fundamental laws in other engineering that guide the development process.
3. Software does not wear out with time. It is conceptual in nature, so it is not affected by factors as dust, rain, etc.	3. Other engineering wear out with time, as a resulet of physical or electrical changes.
4. Environmental factors like Dust, rain . don't affect on software.	4. Environmental factors like Dust, rain havet affect on software.

Constructing a BRIDGE compared to SOFTWARE DEVELOPMENT



Sr. No	Constructing a bridge	Writing a program
1.	The problem is well understood	Only some parts of the problem are understood, others are not
2.	There are many existing bridges	Every program is different and designed for special applications.
3.	The requirement for a bridge typically do not change much during construction	Requirements typically change during all phases of development.
4.	The strength and stability of a bridge can be calculated with reasonable precision	Not possible to calculate correctness of a program with existing methods.
5.	When a bridge collapses, there is a detailed investigation and report	When a program fails, the reasons are often unavailable or even deliberately concealed.
6.	Engineers have been constructing bridges for thousands of years	Developers have been writing programs for 50 years or so.
7.	Materials (wood, stone,iron, steel) and techniques (making joints in wood, carving stone, casting iron) change slowly.	Hardware and software changes rapidly.

QUALITY???

Software quality

Conformance to:

- **Explicitly Stated Functional And Performance Requirements**
- **Explicitly Documented Development Standards**
- **Implicit Characteristics**

which are expected from all professionally developed software.

McCall’s Quality Factors

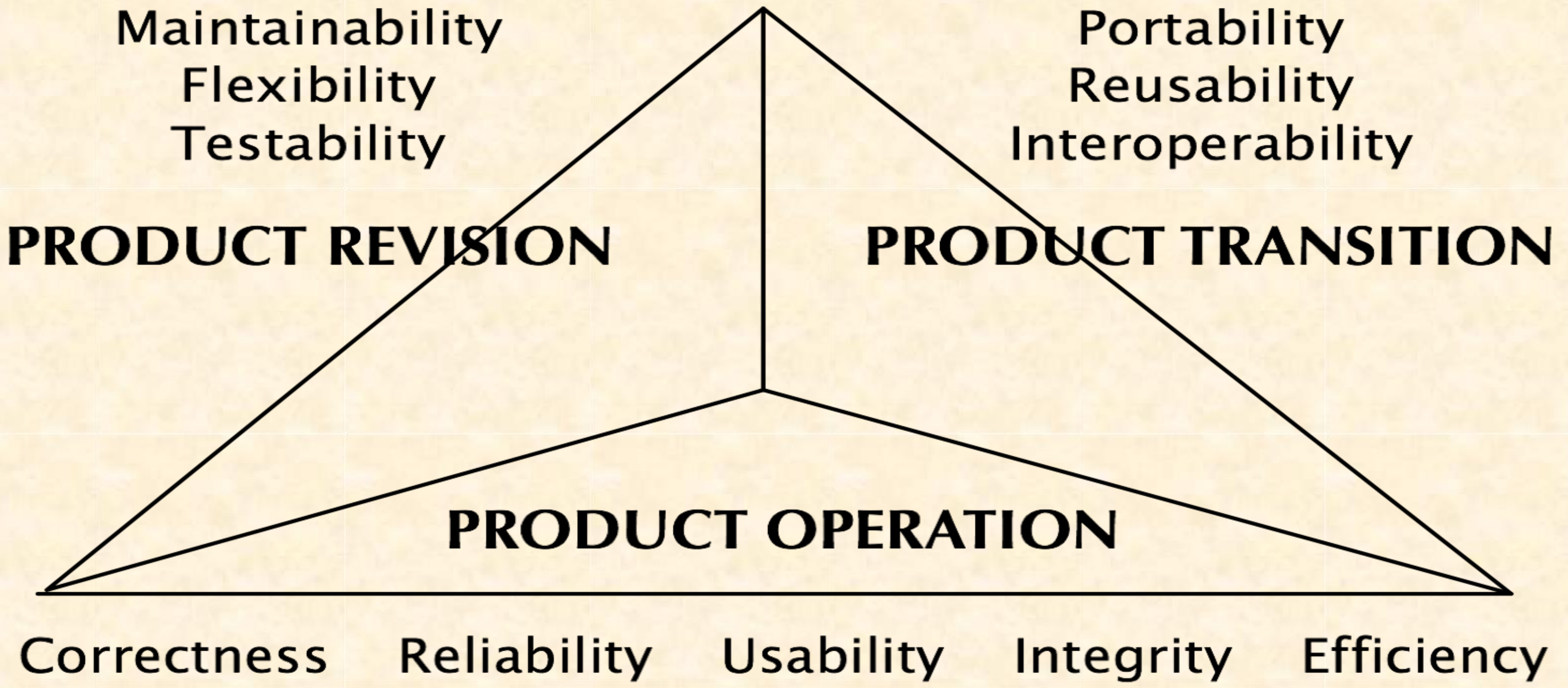
- Two categories of software quality factors:
  - Factors that can be directly measured (bugs or defects)
  - Factors that can be measured only indirectly (usability or maintainability)
- Both sets can (and should) be measured

Mccall’s Quality Factors (contd..)

