1. SOFTWARE ENGINEERING FUNDAMENTALS

Contents:

1. Software and Programs
2. Software Engineering
3. Types of Software
4. Software process and Framework
5. The Unified Process
6. Professional Software Products
8. Software Engineering Diversity
9. Software Engineering Ethics
10. Perspective Process Models
    - Waterfall,, Incremental, Evolutionary, Spiral

1.1 Software

- A set of instruction or computer program that when executed provide desired functions and performance.
- Data structure that enable the programs to adequately manipulate information.
- Descriptive information in both hard copy and virtual forms that describes the operation and use of the programs.

Not only the set of programs:

Also associated with:

- Documentation
- Configuration data

Software Definition:

- collection of programs, configuration files, documentation, user manual, update facilities and support.
- Product that software professionals build and then support over the long term.

Characteristics of Software

1. Software is developed or engineered, but not manufactured.

   Software is a design of strategies, instruction which finally perform the designed, instructed tasks. And a design can only be developed, not manufactured.

2. Software does not “wear out”

   Software is not susceptible to the environmental melodies and it does not suffer from any effects with time

3. Most softwares are custom-built, rather being assembled from existing components

Hardware/Software Reliability Curve

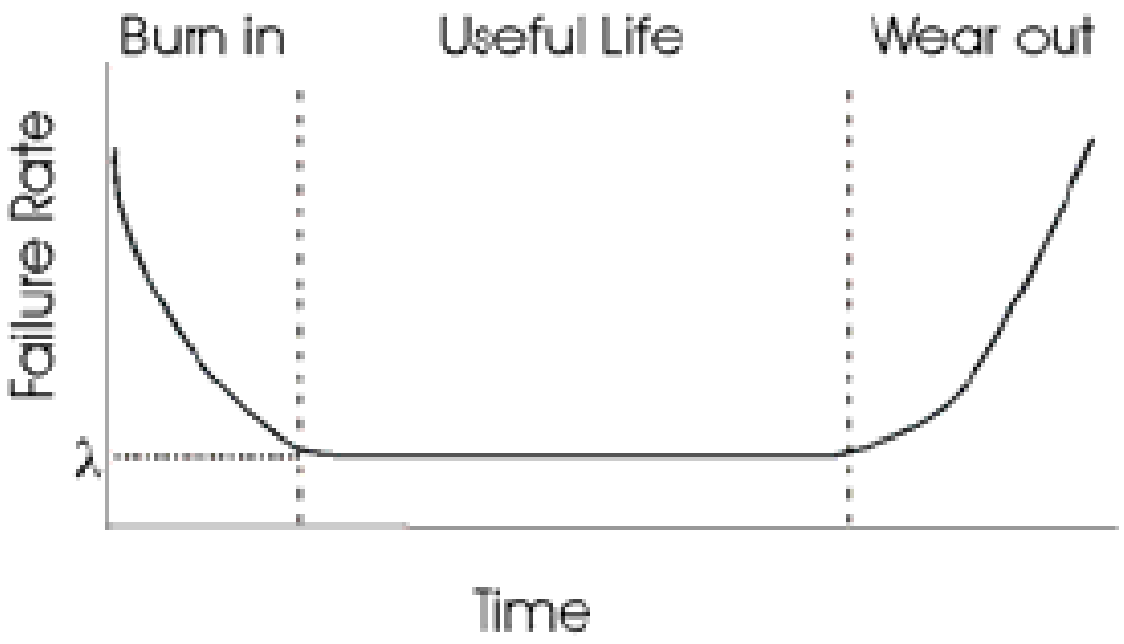


Fig. 1: Hardware Reliability Curve

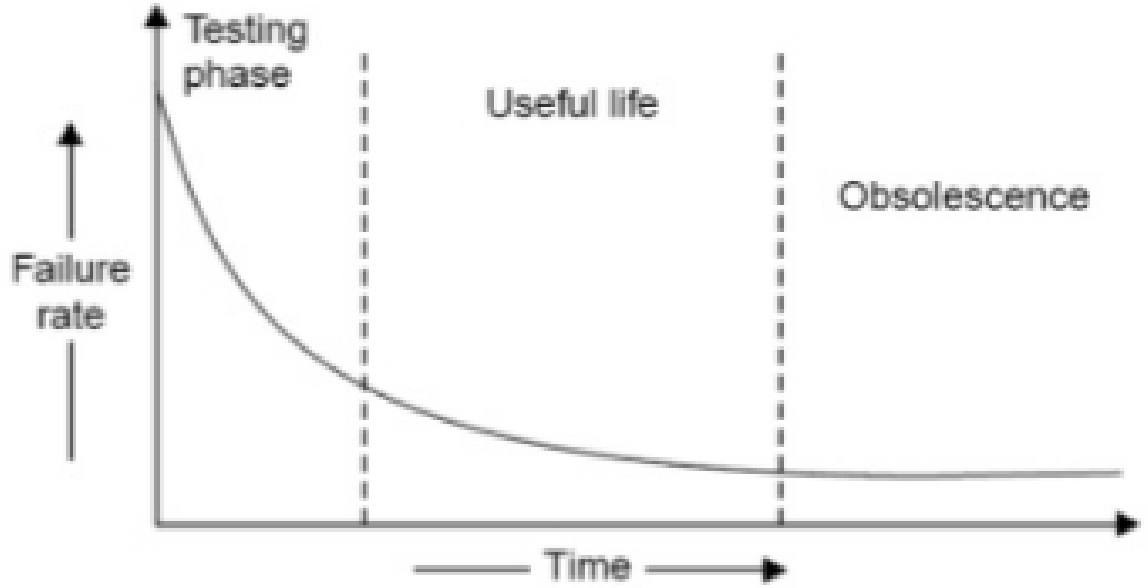


Fig. 2: Software Reliability Curve

Program Vs Software

	Program		Software
1.	Usually small in size.	1.	Large
2.	Author himself is sole user.	2.	Large number of users
3.	Single developer.	3.	Team of developers.
4.	Lacks proper documentation.	4.	Well documented and user manual prepared.
5.	Adhoc (i.e. not systematic / unplanned) development.	5.	Systematic development.
6.	Database not part of program.	6.	Database and program are parts of software.
7.	Programs are instructions that when executed provide desired feature and functions.	7.	Software is the collection of data structures, programs and other documentation to enable manipulation of data.

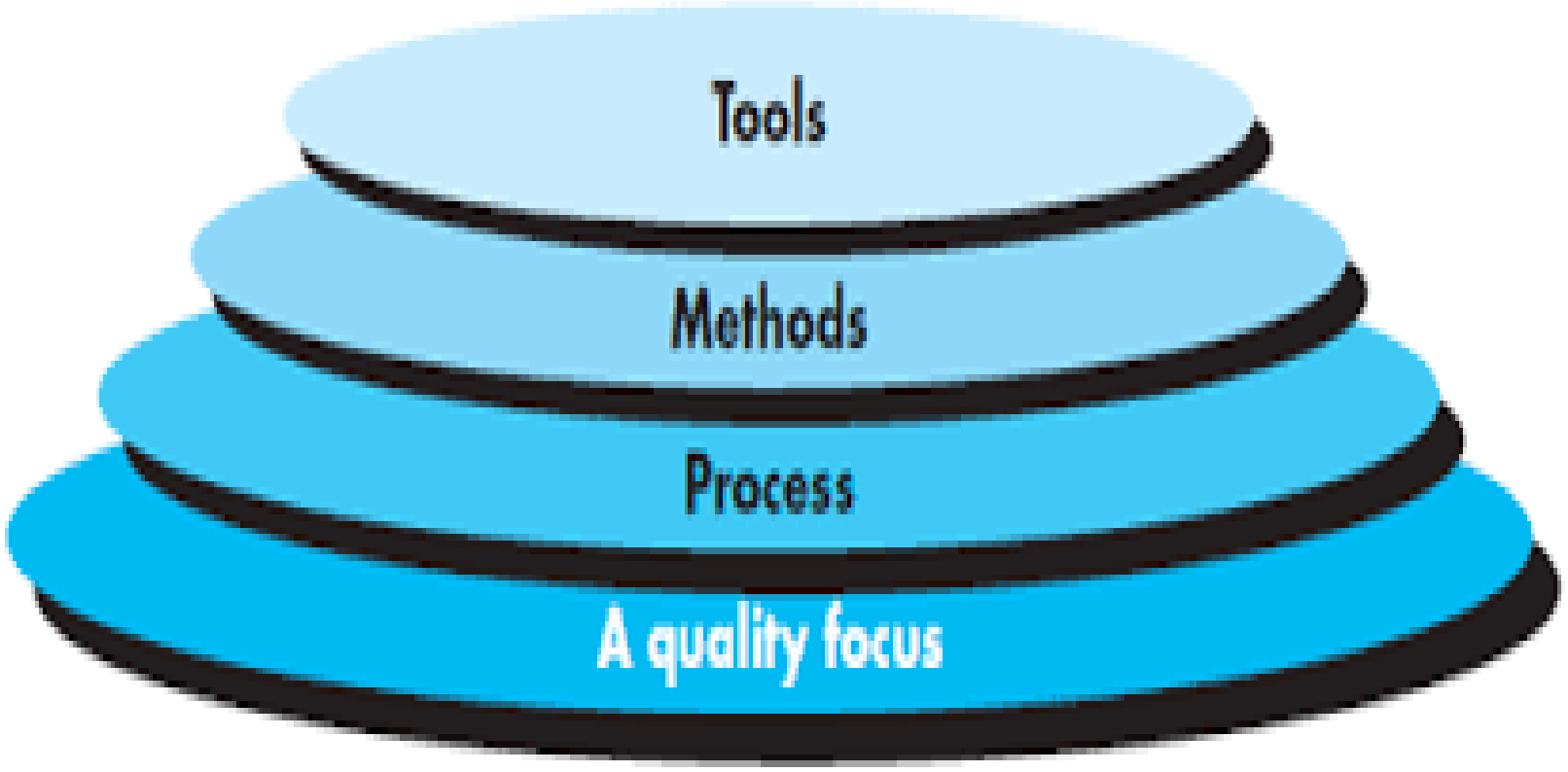
1.2 Software Engineering

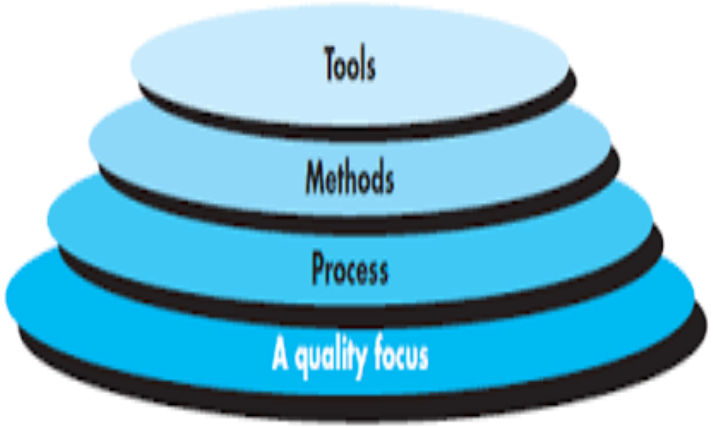
- Field of Computer Science
- Deals with building software systems which are so large
- IEEE defines SE as:
- Application and approaches of a systematic, disciplined, quantifiable approach to development, operation and maintenance of software.
- SE uses tools, methods and process with quality focus on system to be developed.

Brief History of Software Engineering

- The term “Software Engineering” was first coined in a NATO report in 1968.
- In the 1950s, software development was less emphasized. Software development was assigned to third party.
- Programmers moved for machine level to high level. To preserve the privacy, hardware manufacturers started to develop their own software themselves.
- In 1980s, the software cost of a system had risen to 80% and many experts pronounced the field “in crisis” because the software development face the problem of not been delivered on time, over budgeted, unmaintainable due to poor design and documentation, and unreliable due to poor system analysis and testing.
- It was required to apply engineering approach for management, team organization, tools, theories, methodologies and techniques which finally gave rise to “Software Engineering”.

