

ATME College of Engineering, Mysuru

Department of CSE

Subject Presentation

BCS501 – Software Engineering & Project Management

Faculty: Ms. Kavyashree B L

Subject Details

Subject Code	BCS501	CIE Marks	50
Number of Lecture / Hours	3	Exam Marks	50
Total No. of Lecture Hours	45	Exam Hours	03

Course outcomes:

The students should be able to:

CO1: Differentiate process models to judge which process model has to be adopted for the given scenarios.

CO2: Derive both functional and nonfunctional requirements from the case study.

CO3: Analyze the importance of various software testing methods and agile methodology.

CO4: Illustrate the role of project planning and quality management in software development.

CO5: Identify appropriate techniques to enhance software quality.

SOFTWARE ENGINEERING & PROJECT MANAGEMENT(BCS501)

Module-1
Software and Software
Engineering

TEXT BOOKS

- 1. Roger S. Pressman: Software Engineering-A Practitioners approach, 7th Edition, Tata McGraw Hill.
- 2. Bob Hughes, Mike Cotterell, Rajib Mall: Software Project Management, 6th Edition, McGraw Hill Education, 2018.

What is software engineering?

- The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
- **Software engineering** is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.
- **Software engineering** is also:-
 - A modeling activity
 - A problem-solving activity
 - A knowledge acquisition activity
 - A rationale-driven activity

➤ **Software engineers** should adopt

- Systematic and organized approach to their work
- Use appropriate tools and techniques depending on the problem to be solved
- The development constraints and the resources available

➤ **Software Engineering** is the science and art of building (designing and writing programs) a software systems that are:

- On time
- On budget
- With acceptable performance
- With correct operation

Software Products:

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

Software products may be:

Generic

Bespoke

What are the attributes of good software?

- Maintainability
- Dependability
- Efficiency
- Usability

The Nature Of Software

➤ Product:

As a product, it delivers the computing potential embodied by computer hardware or more broadly, by a network of computers that are accessible by local hardware.

➤ Vehicle for delivering a product:

As the vehicle used to deliver the product, software acts as the basis for the control of the computer , the communication of information , and the creation and control of other programs.

Some questions to software developer

➤ Why does it take so long to get software finished?

(Complexity, Scope Creep, Testing and Debugging, Team Coordination, Dependencies, User Feedback, Resource Constraints, Unexpected Issues)

➤ Why are development costs so high?

(Complexity of Requirements, Talent and Expertise, Development Time, Testing and Quality Assurance, Infrastructure and Tools, Maintenance and Support, Customization and Integration, Regulatory and Compliance Requirements)

Some questions to software developer

➤ Why can't we find all errors before we give the software to our customers?

(Complexity and Scale, Incomplete Testing Coverage, Dynamic Environments, Changing Requirements, Human Error, Complex Interactions, Resource Constraints, User Behavior)

➤ Why do we spend so much time and effort maintaining existing programs?