- McCall, Richards, and Walters group these factors into three categories, focused on three aspects of a software product:

  - Its operational characteristics : **PRODUCT OPERATION**
  - Its ability to undergo change : **PRODUCT REVISION**
  - Its adaptability to new environments : **PRODUCT TRANSITION**

McCall’s Quality Factors (contd..)



Product Operation

- It includes five software quality factors, which are related with the requirements that directly affect the operation of the software such as operational performance, convenience, ease of usage and its correctness. These factors help in providing a better user experience.
- Correctness:
  - The extent to which a program satisfies its specifications and fulfills the customer's mission objectives
- Reliability:
  - The extent to which a software performs its intended functions without failure under specified operating conditions.
- Usability:
  - The extent of effort required to learn, operate and understand the functions of the software.

Product Operation (cont.)

- Integrity:
  - The extent to which access to software or data by unauthorized persons can be controlled
- Efficiency:
  - The amount of hardware resources and code the software, needs to perform a function.

Product Revision

It includes three software quality factors, which are required for testing and maintenance of the software. They provide ease of maintenance, flexibility and testing effort to support the software to be functional according to the needs and requirements of the user in the future.

- Maintainability:
  - The effort required to detect and correct an error during maintenance phase.
- Flexibility:
  - The effort required to modify an operational program
- Testability:
  - The effort required to verify a software to ensure that it meets the specified requirements.

Product Transition

- It includes three software quality factors, that allows the software to adapt to the change of environments in the new platform or technology from the previous.
- Portability:
  - the effort required to transfer the program from one hardware and/or software system to another
- Reusability:
  - the extent to which a program (or parts or a program) can be reused in other applications
- Interoperability:
  - the effort required to couple one system to another

Software Development Life Cycle

- SOFTWARE DEVELOPMENT LIFECYCLE (SDLC) is a systematic process for building software that ensures the quality and correctness of the software built.

- Life Cycle models take care of the order of activities performed on a software product from its inception to retirement.

- Different life cycle models may map the basic development activities to phases in different ways.

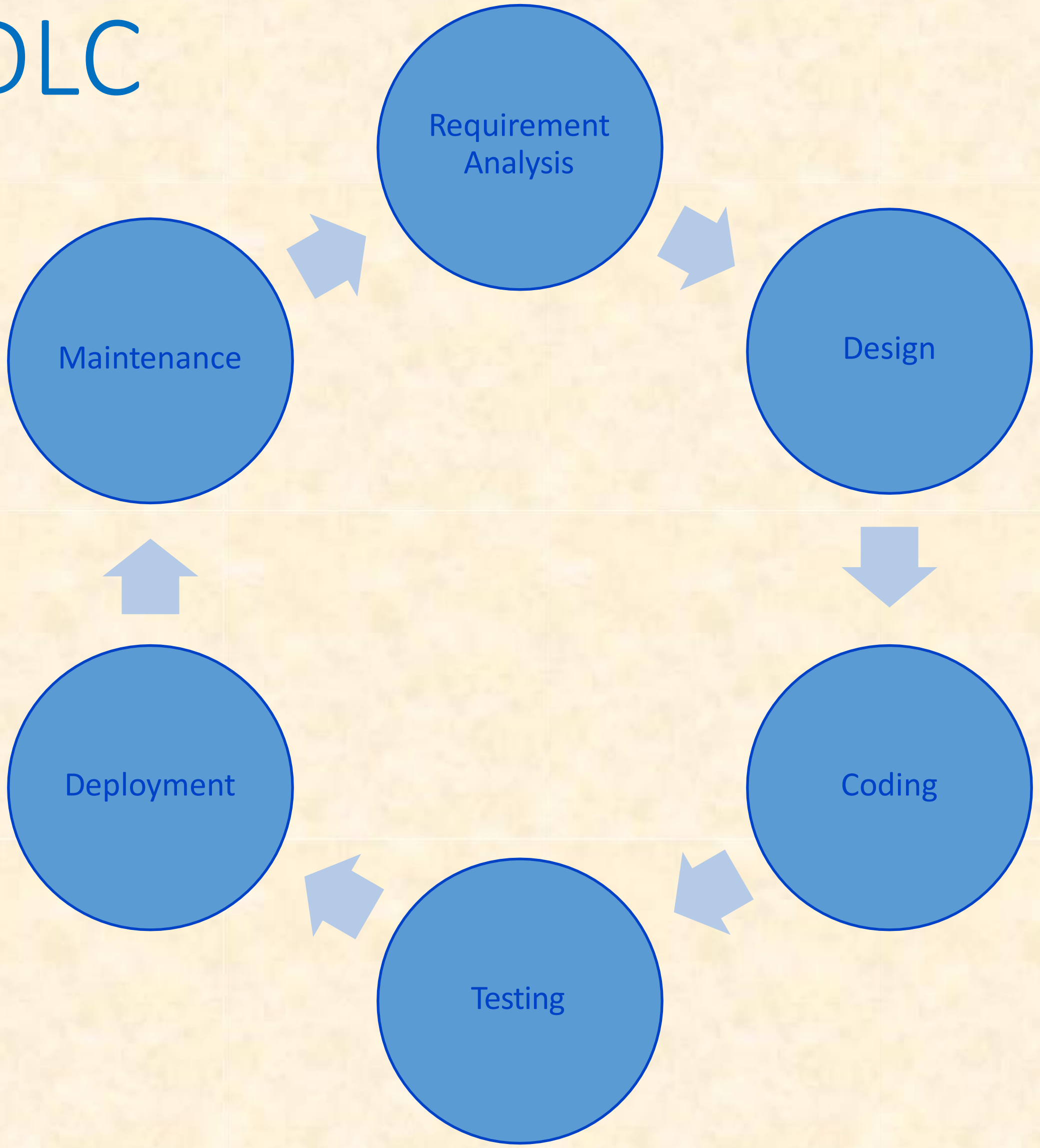
- Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models.