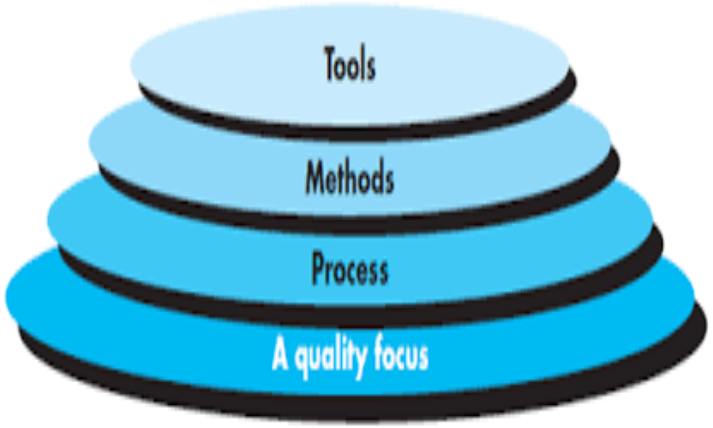
Software Engineering Layers





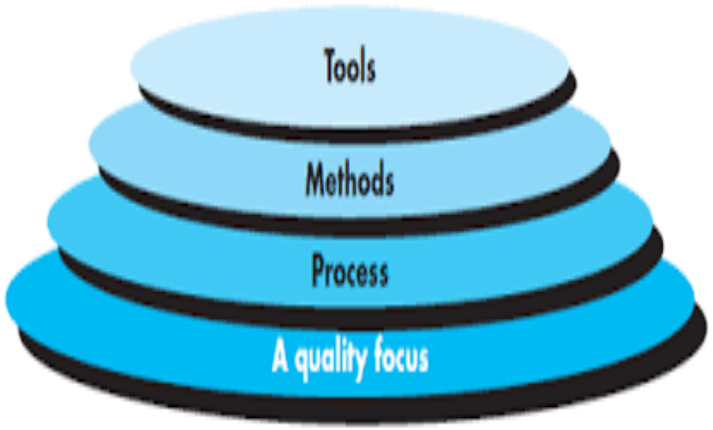
1. A quality Focus:

- Main principle of Software Engineering is Quality Focus.
- An engineering approach must have a focus on quality.
- Total Quality Management (TQM), Six Sigma, ISO 9001, ISO 9000-3, CAPABILITY MATURITY MODEL (CMM), CMMI & similar approaches encourages a continuous process improvement culture



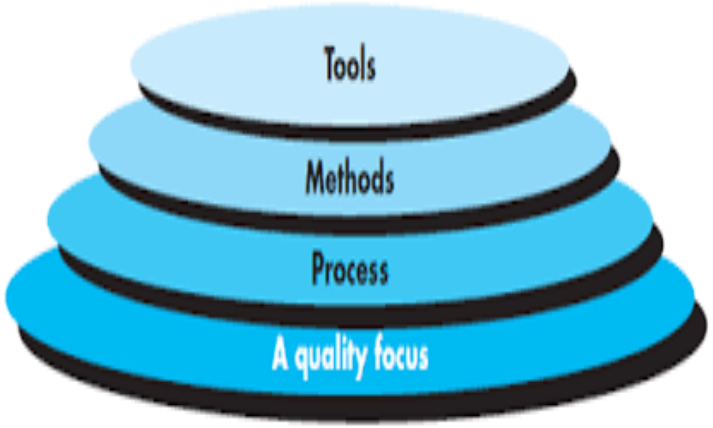
2. Process:

- It is a foundation of Software Engineering
- It is the glue the holds the technology layers
- It defines a framework with activities for effective delivery of software engineering technology



3. Methods:

- It provides technical how-to's for building software
- It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support



4. Tools:

- allows automation of activities which helps to perform systematic activities. A system for the support of software development, called computer-aided software engineering (CASE). Examples: Testing Tools, Bug/Issue Tracking Tools etc…
- Computer-aided software engineering (CASE) is the scientific application of a set of tools and methods to a software system which is meant to result in high-quality, defect-free, and maintainable software products
- CASE tools automate many of the activities involved in various life cycle phases

Role of SE

- SE researches, designs and develops software systems to meet with client requirements. Once the system has been fully designed, Software engineers then test, debug and maintain the system.
- SE translates vague (unclear) requirements and derives into precise specification.
- SE develops model for the system or application and understands the behavior and performance of the system.
- SE provides methods to schedule work, operate systems at various levels and obtain the necessary details of the software.
- SE promotes interpersonal and communication skills and management skills.

1.3 Types of Software

- **Classification on the basis of use of software:**
  1. System Software
  2. Application Software
     1. Generic software
     2. Tailored software
  3. Utility software

Software products

Software engineers are concerned with developing software products, that is, software that can be sold to a customer. There are two kinds of software product:

- **Generic products**
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- **Customized products**
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Product specification

- **Generic products**
  - The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.
- **Customized products**
  - The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.

1.4 Professional Software Development

- Most software development is a professional activity in which software is developed for business purposes, for inclusion in other devices, or as software products such as information systems and computer-aided design systems.
- Software engineering is intended to support professional software development rather than individual programming.
- It includes techniques that support program specification, design, and evolution, none of which are normally relevant for personal software development.

Professional Software Development

- This is one of the important differences between professional and amateur software development.
- If you are writing a program for yourself, no one else will use and you don’t have to worry about writing program guides, documenting the program design, and so on.
- However, if you are writing software that other people will use and other engineers will change, then you usually have to provide additional information as well as the code of the program.
- software is not just the programs themselves but also all associated documentation, libraries, support websites, and configuration data that are needed to make these programs useful.
- A professionally developed software system is often more than a single program.
- A system may consist of several separate programs and configuration files that are used to set up these programs.
- It may include system documentation, which describes the structure of the system, user documentation, which explains how to use the system, and websites for users to download recent product information.

1.5 Software Engineering Importance

- Software engineering is important for two reasons:
  1. More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
  2. It is usually cheaper, in the long run, to use software engineering methods and techniques for professional software systems rather than just write programs as a personal programming project. Failure to use software engineering method leads to higher costs for testing, quality assurance, and long-term maintenance

1.6 Software Process

- The systematic approach that is used in software engineering is sometimes called a software process.
- A software process is a sequence of activities that leads to the production of a software product.
- Four fundamental activities are common to all software processes:
  1. **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation.
  2. **Software development,** where the software is designed and programmed.
  3. **Software validation,** where the software is checked to ensure that it is what the customer requires.
  4. **Software evolution,** where the software is modified to reflect changing customer and market requirements.

- A road map
- Helps to create a timely and high quality result.

a) Communication	Understand objectives of the project, define software features and functions.
b) Planning	Initial study of: technical tasks to be conducted, risk that are likely, work schedule, etc.
c) Modeling	Create a sketch or models using Use-Case diagrams, DFD, ERD, etc.
d) Construction	Actual coding, testing, debugging and integration
e) Deployment	Delivery to customer and get feedback

1.7 Issues that affect software

1. **Heterogeneity:** Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
2. **Business and social change:** Businesses and society are changing incredibly quickly as emerging economies develop and new technologies become available.
3. **Security and trust:** It is essential that we can trust that software. We have to make sure that malicious users cannot successfully attack our software and that information security is maintained.
4. **Scale:** Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

1.8 Software Engineering Diversity

- There are many different types of software system like as
  - Stand-alone application,
  - Interactive transaction-based application
  - Embedded Control System
  - Batch processing systems
  - Entertainment systems, and so on.
- And, there is no universal set of software techniques that is applicable to all of these.
- The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.
- Each type of system requires specialized software engineering techniques because the software has different characteristics.
- For example, an embedded control system in an automobile is safety-critical. Such a system needs extensive verification and validation. It is burned into ROM (read-only memory) when installed in the vehicle, therefore very expensive to change.
- For an interactive web-based system or app, iterative development and delivery is the best approach, with the system being composed of reusable components.

SE Fundamental Principles

- Nevertheless, some fundamental principles apply to all types of software system, irrespective of the development techniques used are:
  - Systems should be developed using a managed and understood development process. The organization developing the software should plan the development process and have clear ideas of what will be produced and when it will be completed. Of course, different processes are used for different types of software.
  - Dependability and performance are important for all types of system. Software should behave as expected, without failures, and should be available for use when it is required.
  - Understanding and managing the software specification and requirements (what the software should do) are important.
  - Where appropriate, you should reuse software that has already been developed rather than write new software.

Application types

1. **Stand-alone applications**
   - These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
2. **Interactive transaction-based applications**
   - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
3. **Embedded control systems**
   - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.
4. **Batch processing systems**
   - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.

5. **Entertainment systems**
   - These are systems that are primarily for personal use and which are intended to entertain the user.
6. **Systems for modeling and simulation**
   - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
7. **Data collection systems**
   - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
8. **Systems of systems**
   - These are systems that are composed of a number of other software systems.

1.9 Software Engineering Ethics

- Like other engineering disciplines, software engineering is carried out within a social and legal framework that limits the freedom of people working in that area.
- As a software engineer, you must accept that your job involves wider responsibilities than simply the application of technical skills.
- You must also behave in an ethical and morally responsible way if you are to be respected as a professional engineer.
- It goes without saying that you should uphold normal standards of honesty and integrity.
- You should not use your skills and abilities to behave in a dishonest way or in a way that will bring disrepute to the software engineering profession.
- However, there are areas where standards of acceptable behavior are not bound by laws but by the more tenuous notion of professional responsibility.
- Some of these are:
  - Confidentiality
  - Competence
  - Intellectual property rights
  - Computer misuses

Areas of Professional responsibilities

1. **Confidentiality**
   - You should normally respect the confidentiality of your employers or clients regardless of whether or not a formal confidentiality agreement has been signed.
2. **Competence**
   - You should not misrepresent your level of competence.
   - You should not knowingly accept work that is outside your competence.
3. **Intellectual property rights**
   - You should be aware of local laws governing the use of intellectual property such as patents and copyright.
   - You should be careful to ensure that the intellectual property of employers and clients is protected.
4. **Computer misuse**
   - You should not use your technical skills to misuse other people's computers.
   - Computer misuse ranges from relatively trivial (game playing on an employer's machine) to extremely serious (dissemination of viruses or other malware).

ACM/IEE Code of ethics

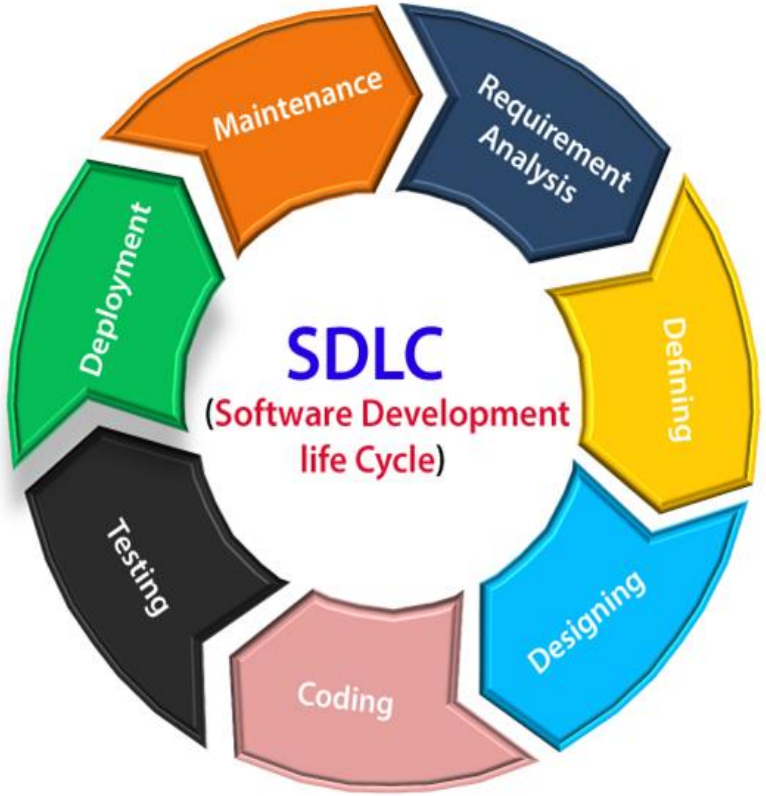
1. **PUBLIC** — Software engineers shall act consistently with the public interest.
2. **CLIENT AND EMPLOYER** — Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. **PRODUCT** — Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. **JUDGMENT** — Software engineers shall maintain integrity and independence in their professional judgment.
5. **MANAGEMENT** — Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. **PROFESSION** — Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. **COLLEAGUES** — Software engineers shall be fair to and supportive of their colleagues.
8. **SELF** — Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Essential attributes of Good software

Product characteristic	Description
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Software has to be secure so that malicious users cannot access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, resource utilization, etc.
Maintainability	Software should be written in such a way that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.