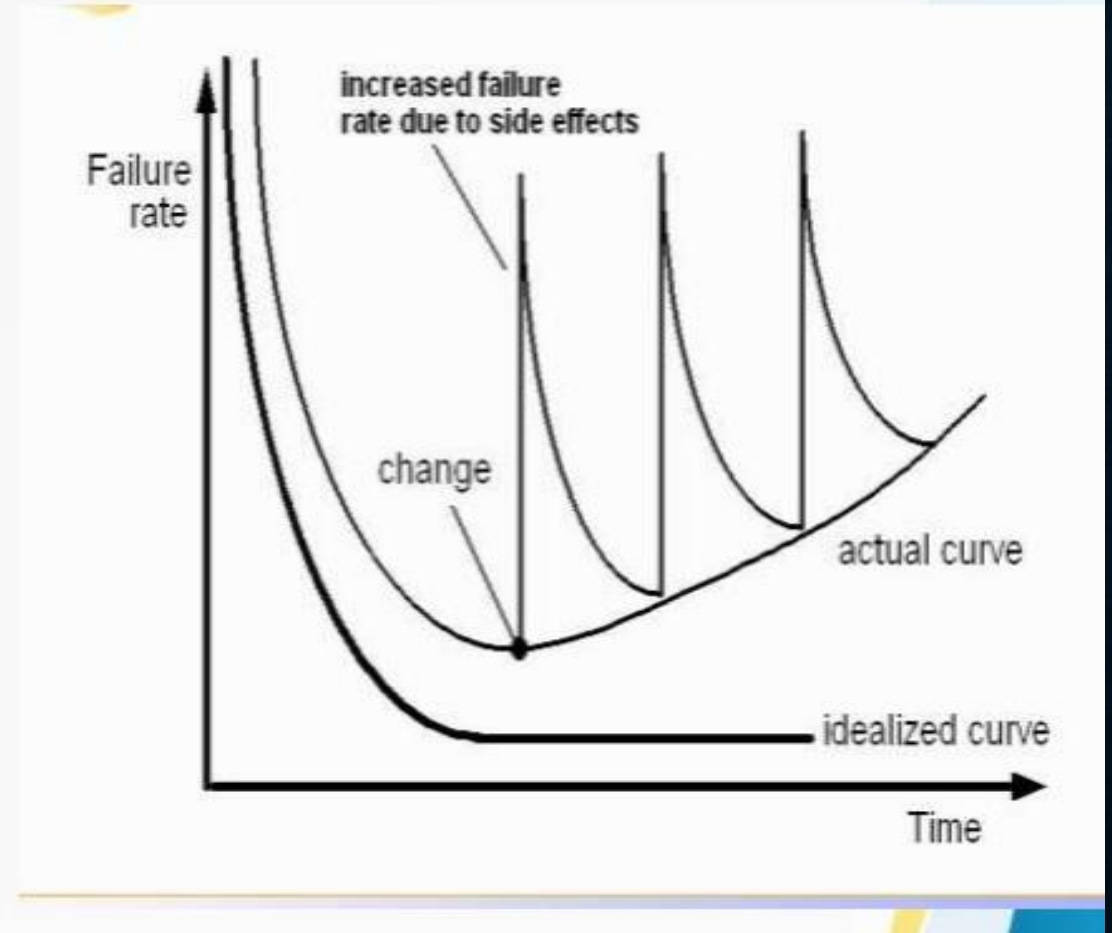
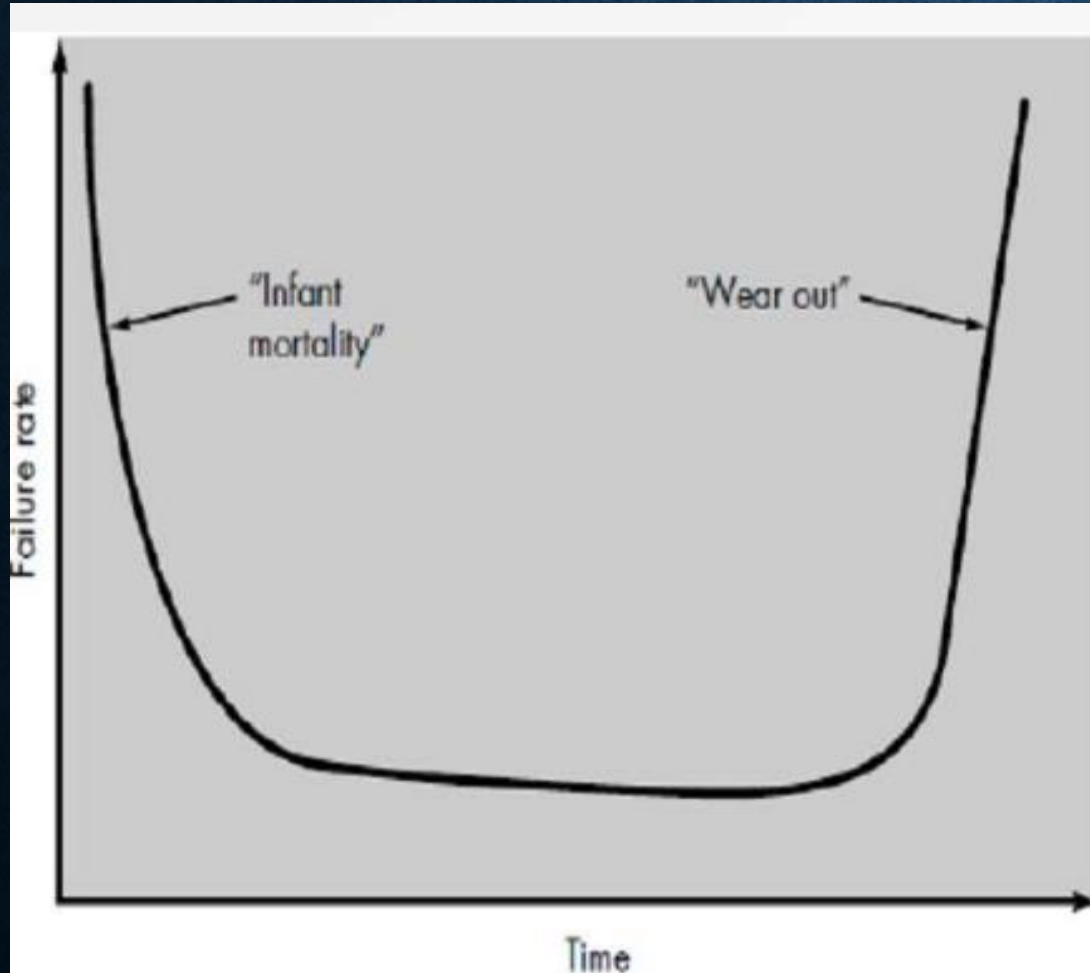
(Bug Fixes and Updates, Evolving Requirements, Security Patches, Compatibility Issues, Performance Optimization, User Support and Training, Regulatory and Compliance Changes, Documentation and Knowledge Transfer, Legacy Code, Interdependencies, User Feedback and Improvements)