- During any life cycle phase, more than one activity may also be carried out.

Need of SDLC

- Development of software product in a systematic and disciplined manner
- Clear understanding among team members about when and what to do
- Each phase defines entry and exit criteria for every phase.

Phases of SDLC



Software Development Life Cycle (SDLC):Requirement Analysis

- During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

- Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product.

- Before building a product a core understanding or knowledge of the product is very important.

- Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

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

Software Development Life Cycle (SDLC): Design

- This stage involves the design of the entire system and its elements.
- There are two kinds of design, high-level design and low-level design.
- According to their definitions, a high-level design (HLD) is the overall plan of the system, while a low-level design (LLD) is a design of its components. Failure at this phase certainly result cost overruns and total collapse at worst.
- The system and software design documents (SDD) are prepared as per the requirement specification document. There are two kinds of design documents developed in this phase:

Software Development Life Cycle (SDLC):Design

- High-Level Design (HLD)
    - Brief description and name of each module
    - An outline about the functionality of every module
    - Interface relationship and dependencies between modules
    - Database tables identified along with their key elements
    - Complete architecture diagrams along with technology details
- Low-Level Design(LLD)
    - Functional logic of the modules
    - Database tables, which include type and size
    - Complete detail of the interface
    - Addresses all types of dependency issues
    - Listing of error messages
    - Complete input and outputs for every module

Software Development Life Cycle (SDLC):Coding

- Once the system design phase is over, the next phase is coding.
- In this stage of SDLC the actual development starts, and the product is built.
- In the coding phase, tasks are divided into units or modules and assigned to the various
  developers for writing codes as per the chosen programming language.
- It is the longest phase of the Software Development Life Cycle process.

Software Development Life Cycle (SDLC): Testing

- Testing starts once the coding is complete and the modules are released for testing.

- In this phase, the developed software is tested thoroughly, and any defects found are assigned to developers to get them fixed.

- During this phase, QA and testing team may find some bugs/defects which they communicate to developers.

- The development team fixes the bug and send back to QA for a re-test.

- This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Software Development Life Cycle (SDLC): Deployment

- At this stage, the goal is to deploy the software.
- When the software is completed and has no bugs, it is shipped to the market for beta testing.
- The support team collects feedback of the first users, and if any bugs come up, the development team fixes them. After that, the final version is rolled out.

Software Development Life Cycle (SDLC):Maintenance

- Once the software system is deployed, and customers start using the developed system, following 3 activities occur:
  - Bug fixing - bugs are reported because of some scenarios which are not tested at all
  - Upgrade - Upgrading the application to the newer versions of the Software
  - Enhancement - Adding some new features into the existing software

Importance of SDLC

- It provides visibility for the engaged parties
- It allows to control the project
- Predictable deliveries throughout an entire development process
- Eliminating risks like going over budget or deadline breach
- The process goes on until all the requirements are met

SDLC Models

- There are various software development life cycle models defined and designed which are followed during 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, in order to ensure success in process of software development.

- Following are the most important and popular SDLC models (which will discuss in this syllabus) followed in the industry:

1. SDLC: Waterfall Model
2. SDLC: Prototype Model
3. SDLC: Spiral Model
4. SDLC: Evolutionary Development Models
5. SDLC: Iterative Enhancement Models.

Waterfall Model

The classical waterfall model is intuitively the most obvious way to develop software.