1.10 Software Development Life Cycle

1.10 Software Development Life Cycle

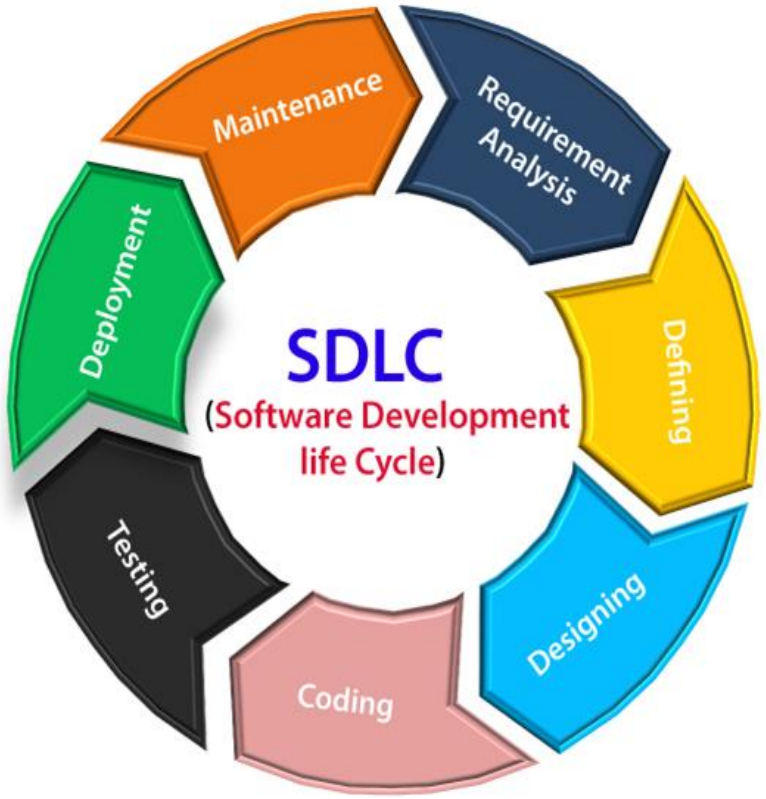


1. Planning and Requirement Analysis:

- What the customer wants to build? Gather it.
- Feasibility study
- Software Requirement Specification (SRS) developed.

2. Define Requirements:

- Requirements documented
- Get accepted from the project stakeholders.
- Refinement of requirements done.

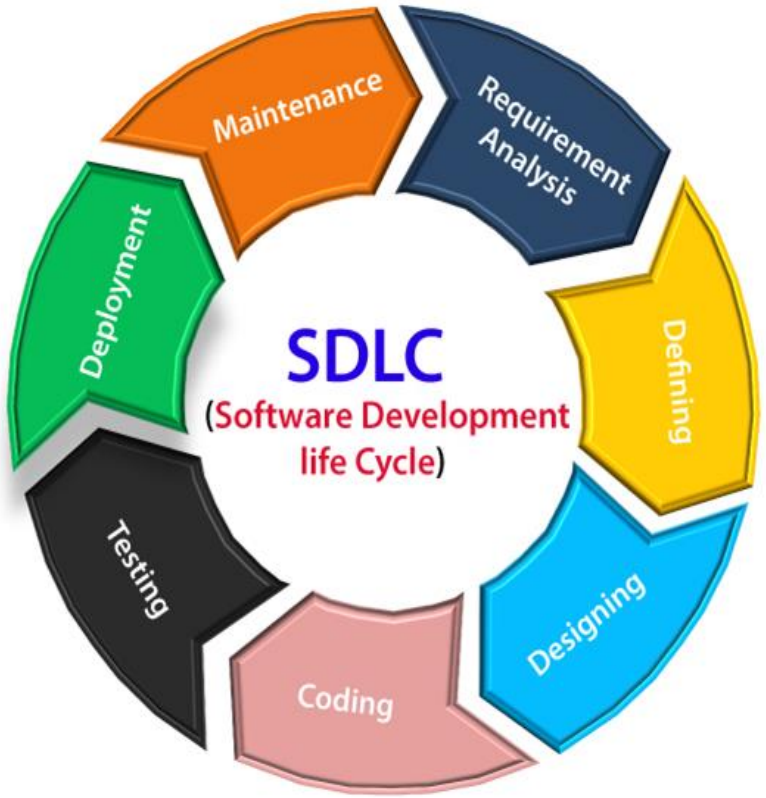


3. Design

- Various tool and artifacts used for designing of the software.
- Class diagram, components diagram, Usecase diagram and so on.

4. Coding:

- Actual coding done
- Selection of suitable programming language
- Follow the guidelines described by management.



5. Testing

- Different levels of testing done to solve the problems
  - ❑Unit testing
  - ❑Integration testing
  - ❑System testing
  - ❑Acceptance testing

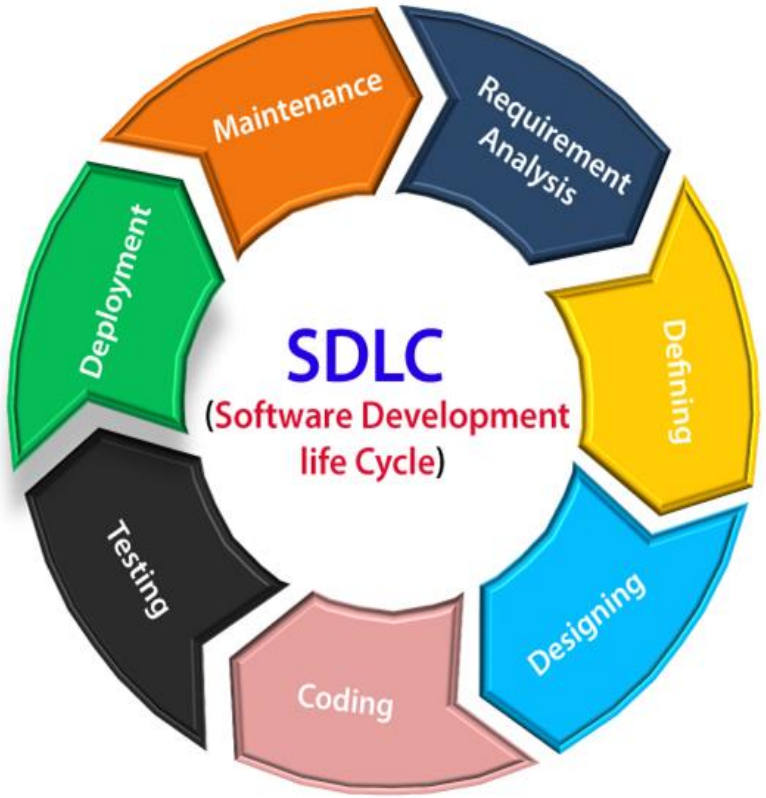
6. Deployment

- Software released
- Handed over to customer

7. Maintenance

Maintenance required due to:

- Technological changes
- Users desires
- Political changes
- Government policy changes
- Many more….

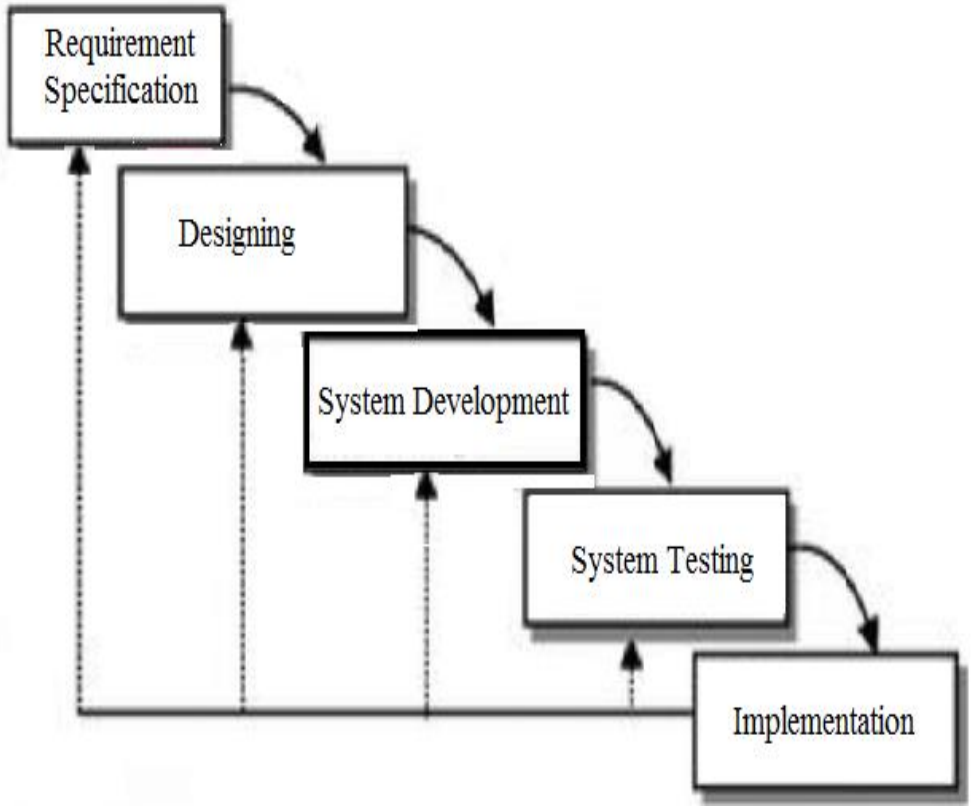


1.11 Perspective Process Models

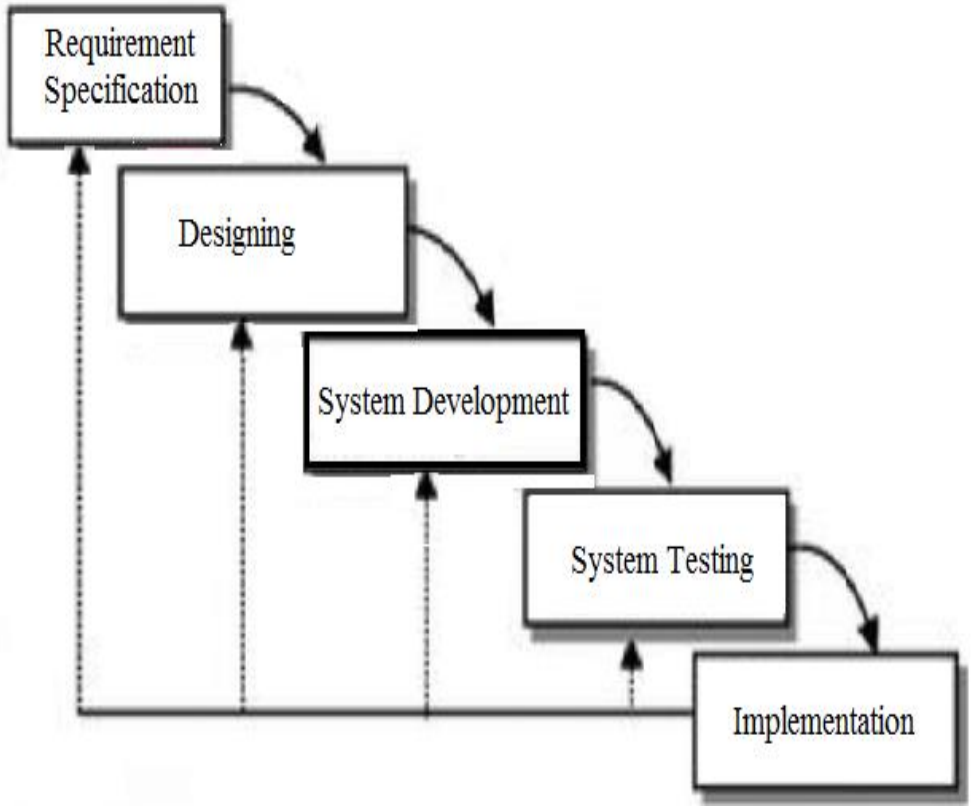
- Traditional models
- originally proposed to bring order to the vague of software development.
- they prescribe a set of process elements such as framework activities, software engineering actions, tasks, and work products, quality assurance, and change control mechanism for each project.
  - ❑Linear Sequential Model
  - ❑Waterfall Model
  - ❑RAD Model
  - ❑Spiral Model
  - ❑Incremental Model
  - ❑Evolutionary Model

i) Waterfall Model

- This model was first introduced by Dr. Winston W. Royce in a paper published in 1970.
- The waterfall model is a classical model used in system development life cycle to create a system with a linear and sequential approach.
- It is termed as waterfall because the model develops systematically from one phase to another in a downward fashion.
- This model is divided into different phases and the output of one phase is used as the input of the next phase.
- Every phase has to be completed before the next phase starts and there is no overlapping of the phases.