Software Characteristics

- Software is developed or engineered, it is not manufactured in the classical sence.
- Although the industry is moving towards component based assembly, most software continues to be custom to built.

Software Characteristics

➤ Software doesn't "wear out".



Software applications Domains

- System Software
- Application Software
- Engineering/Scientific Software
- Embedded Software
- Product-Line Software
- Web Applications
- Artificial Intelligence Software

New Challenges for Software Developers

- Open-World Computing
- Netsourcing
- Open Source

Legacy Software

Dayani-Fard and his colleagues describe legacy software in the following way:

“Legacy software systems . . . were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms. The proliferation of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve.”

Legacy Software

Legacy systems often evolve for one or more of the following reasons:

- The software must be adapted to meet the needs of new computing environments or technology.
- The software must be enhanced to implement new business requirements.
- The software must be extended to make it interoperable with other more modern systems or databases.
- The software must be re-architected to make it viable within a network environment.

THE UNIQUE NATURE OF WEBAPPS

The following attributes are encountered in the vast majority of WebApps:

- Network intensiveness
- Concurrency
- Unpredictable load
- Performance
- Availability
- Data driven
- Content sensitive
- Continuous evolution
- Immediacy
- Security
- Aesthetics

Challenges Faced to Build Software

- Understand the problem before you build a solution.
- Design is a pivotal software engineering activity.
- Both quality and maintainability are an outgrowth of good design.

A Layered Technology



- **Software Engineering** is a layered technology.

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.

Process :-

- The foundation for software engineering is the process layer.
- Software engineering process is the glue that holds the technology layers together.
- Process defines a framework activities for effective delivery of software engineering technology.
- EX:Timely development of software,management & control of software projects.

Methods :-

- Software engineering methods provide the technical how-to's for building software.
- Methods encompass a broad array of tasks that include requirements analysis, design, program construction, testing, and support.

Tools :-

- Software engineering tools provide automated or semi-automated support for the process and the methods.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established.
- CASE combines software, hardware, and a software engineering database (a repository containing important information about analysis, design, program construction, and testing) to create a software engineering environment analogous to CAD/CAE (computer-aided design/engineering) for hardware.

SOFTWARE PROCESS

Software process:-

A Process is a collection of activities, actions and tasks that are performed when some work product is to be created.