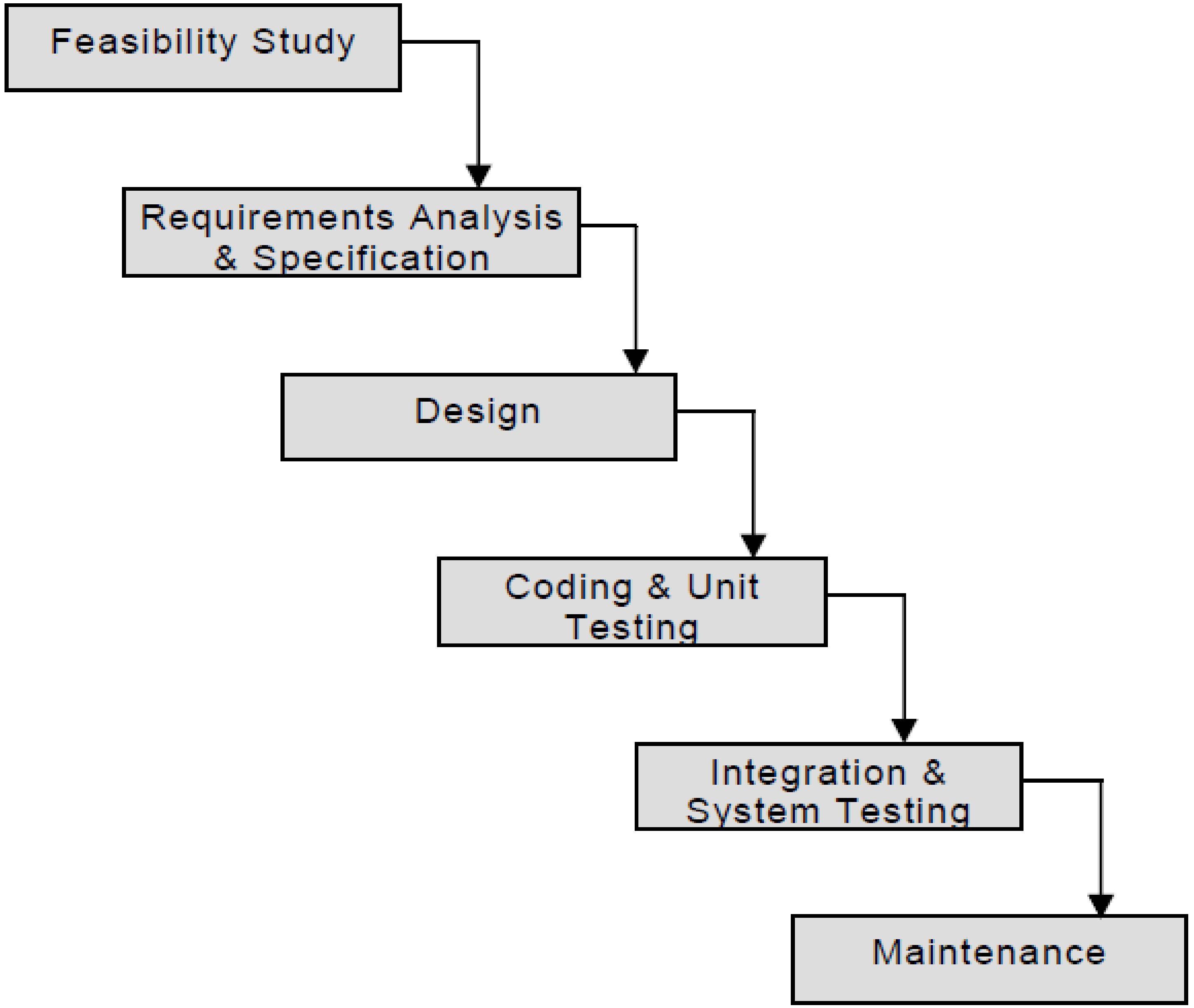
Though the classical waterfall model is elegant and intuitively obvious,

it is not a practical model in the sense that it is generally not used as it is in actual software development projects.

Thus, this model can be considered to be a *theoretical way of developing software*.

But all other life cycle models are essentially derived from the classical waterfall model.

So, in order to be able to appreciate other life cycle models it is necessary to learn the classical waterfall model.


Feasibility Study
Requirements Analysis & Specification
Design
Coding & Unit Testing
Integration & System Testing
Maintenance

- All these **phases are cascaded** to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases.
- The next phase is started **only after the defined set of goals are achieved** for previous phase and it is signed off, so the name "Waterfall Model". In this model phases do not overlap.

Activities undertaken during Feasibility Study

- At the time of feasibility study PROJECT MANAGERS or TEAM LEADERS have a rough understanding that what is to be done.
- This is done based on different input and output data to be produced by the system.
- After overall understanding, different possible solutions are investigated in terms of ***resources required, cost of development, development time***.
- Based on this analysis they pick the ***best solution and determine whether the solution is feasible financially and technically***.
- They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

Activities undertaken during Requirement Analysis and specification

- The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely
  i. **Requirements gathering and analysis, and**
  ii. **Requirements specification**
- The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed.
  - This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.

Activities undertaken during Requirement Analysis and specification (contd..)

- After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start.

  - During this activity, the user requirements are systematically organized into a ***Software Requirements Specification (SRS)*** document.

Activities undertaken during Design

- The goal of the design phase is to transform the requirements specified in the SRS document into a ***structure that is suitable for implementation in some programming language.***
- In technical terms, during the design phase the ***software architecture is derived*** from the SRS document.
- Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

Activities undertaken during Coding and Unit Testing

- The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module.

- The ***end-product of this phase is a set of program modules*** that have been individually tested.

- During this phase, ***each module is unit tested*** to determine the correct working of all the individual modules.

- It involves ***testing each module in isolation*** as this is the most efficient way to debug the errors identified at this stage.