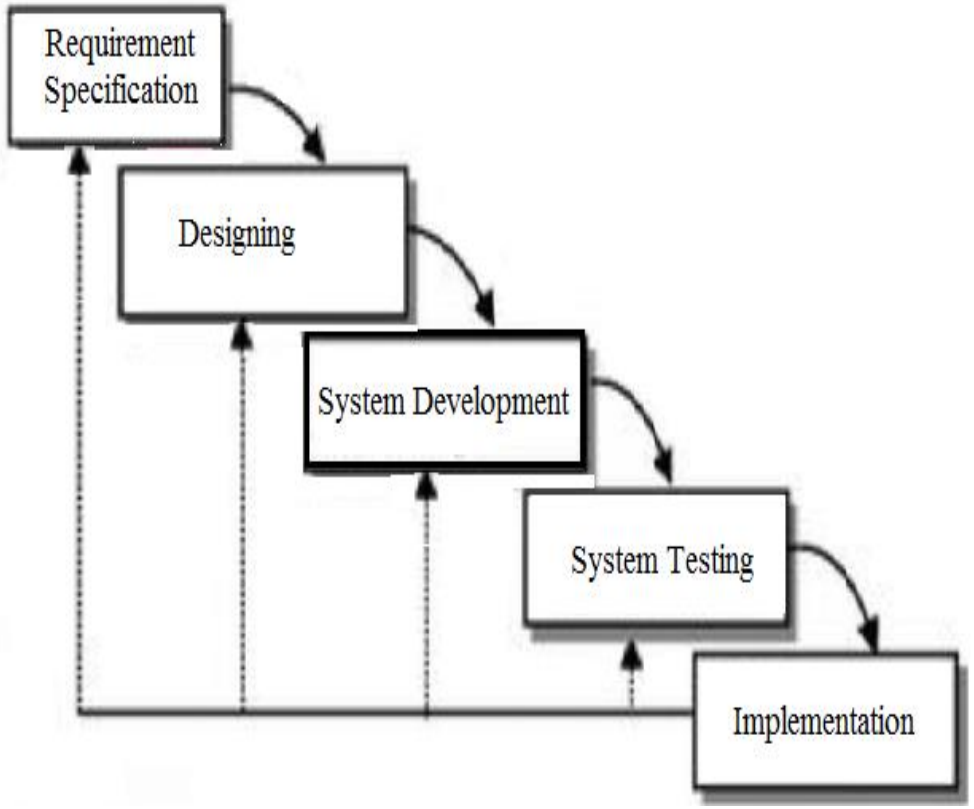


1. **Requirement Analysis and Specification**
   - This phase identifies and documents the exact requirements of the system through feasibility study.
   - This is done by combination of customers, developer and organization.



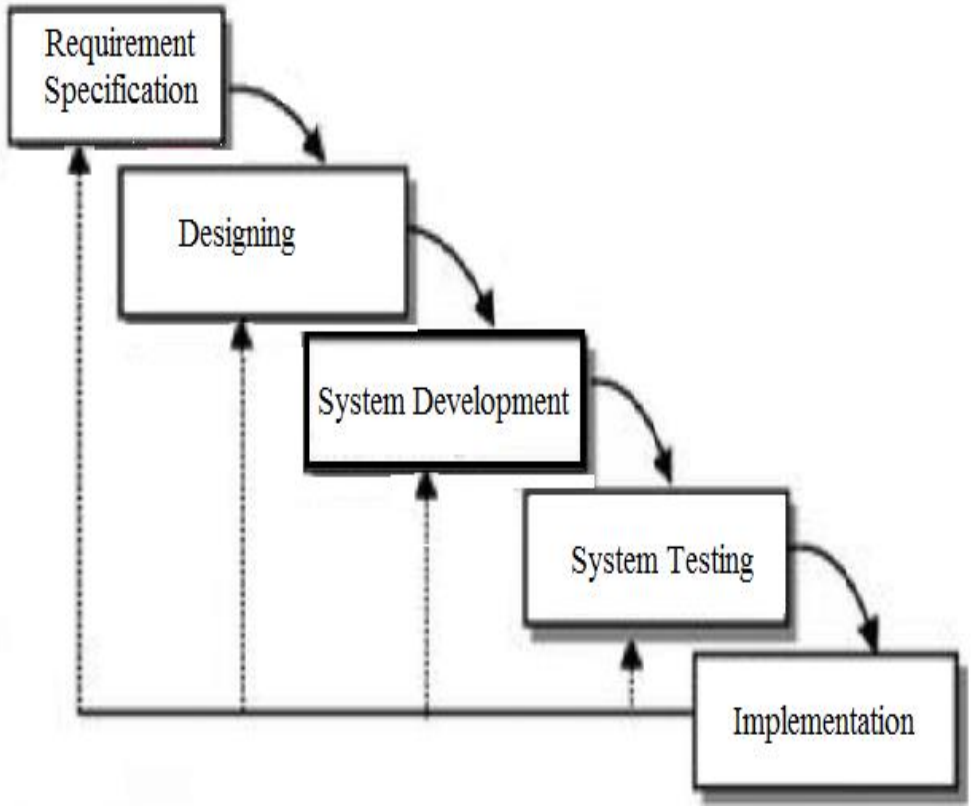
2. Design

- From the specified requirement, software engineers develop a design.
- It can be split into two phases:
- High Level Design and Detailed Design.
- The High-level design deals with the overall module structure and organization.
- Then, it is refined by designing each module (i.e. detailed design). Screen layout, business rules, process diagrams, DFD, ERD, etc. are used in this phase.



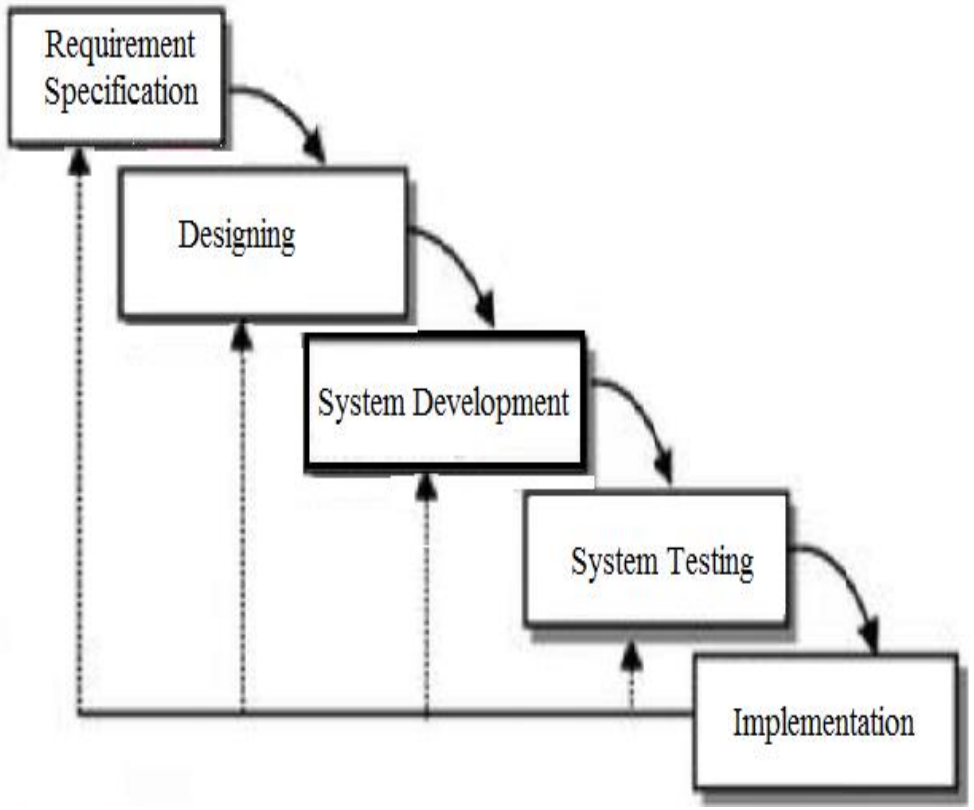
3. System Development

- Produces the actual code that will be developed to the customers.
- Also develops prototypes, modules, test drivers, etc.
- Testing is done at program modules and at various levels.
- Includes unit testing, black-box testing, white-box testing, etc.



4. System Testing

- All modules are tested individually in the previous phase are then integrated to make a complete system,
- and performance test is done on the whole system.



5. Implementation

- Once the system phases all the tests, it is delivered to the customers
- and enters the maintenance phase.
- Any modifications made to the system, after initial delivery, are usually attributed to this phase.

Advantages:

- Easy to understand and implement
- Suitable for simple project
- Well understood milestones.
- Process and results are well documented.

Disadvantages:

- Difficult to trace back
- No working software is produced until late during the life cycle.
- Not suitable for large product
- Risk analysis is not performed.
- Cannot accommodate changing requirements.
- Product may be outdated at the end.

When to use it?

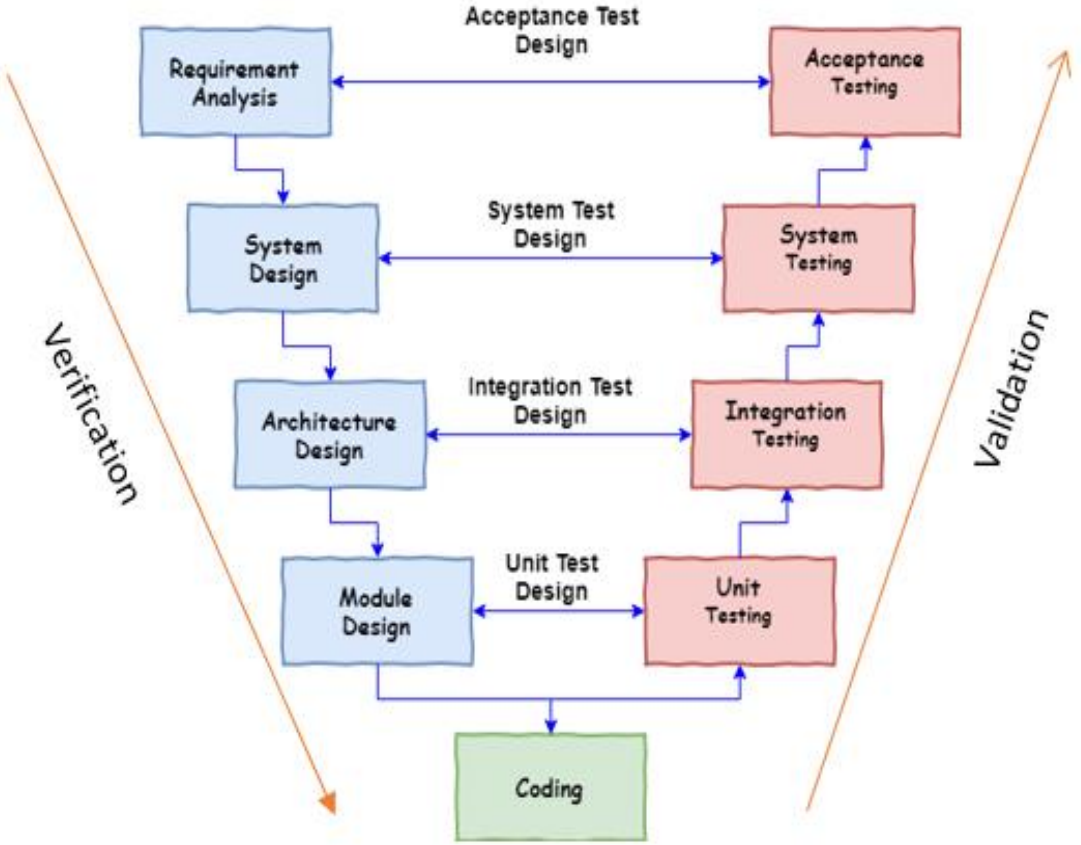
- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Major Drawback:

The major drawback of the Waterfall model is we move to the next stage only when the previous one is finished and there is no chance to go back if something is found wrong in later stages.

ii) The V Model

- The major drawback of the Waterfall model is we move to the next stage only when the previous one is finished and there is no chance to go back if something is found wrong in later stages.
- V-model provides means of testing of software at each stage in reverse manner.
- The V-Model is SDLC model where execution of processes happens in a sequential manner in V-shape.
- It is also known as Verification and Validation Model.
- V-Model is an extension of the Waterfall Model and is based on association of a testing phase for each corresponding development stage.
- This means that for every single phase in the development cycle there is a directly associated testing phase.
- This is a highly disciplined model and next phase starts only after completion of the previous phase.