Activity: An **activity** refers to a specific task or set of tasks that individuals or groups engage in to accomplish a particular goal or objective.

Ex: Communication with stake holder.

Action: In a general sense, an **action** refers to the process of doing something, often in pursuit of a goal or objective.

Ex: Architectural design.

Task: A **task** is a specific, individual piece of work or duty that needs to be completed as part of a larger project or goal.

Ex: Conducting a Unit test.

Common Process Framework Activities

COMMUNICATION



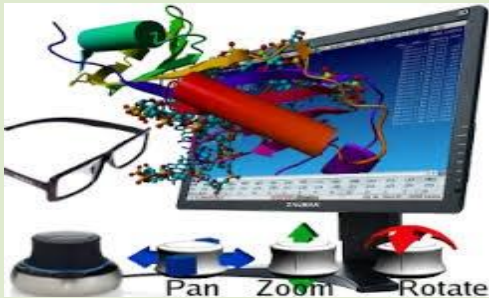
Communication with customers/ stakeholders to understand project requirements for defining software features.

PLANNING



Software Project Plan which defines workflow that is to follow. It describes technical task, risks, resources, product to be produced & work schedule.

MODELING



Creating models to understand requirements and shows design of software to achieve requirements.

CONSTRUCTIONS



Code generation (manual or automated) & testing (to uncover errors in the code)

DEPLOYMENT



Deliver Software to Customer collect feedback from customer based on evaluation Software Support

Umbrella Activities

Software project tracking and control

- ❑ Assessing progress against the project plan.
- ❑ Take adequate action to maintain schedule.

Formal technical reviews

- ❑ Assessing software work products in an effort to uncover and remove errors before goes into next action or activity.

Software quality assurance

- ❑ Define and conducts the activities required to ensure software quality.

Software configuration management

- ❑ Manages the effects of change.

Document preparation and production

- ❑ Help to create work products such as models, documents, logs, form and list.

Reusability management

- ❑ Define criteria for work product reuse
- ❑ Mechanisms to achieve reusable components.

Measurement

- ❑ Define and collect process, project, and product measures
- ❑ Assist the team in delivering software that meets customer's needs.

Risk management

- ❑ Assess risks that may affect the outcome of project or quality of product (i.e. software)

SOFTWARE ENGINEERING PRACTICE

In a classic book, *How to Solve It*, written before modern computers existed. George Polya [Pol45] outlined the essence of problem solving, and consequently, the essence of software engineering practice:

The Essence of Practice

- *Understand the problem* (communication and analysis).
- *Plan a solution* (modeling and software design).
- *Carry out the plan* (code generation).
- *Examine the result for accuracy* (testing and quality assurance).

SOFTWARE ENGINEERING PRACTICE

Understand the problem (communication and analysis).

- Who has a stake in the solution to the problem? That is, who are the stakeholders?
- What are the unknowns? What data, functions, and features are required to properly solve the problem?
- Can the problem be compartmentalized? Is it possible to represent smaller problems that may be easier to understand?
- Can the problem be represented graphically? Can an analysis model be created?

SOFTWARE ENGINEERING PRACTICE

Plan a solution (modeling and software design).

- Have you seen similar problems before?
- Has a similar problem been solved? If so, are elements of the solution reusable?
- Can subproblems be defined? If so, are solutions readily apparent for the subproblems?
- Can you represent a solution in a manner that leads to effective implementation? Can a design model be created?

SOFTWARE ENGINEERING PRACTICE

Carry out the plan (code generation).

- Does the solution conform to the plan? Is source code traceable to the designmodel?
- Is each component part of the solution provably correct? Have the design and code been reviewed, or better, have correctness proofs been applied to the algorithm?

Examine the result for accuracy (testing and quality assurance).

- Is it possible to test each component part of the solution?
- Does the solution produce results that conform to the data, functions, and features that are required?

SOFTWARE ENGINEERING PRACTICE

General Principles

- The First Principle: *The Reason It All Exists*
A software system exists for one reason: *to provide value to its users.*
- The Second Principle: *KISS (Keep It Simple, Stupid!)*
All design should be as simple as possible, but no simpler.
- The Third Principle: *Maintain the Vision*
A clear vision is essential to the success of a software project.
- The Fourth Principle: *What You Produce, Others Will Consume*
So, always specify, design, and implement knowing someone else will have to understand what you are doing.