Activities undertaken during Integration and system testing

- Integration of different modules is undertaken once they have been coded and unit tested.
- During the integration and system testing phase, the modules are integrated in a planned manner.
- The different modules making up a software product are almost never integrated in one shot.
- Finally, when all the modules have been successfully integrated and tested, system testing is carried out.
- The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document.
- System testing usually consists of three different kinds of testing activities:
  - α – testing: It is the system testing performed by the development team.
  - β – testing: It is the system testing performed by a friendly set of customers.
  - Acceptance Testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Activities undertaken during Maintenance

- Maintenance involves performing any one or more of the following three kinds of activities:
  - Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
  - Improving the implementation of the system and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
  - Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.

Waterfall Model Application

- Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are:

  - Requirements are very well documented, clear and fixed.
  - Product definition is stable.
  - Technology is understood and is not dynamic.
  - There are no ambiguous requirements.
  - Ample resources with required expertise are available to support the product.
  - The project is short.

Pros & Cons of Waterfall Model

Pros	Cons
Simple and easy to understand and use	No working software is produced until late during the life cycle.
Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.	High amounts of risk and uncertainty.
Phases are processed and completed one at a time.	Not a good model for complex and object-oriented projects.
Works well for smaller projects where requirements are very well understood.	Poor model for long and ongoing projects.
Clearly defined stages.	Not suitable for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model.
Well understood milestones.	It is difficult to measure progress within stages.
Easy to arrange tasks.	Cannot accommodate changing requirements.
Process and results are well documented.	Adjusting scope during the life cycle can end a project.
	Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

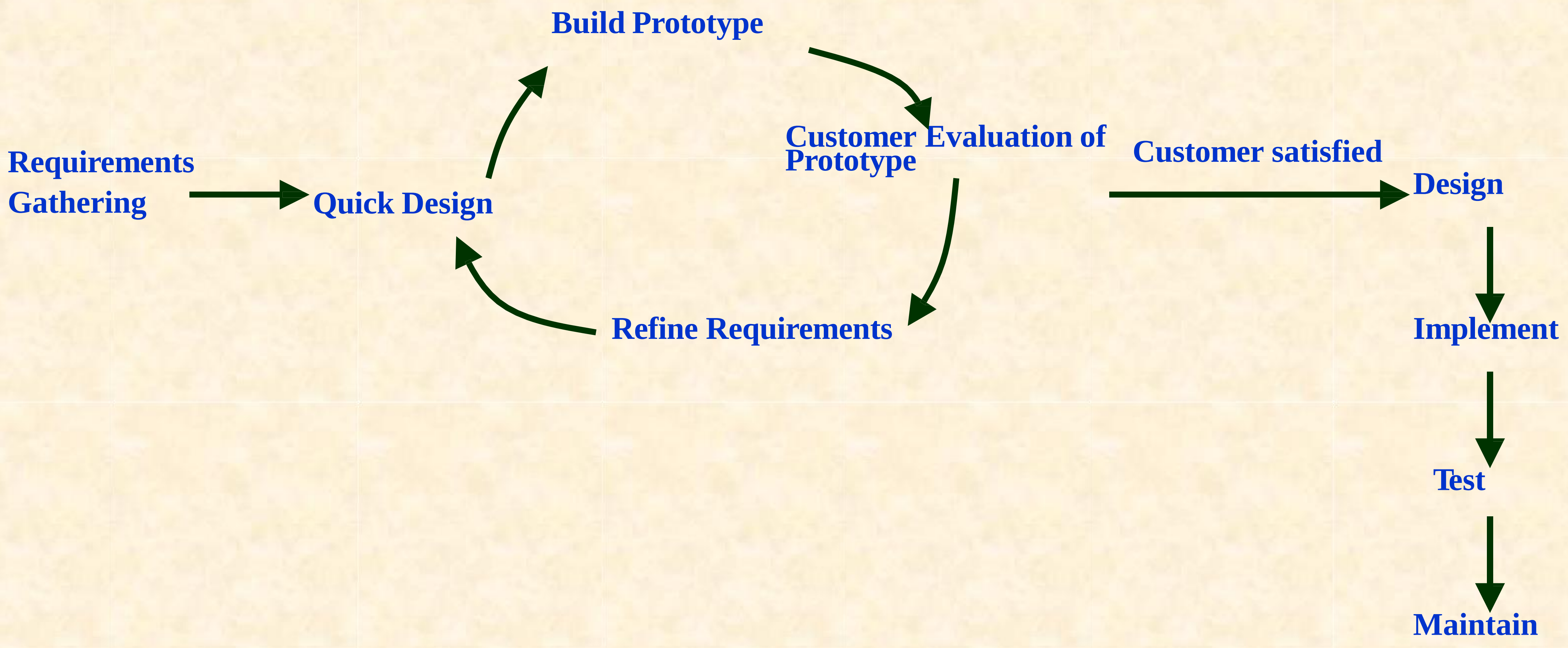
Prototype Model

- The Prototyping model is suitable for projects, which either the customer requirements or the technical solutions are not well understood.
- This risks must be identified before the project starts. This model is especially ***popular for the development of the user interface part of the project.***
- In this process model, *the system is partially implemented before or during the analysis phase thereby giving the customers an opportunity to see the product early in the life cycle*.
- The process starts by interviewing the customers and developing the incomplete high-level paper model.
- This document is used to build the initial prototype supporting only the basic functionality as desired by the customer.
- Once the customer figures out the problems, the prototype is further refined to eliminate them. The process continues till the user approves the prototype and finds the working model to be satisfactory.