- The left-hand side shows the development activities and the right-hand side shows the testing activities.
- In the development phase, both verification and validation are performed along with the actual development activities.
- Verification and Validation phases are joined by coding phase in V-shape.
- Thus it is called V-Model.
- This demonstrates that testing can be done in all phase of development activities as well

When to use?

- Where requirements are clearly defined and fixed.
- The V-Model is used when ample technical resources are available with technical expertise.
- Project is short.

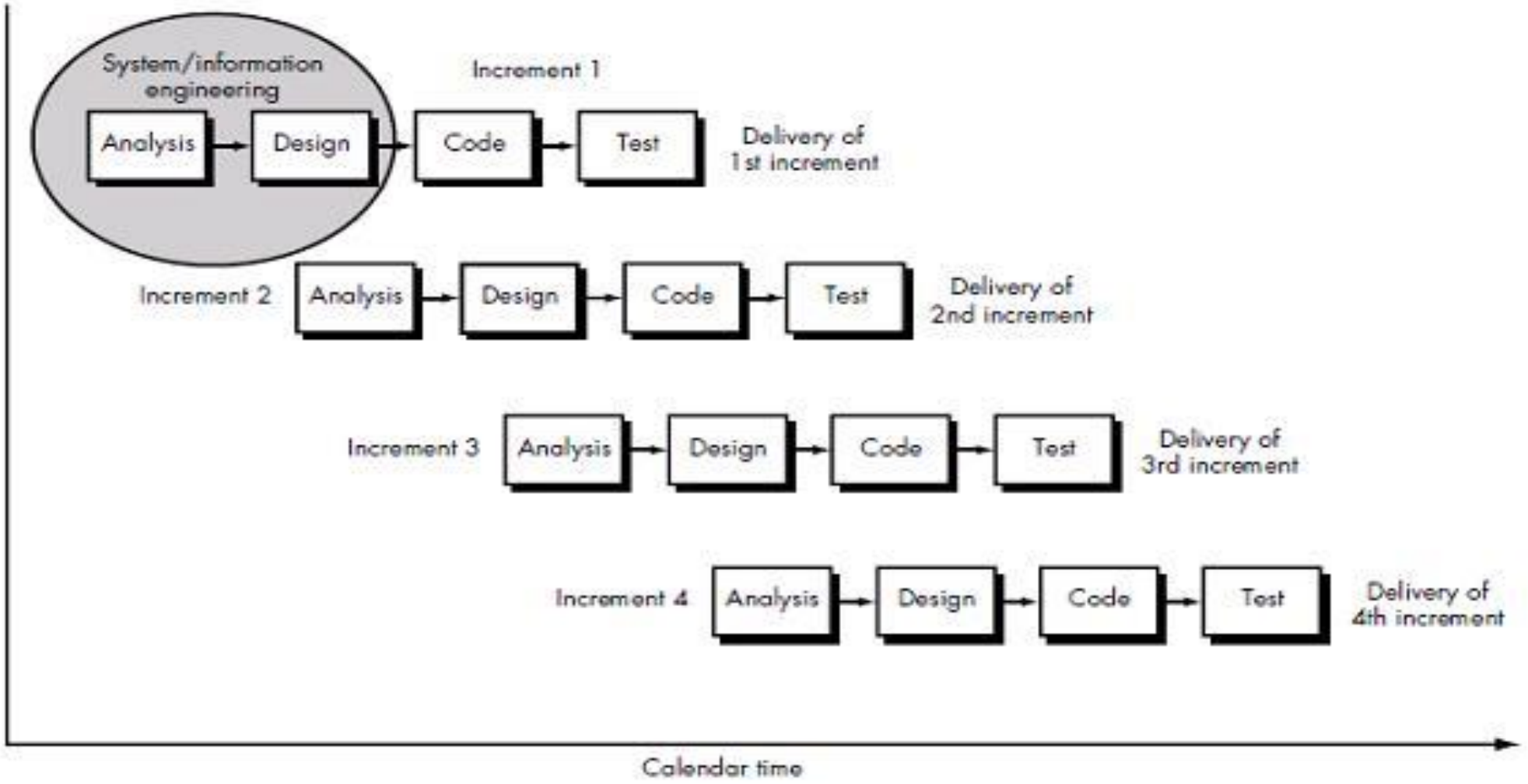
Advantages:

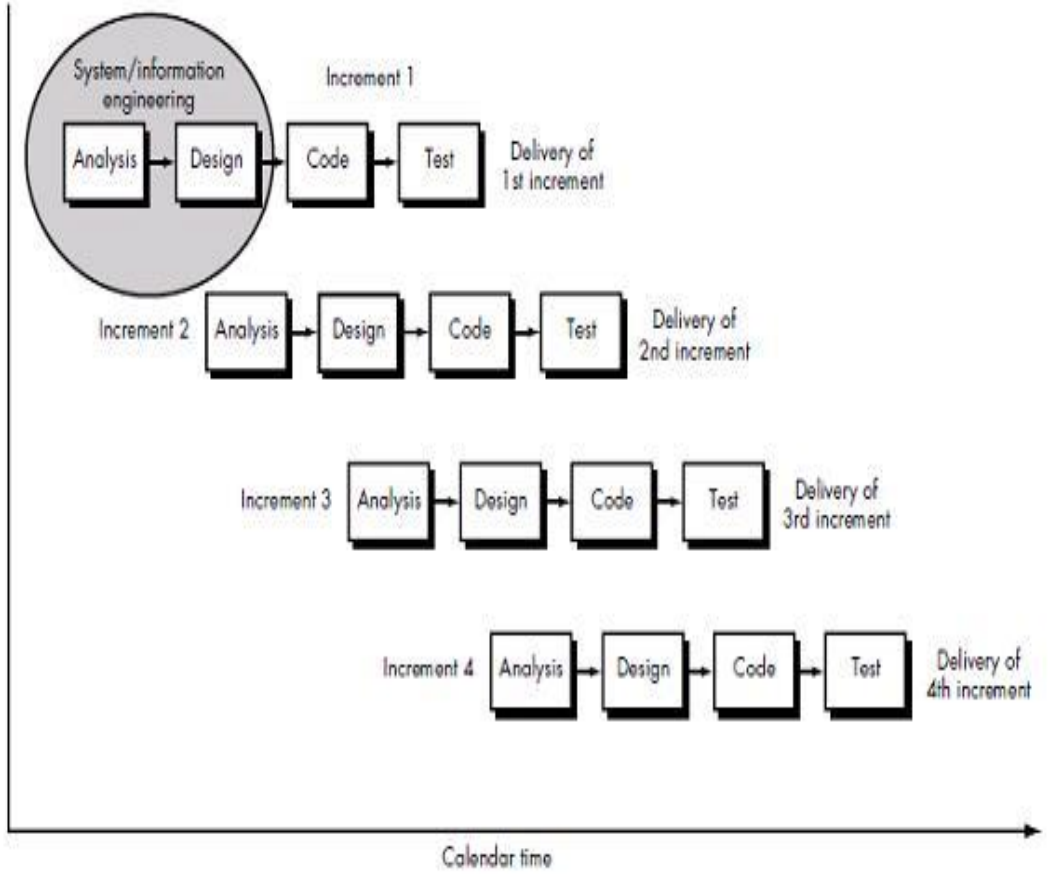
- This is a highly disciplined model and Phases are completed one at a time.
- V-Model is used for small projects where project requirements are clear.
- Simple and easy to understand and use.
- This model focuses on verification and validation activities early in the life cycle thereby enhancing the probability of building an error-free and good quality product.
- It enables project management to track progress accurately.

Disadvantages:

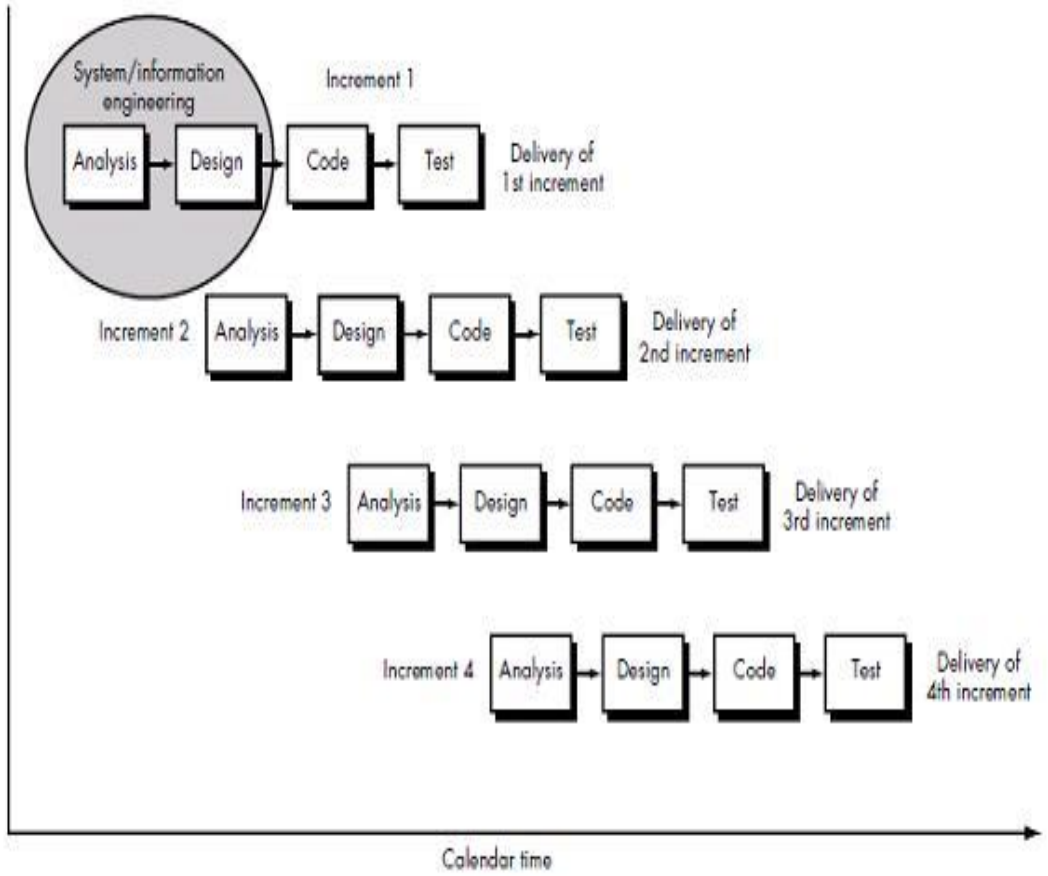
- High risk and uncertainty.
- It is not a good for complex and object-oriented projects.
- It is not suitable for projects where requirements are not clear and contains high risk of changing.
- This model does not support iteration of phases.
- It does not easily handle concurrent events.

iii) Incremental Model

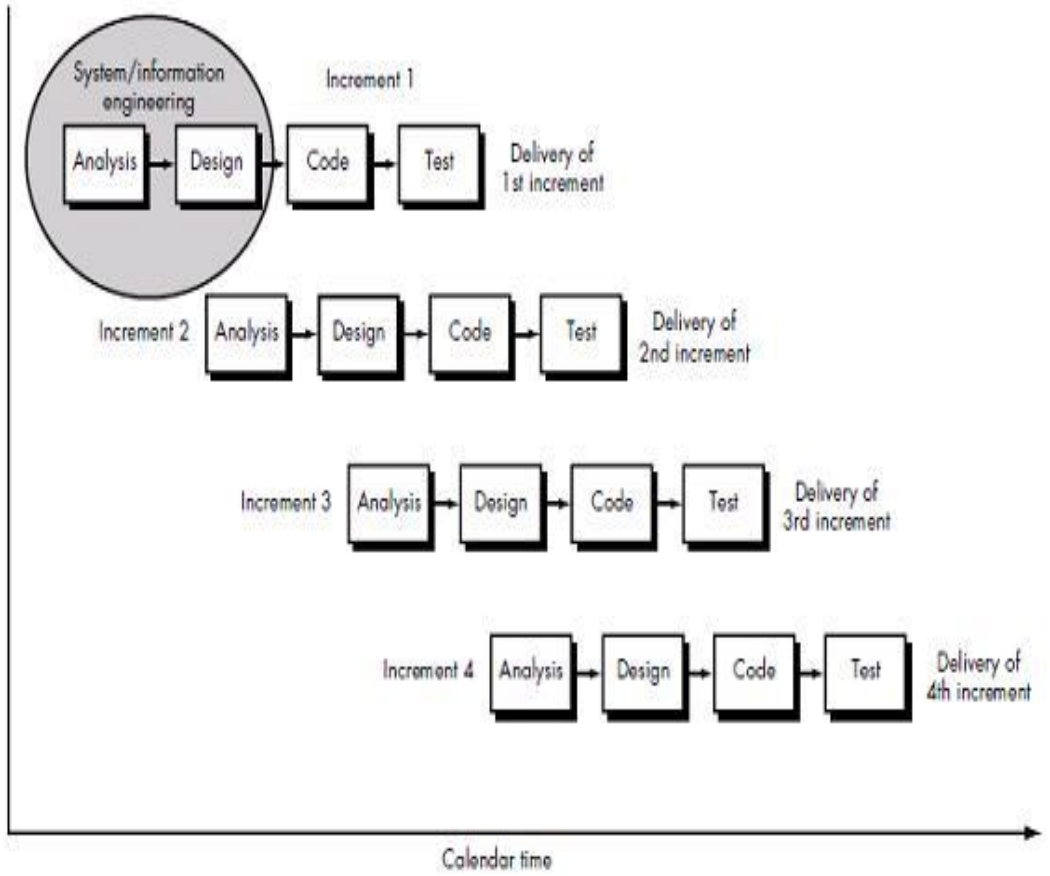




- product is designed, implemented and tested incrementally.
- Little more is added each time until the product is finished. It involves both development and maintenance.
- This is also known as the Iterative Model
- The basic idea of this model is that the software should be developed in increments, where each increment adds functional capability to the system until the full system is implemented.



- The product is decomposed into a number of components, each of which is designed and built separately.
- Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle.
- Cycles are divided up into smaller, more easily managed modules.
- Each module passes through the requirements, design, implementation and testing phases.



For example: word processing software developed using this model might deliver:

- file management, editing and printing in the first increment,
- most sophisticated editing and document production capabilities in the second increment,
- Spelling and grammar checking in the third increment and so on.

Advantages:

- Incremental Model allows partial utilization of the product and avoids a long development time.
- Supports changing environment
- Generates working software quickly and early during the software life cycle.
- This model is more flexible and less costly to change scope and requirements.
- It is easier to test and debug as smaller changes are made during each iteration.
- In this model, the customer can respond to each built. During the life cycle, software is produced early which facilitates customer evaluation and feedback
- As testing is done after each iteration, faulty elements of the software can be quickly identified because few changes are made within any single iteration.

Disadvantages:

- As additional functional is added to the product at every stage, problems may arise related to system architecture which was not evident in earlier stages.
- It needs good planning and design at every step, more management attention is required.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall, more resources may be required.
- End of project may not be known, which is a risk.

When to use Iterative Enhancement model?

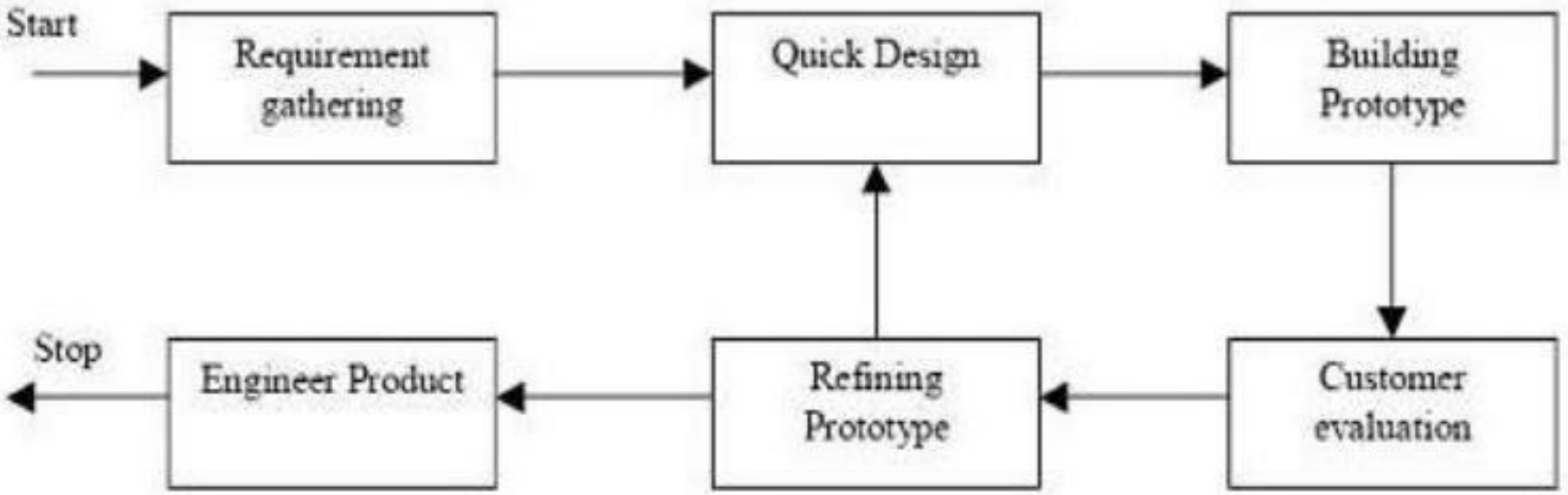
The below are the cases when we need to use Iterative Enhancement Model:

- Some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- New technology is being adapted.
- There are some high-risk features and goals which may change in the future.

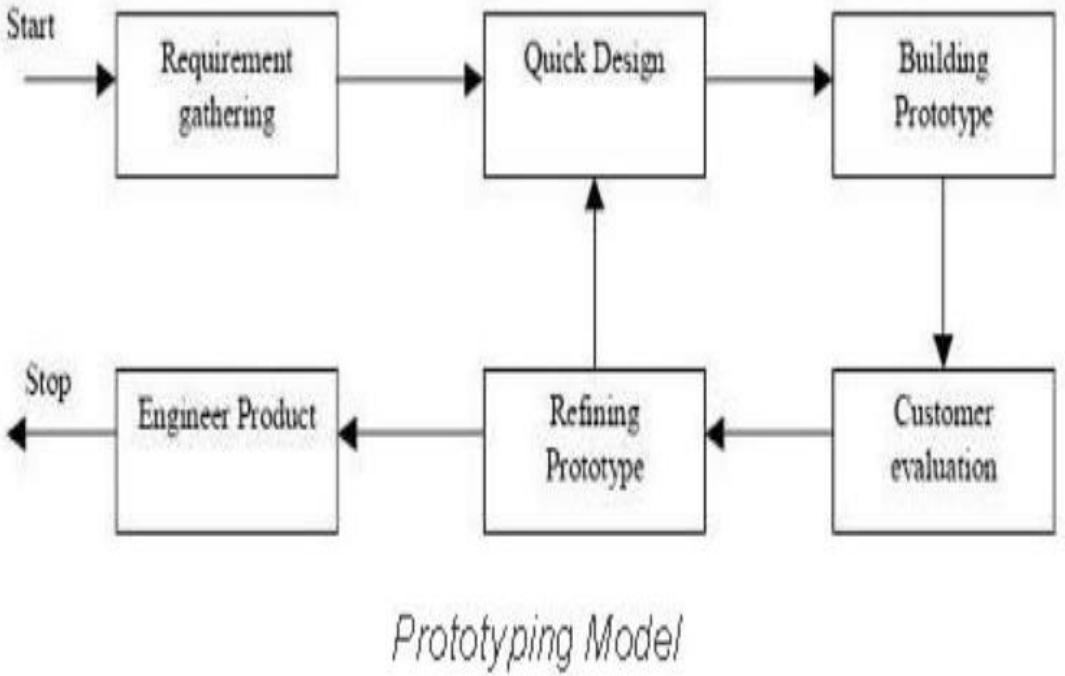
iv) Evolutionary Model

- Evolutionary models are iterative
- suitable for new systems where no clear idea of the requirements, inputs and outputs parameters.
- produces an increasingly more complete version of the software with each iteration.
- The two common evolutionary process models are:
  1. Prototyping model
  2. Spiral Model

a) Prototype Model



Prototyping Model



Prototyping Model

- prototype (an early approximation of a final system or product)
- In this model, a prototype is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved.
- Then, the complete system is developed.
- The analyst works with users to determine the initial or basic requirements for the system.
- The analyst then quickly builds a prototype and gives it to the user.
- The analyst uses the feedback to improve the prototype and takes the newer version back to users.
- The process is repeated until the user is relatively satisfied

Advantages:

- Users are actively involved in the development.
- Users get better understanding of the system being developed.
- Errors can be detected much earlier.
- Missing functionality can be identified quickly.
- Reduces project risk and less documentation.

Disadvantages:

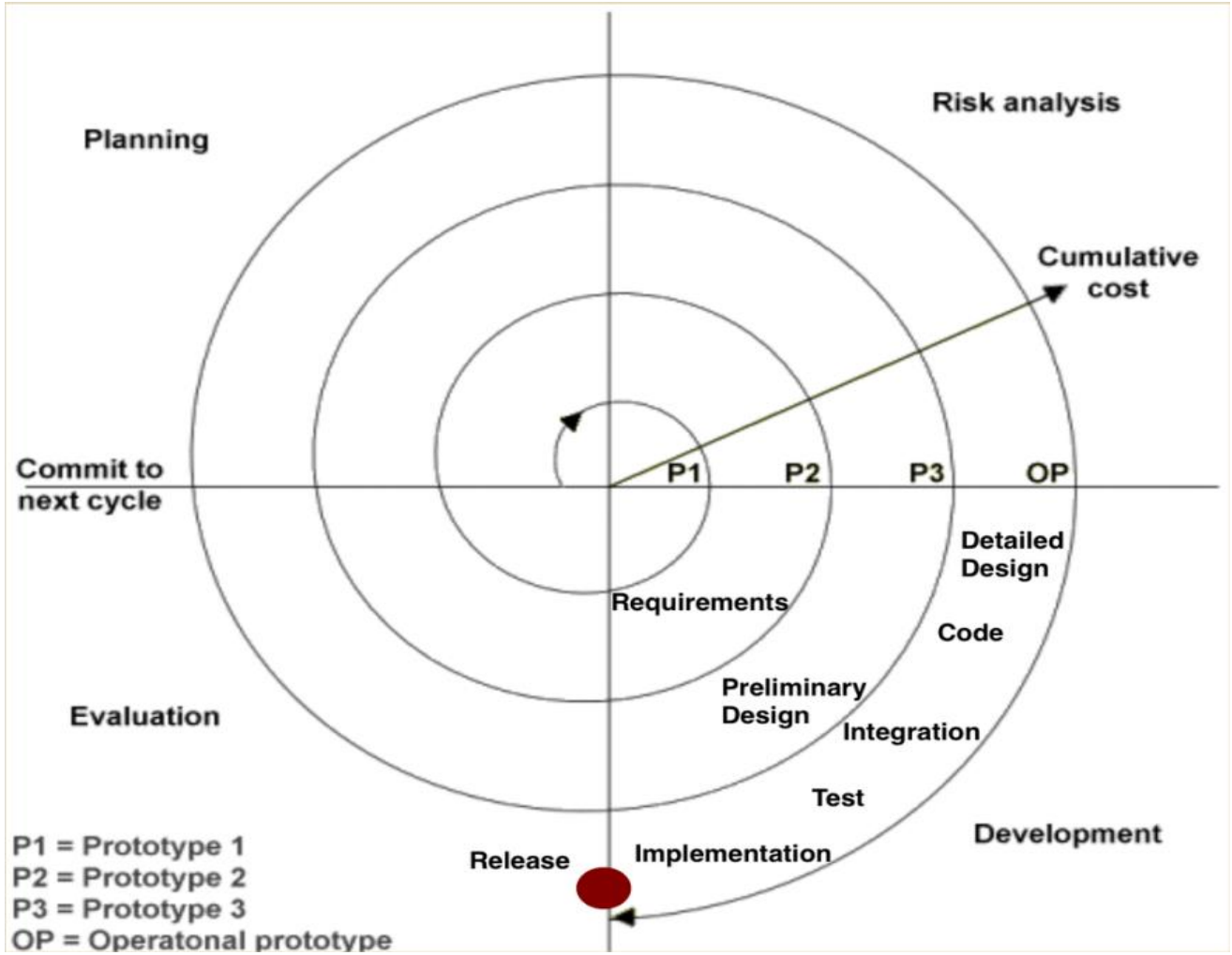
- May increase the complexity of the system since the user may demand more functionality,
- Integration can be very difficult.
- Frequent improvement and rebuilding of software.
- Slower process.