Prototype Model

- This is a valuable mechanism for gaining better understanding of the customer’s needs:
  - How the screens might look like
  - How the user interface would behave
  - How the system would produce outputs
- The developer may be unsure of the **efficiency of an algorithm**, the **adaptability of an operating system**, the form that the **human-machine interaction** should take etc.

The Prototype Model



Prototype Model

- Basic Requirement Identification: This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.
- Developing the initial Prototype: The initial Prototype is developed in this stage, where the very basic requirements are show cased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed and the workarounds are used to give the same look and feel to the customer in the prototype developed.

Prototype Model

- Review of the Prototype: The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

- Revise and enhance the Prototype: The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like, time and budget constraints and technical feasibility of actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until customer expectations are met.

Prototype Model: Advantages

- The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
- New requirements can be easily accommodated as there is scope for refinement.
- Missing functionalities can be easily figured out.
- Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
- The developed prototype can be reused by the developer for more complicated projects in the future.
- Flexibility in design.

Prototype Model: Disadvantages

- Costly w.r.t time as well as money.
- There may be too much variation in requirements each time the prototype is evaluated by the customer.
- Poor Documentation due to continuously changing customer requirements.
- It is very difficult for the developers to accommodate all the changes demanded by the customer.
- There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.
- After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.
- Developers in a hurry to build prototypes may end up with sub-optimal solutions.
- The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

Spiral Model (Boehm's Model)

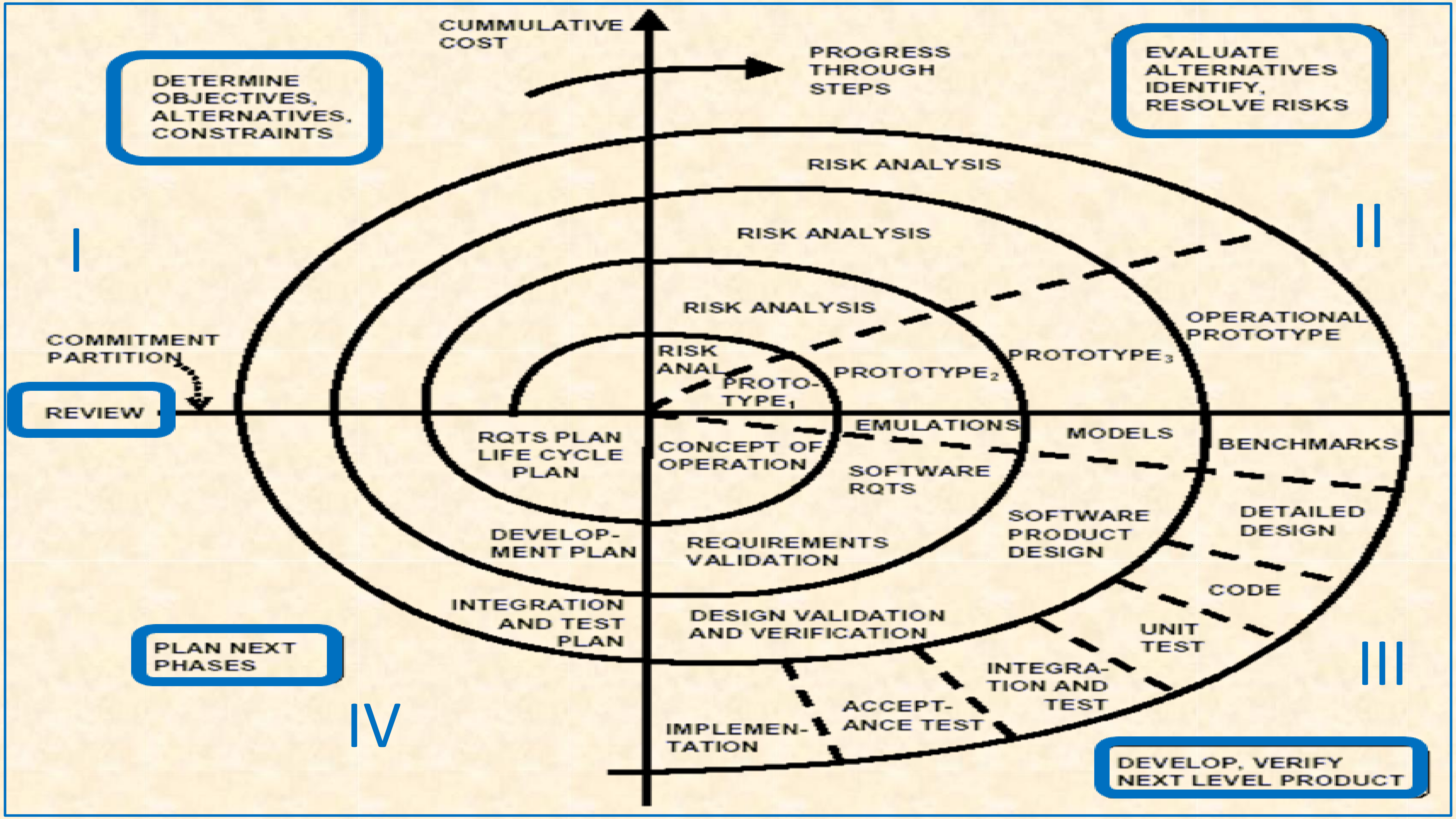
- **Spiral model (given by Barry Boehm)** is one of the most important Software Development Life Cycle models, which provides support for **Risk Handling**.
- In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project.
- **Each loop of the spiral is called a Phase of the software development process.**
- The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks.
- As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using spiral model.
- The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