When to use Prototyping model?

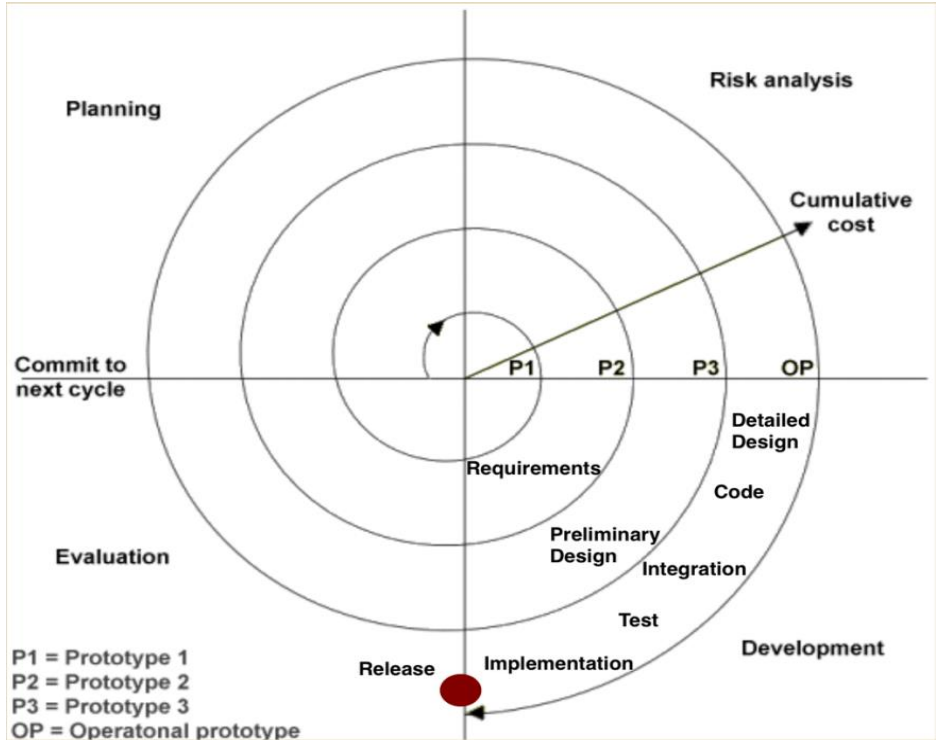
- When requirement is not clear.
- Useful in development of systems having high level of user interactions such as online systems.

b) Spiral Model

- This Spiral model, proposed by Barry Boehm (1985),
- is a combination of iterative development process model and sequential linear development model
- i.e. the waterfall model with a very high emphasis on risk analysis.
- It allows incremental releases of the product or incremental refinement through each iteration around the spiral.

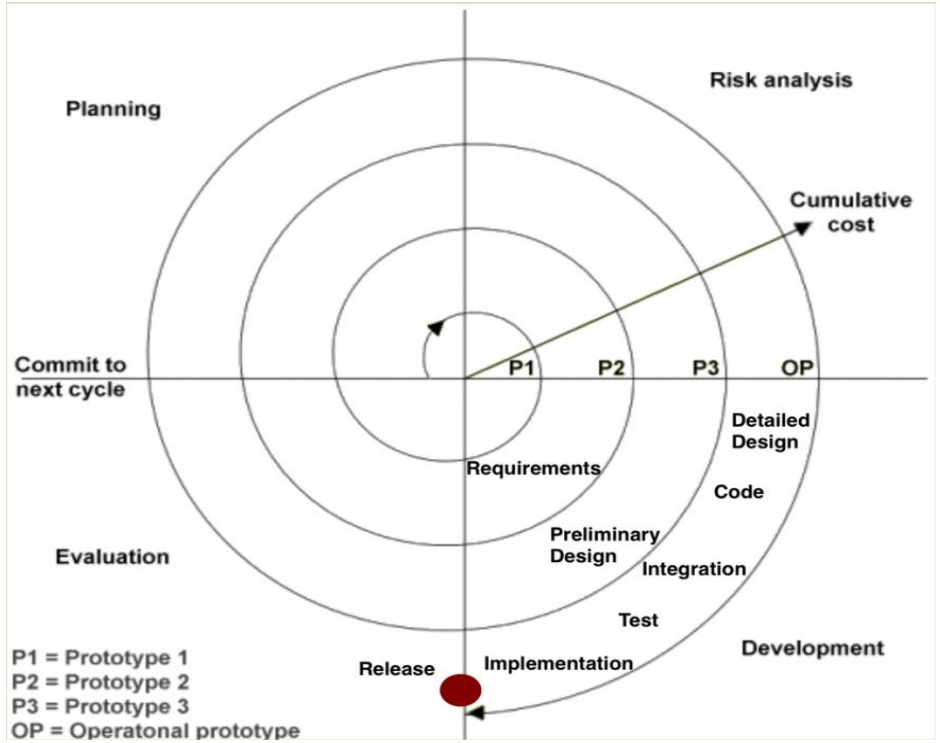


- The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.
- A software project repeatedly passes through these phases in each iteration/ spiral.



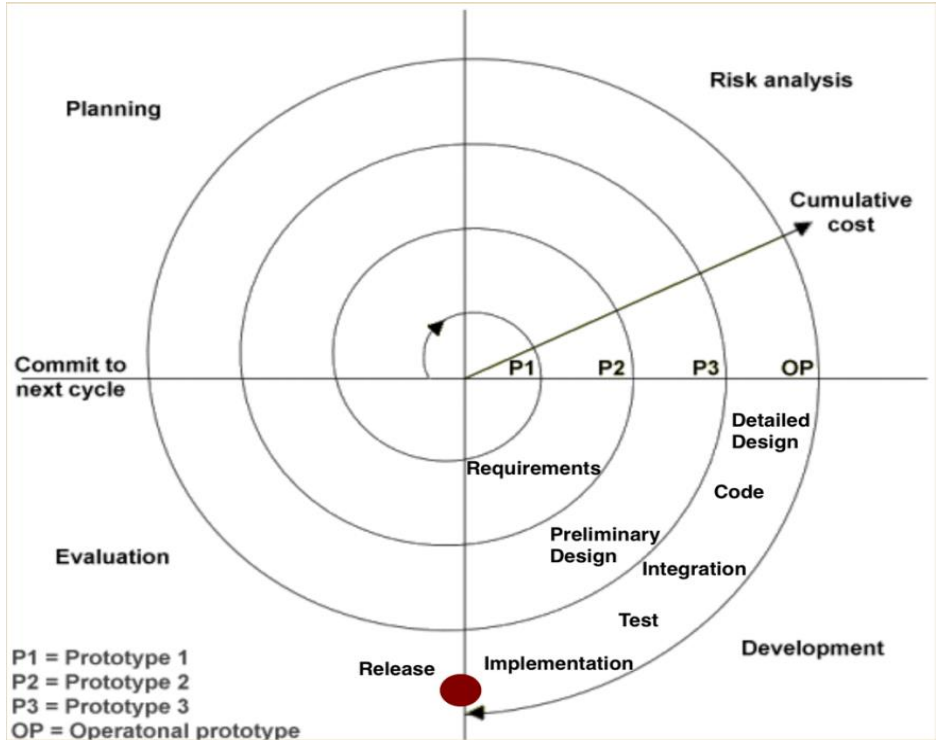
i) Planning:

- This phase starts with gathering the business requirements in the baseline spiral.
- In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.



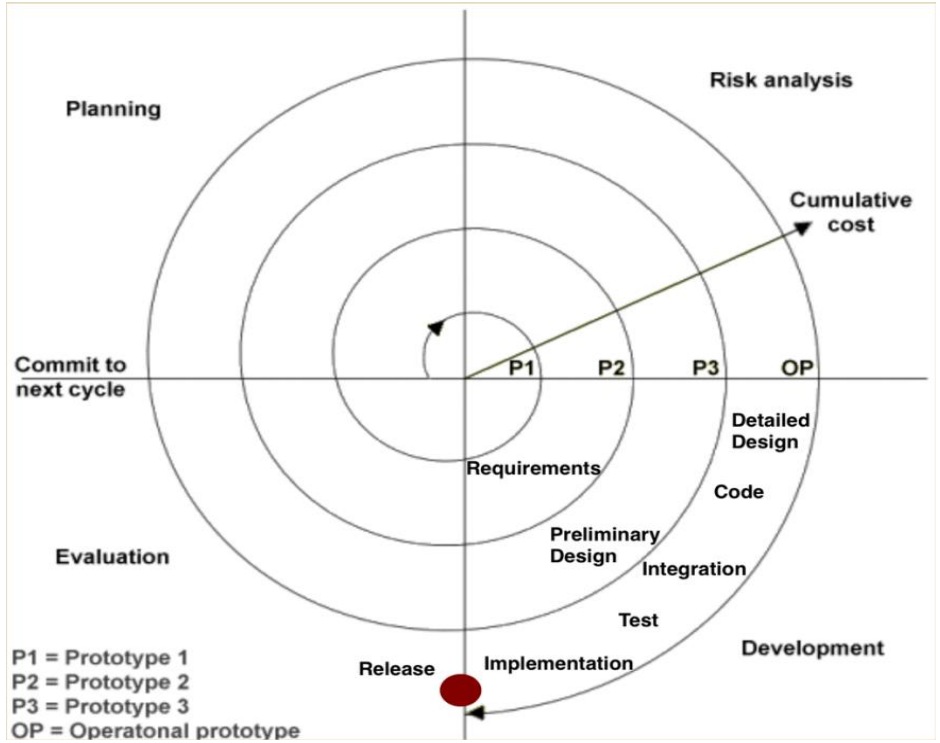
ii) Risk Analysis:

- Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks,
- such as schedule slippage and cost overrun.
- A prototype is produced at the end of the risk analysis phase.
- If any risk is found during the risk analysis then alternate solutions are suggested and implemented.



iii) Software Engineering / Development

- This phase refers to production of the actual software product at every spiral.
- Actual development and testing of the software takes place in this phase.
- It includes testing, coding and deploying software at the customer site.
- A POC (Proof of Concept) is developed in this phase to get customer feedback.
- In this phase software is developed, along with testing at the end of the phase.
- Hence in this phase the development and testing is done



iv) Evaluation:

- This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Advantages:

- Risk monitoring is one of the core parts which makes it pretty attractive, especially when you manage large and expensive projects. Moreover, such approach makes your project more transparent because, by design, each spiral must be reviewed and analyzed
- Customer can see the working product at the early stages of software development lifecycle
- Different changes can be added at the late life cycle stages
- Project can be separated into several parts, and riskier of them can be developed earlier which decreases management difficulties
- Project estimates in terms of schedule, costs become more and more realistic as the project moves forward, and loops in spiral get completed
- Strong documentation control

Disadvantages:

- Since risk monitoring requires additional resources, this model can be pretty costly to use. Each spiral requires specific expertise, which makes the management process more complex. That's why this SDLC model is not suitable for small projects
- A large number of intermediate stages. As a result, a vast amount of documentation
- Time management may be difficult. Usually, the end date of a project is not known at the first stages

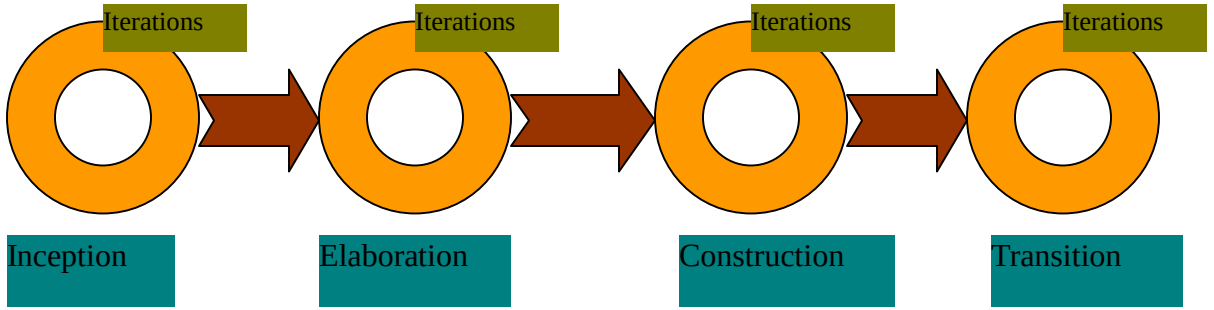
When to use Spiral model:

Spiral Model can be used in the below scenario:

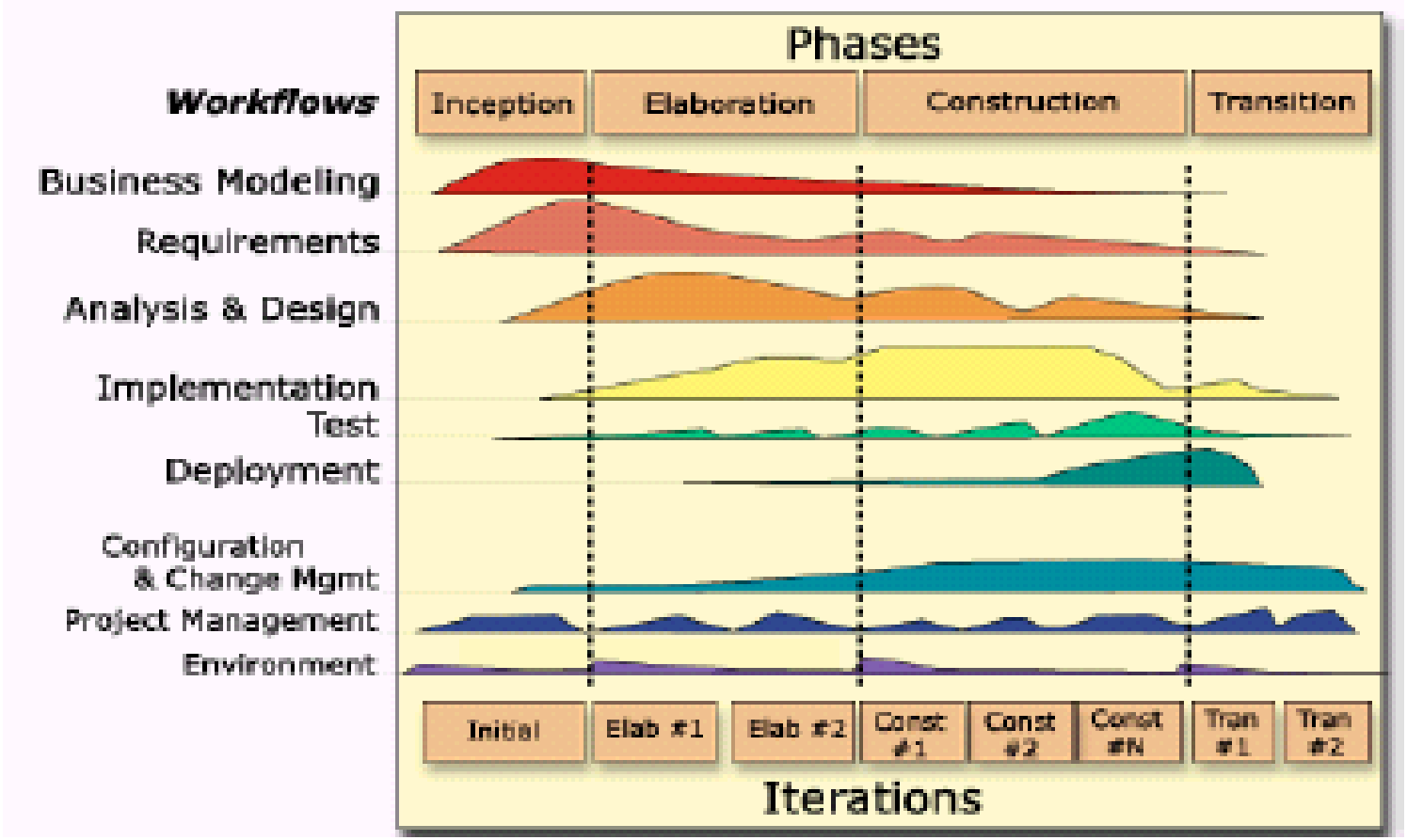
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- Significant changes are expected (research and exploration)

iv) Rational Unified Process (RUP) Model

- RUP is a complete software-development process framework , developed by Rational Corporation.
- It's an iterative development methodology based upon industry-proven best practices.
- Processes derived from RUP vary from lightweight—addressing the needs of small projects —to more comprehensive processes addressing the needs of large, possibly distributed project teams.
- RUP is divided into four phases, named:
  - Inception
  - Elaboration
  - Construction
  - Transition
- Each phase has iterations, each having the purpose of producing a demonstrable piece of software. The duration of iteration may vary from two weeks or less up to six months



The iterations and phases in RUP model are shown in below diagram:



RUP Overview Diagram

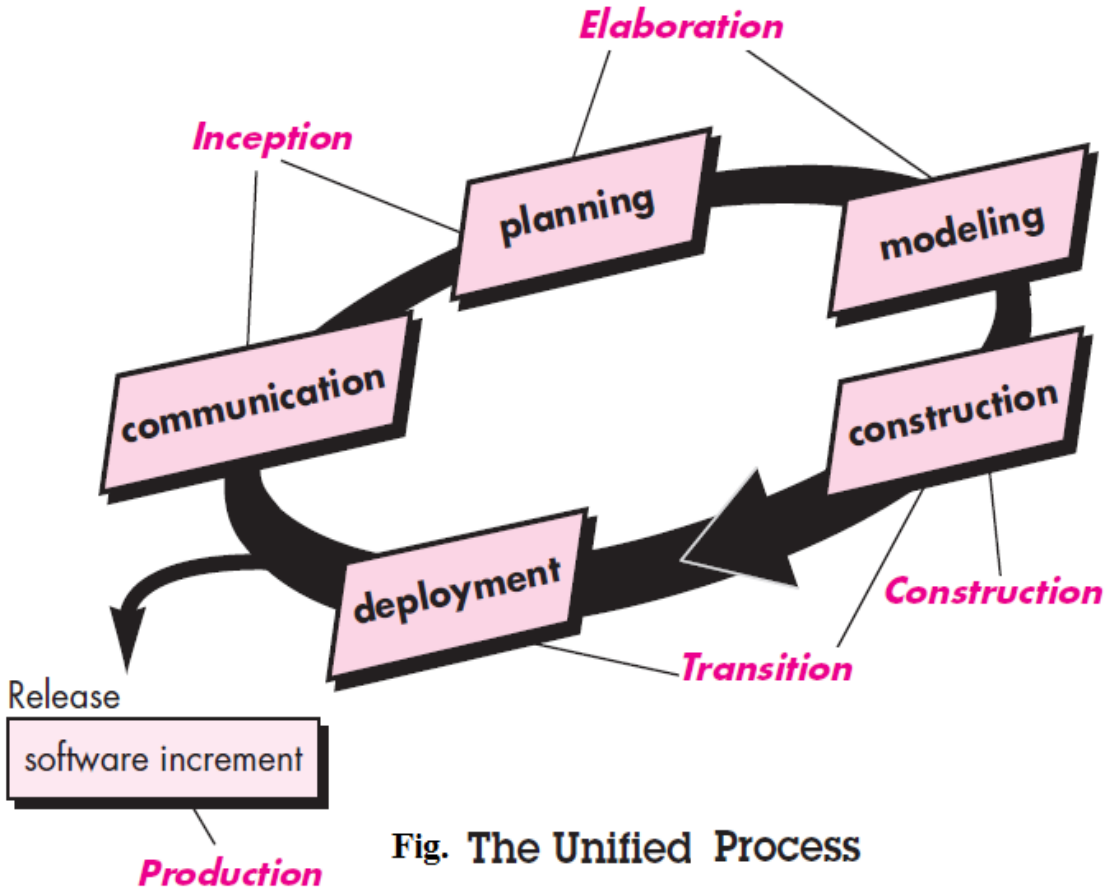


Fig. The Unified Process

- Unified process (UP) is an architecture-centric, use-case driven, iterative and incremental development process that leverages unified modeling language and is compliant with the system process engineering metamodel.
- Unified process can be applied to different software systems with different levels of technical and managerial complexity across various domains and organizational cultures.
- UP is also referred to as the unified software development process.

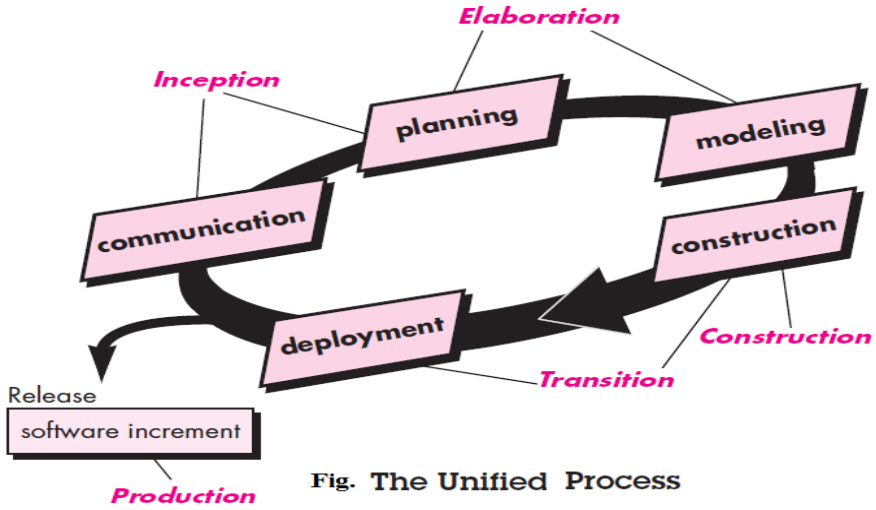


Fig. The Unified Process

1. **Inception**: The inception phase is similar to the requirements collection and analysis stage of the waterfall model of software development. In this phase, requirements are collected from the customer and analyze the project's feasibility, its cost, risks, and profits.

2. **Elaboration**: In this phase, the activities undertaken in the inception phase are expanded. The major goals of this phase include creating fully functional requirements (use-cases) and creating a detailed architecture for fulfillment of the requirements.

3. **Construction**: In this phase, actual code is written and the features are implemented for each iteration. The first iteration of the software is rolled out depending on the key use-cases that make up the core functionalities of the software system.

4. **Transition**: In this phase, next iterations are rolled out to the customer and fix bugs for previous releases. Finally, builds of the software is deployed to the customer.

FAQ

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

FAQ

What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. There are no methods and techniques that are good for everything.
What differences has the Internet made to software engineering?	Not only has the Internet led to the development of massive, highly distributed, service-based systems, it has also supported the creation of an "app" industry for mobile devices which has changed the economics of software.

Key Points

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Software is not just a program or programs but also includes all electronic documentation that is needed by system users, quality assurance staff, and developers. Essential software product attributes are maintainability, dependability and security, efficiency, and acceptability.
- The software process includes all of the activities involved in software development. The high-level activities of specification, development, validation, and evolution are part of all software processes.
- There are many different types of system, and each requires appropriate software engineering tools and techniques for their development. Few, if any, specific design and implementation techniques are applicable to all kinds of system.
- The fundamental ideas of software engineering are applicable to all types of software system. These fundamentals include managed software processes, software dependability and security, requirements engineering, and software reuse.
- Software engineers have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues but should be aware of the ethical issues that affect their work.
- Professional societies publish codes of conduct that embed ethical and professional standards. These set out the standards of behavior expected of their members.

Exercise (from text book):

1. Explain why professional software that is developed for a customer is not simply the programs that have been developed and delivered.
2. What is the most important difference between generic software product development and custom software development? What might this mean in practice for users of generic software products?
3. Briefly discuss why it is usually cheaper in the long run to use software engineering methods and techniques for software systems.
4. Software engineering is not only concerned with issues like system heterogeneity, business and social change, trust, and security, but also with ethical issues affecting the domain. Give some examples of ethical issues that have an impact on the software engineering domain.
5. Based on your own knowledge of some of the application types, explain, with examples, why different application types require specialized software engineering techniques to support their design and development.
6. Explain why the fundamental software engineering principles of process, dependability, requirements management, and reuse are relevant to all types of software system.
7. Explain how electronic connectivity between various development teams can support software engineering activities.
8. Noncertified individuals are still allowed to practice software engineering. Discuss some of the possible drawbacks of this.

End of Chapter