Spiral Model

- Each phase of Spiral Model is divided into four quadrants as shown in the above figure. 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 is identified and the risks are resolved using the best possible strategy. At the end of this quadrant, Prototype is built for the best possible solution.
3. **Develop 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. Review of the progress done so far is also taken care of in this phase,

SPIRAL MODEL



Spiral Model

- The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype.
- The spiral model supports coping up with risks by providing the scope to build a prototype at every phase of the software development.
- In each phase of the Spiral Model, the features of the product dated and analyzed and the risks at that point of time are identified and are resolved through prototyping. Thus, this model is much more flexible compared to other SDLC models.

Spiral Model as Meta Model

- The Spiral model is called as a Meta Model because it subsumes all the other SDLC models. For example, a single loop spiral represents the Iterative Waterfall Model.
- The spiral model incorporates the stepwise approach of the Classical Waterfall Model.
- The spiral model uses the approach of **Prototyping Model** by building a prototype at the start of each phase as a risk handling technique.
- Also, the spiral model can be considered as supporting the evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

Spiral Model: Advantages

- **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
- **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
- **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
- **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

Spiral Model: Disadvantages

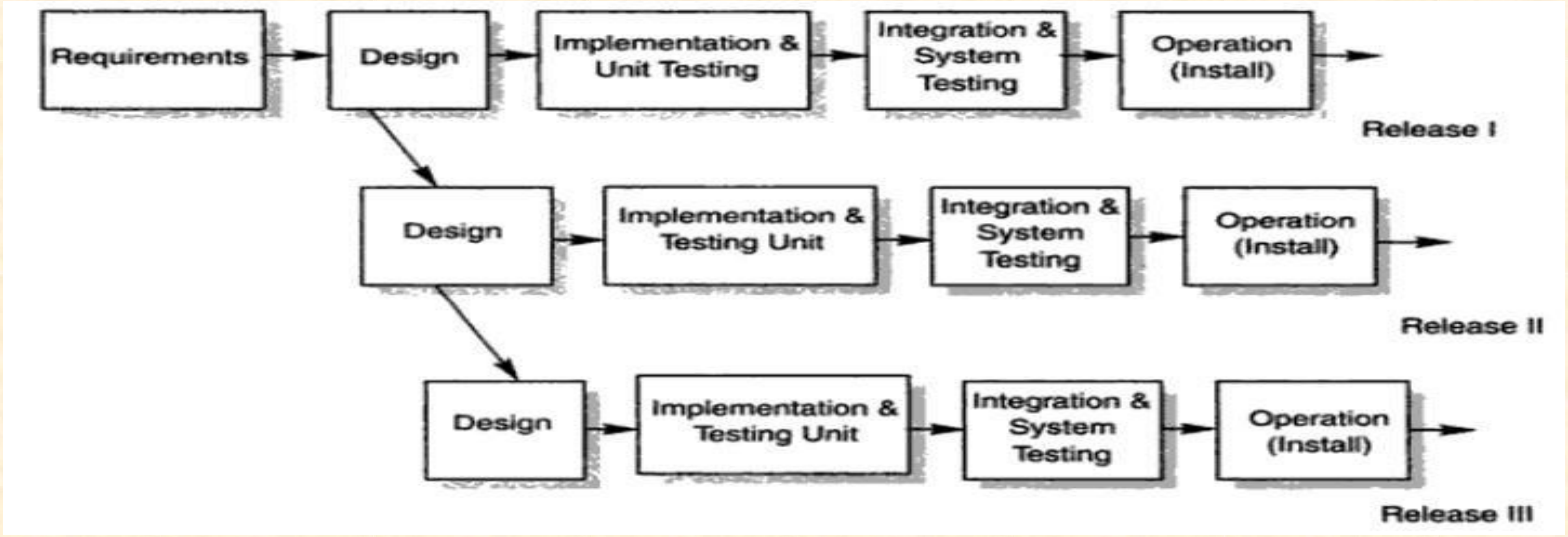
- **Complex:** The Spiral Model is much more complex than other SDLC models.
- **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
- **Too much dependable on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced expertise, it is going to be a failure to develop a project using this model.
- **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

Iterative Enhancement Model

- This model is similar to the Waterfall Model. The requirement analysis phase is the initial phase of life cycle where all the requirements are summarized.
- All requirements are not implemented together rather a priority list is created in consent with the customer.
- The requirements are implemented phase wise and at the end of each phase we get a usable product.
- The phase wise implementation is evaluated to see for any modifications needed in further phases.
- This process is then repeated, producing a new version of the software at the end of each iteration of the model.

Iterative Model design

- The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).


Requirements
Design
Implementation & Unit Testing
Integration & System Testing
Operation (Install)
Release I
Design
Implementation & Testing Unit
Integration & System Testing
Operation (Install)
Release II
Design
Implementation & Testing Unit
Integration & System Testing
Operation (Install)
Release III

Iterative Model Design

- Model gives a working model quickly in few weeks or months.
- The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model.
- As the software evolves through successive cycles, tests have to be repeated and extended to verify each version of the software.

Iterative Model : Application

- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high-risk features and goals which may change in the future.

Evolutionary Model

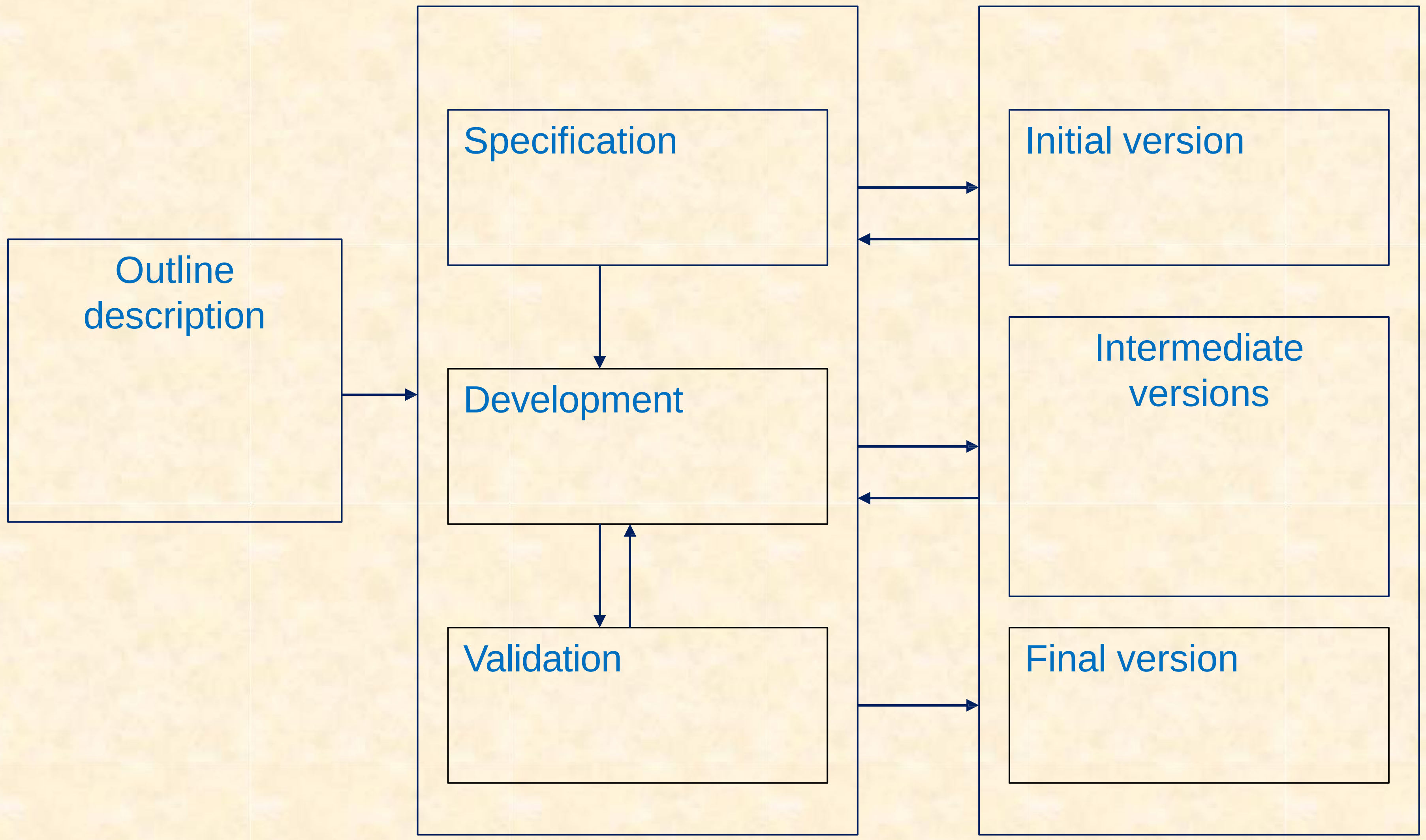
- **Evolutionary model** delivers the final system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done.
- It is better for software products that have their feature sets redefined during development because of user feedback and other factors.
- The Evolutionary development model similar to the iterative enhancement model divides the development cycle into smaller, incremental waterfall models.
- Here the main difference is that we donot get a working model at the end of each cycle.
- Example : A simple database application
    - First phase may implement just the GUI
    - Second File Manipulation
    - Third Queries
    - And Fourth the Updation

  We need all these 4 parts to be there to make system usable

- Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves with time.

- Evolutionary model may be incorporated when rapid delivery of working model is not a constraint.

Evolutionary Model


Outline description
Specification
Development
Validation
Initial version
Intermediate versions
Final version