SOFTWARE ENGINEERING PRACTICE

General Principles

➤ The Fifth Principle: Be Open to the Future

Never design yourself into a corner. Always ask “what if,” and prepare for all possible answers by creating systems that solve the general problem, not just the specific one.

➤ The Sixth Principle: Plan Ahead for Reuse

Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they are incorporated.

➤ The Seventh principle: Think!

Placing clear, complete thought before action almost always produces better results.

SOFTWARE MYTHS

Software myths are commonly held misconceptions about software development that can lead to misunderstandings and poor practices.

There are 3 types of Software Myths

- Management Myth
- Customer Myth
- Developers/Practioners Myth

SOFTWARE MYTHS

Management Myth

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?

Myth: If we get behind schedule, we can add more programmers and catch up(sometimes called the "Mongolian horde" concept).

Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

SOFTWARE MYTHS

Customer Myth

Myth: A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.

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

SOFTWARE MYTHS

Developers/Practioners Myth

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

Myth: Until I get the program “running” I have no way of assessing its quality.

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

Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

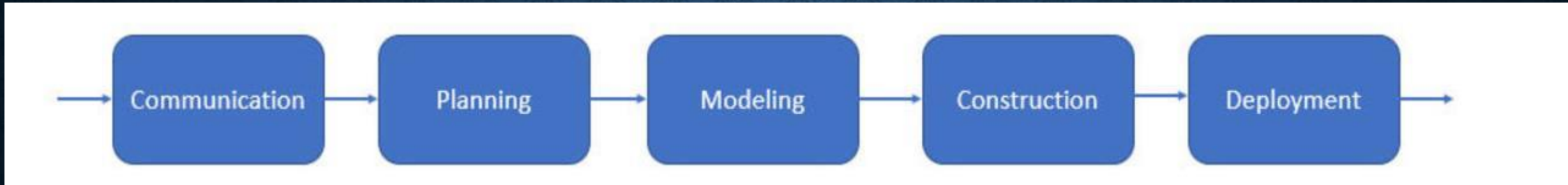
PROCESS FLOW

Process flow describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.

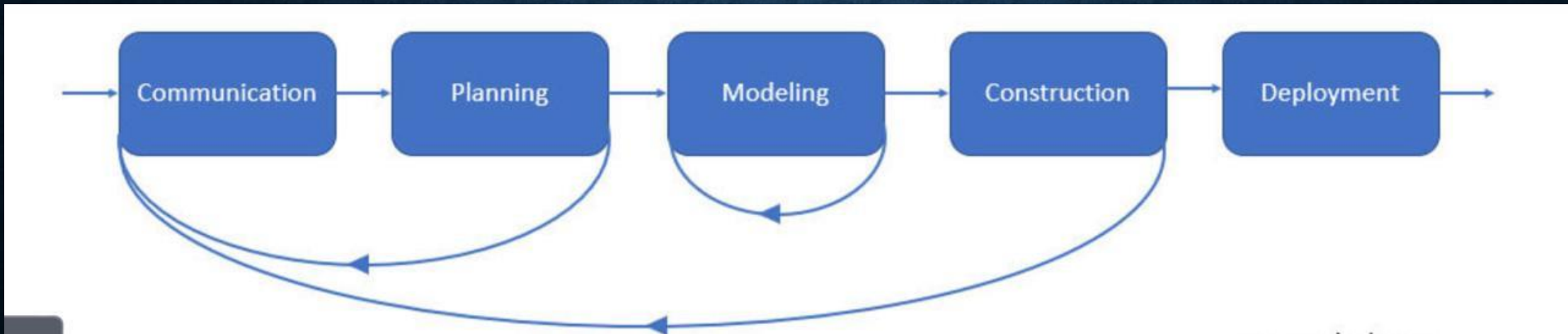
It is often represented visually through flowcharts or diagrams, which help clarify the process, making it easier to understand, analyze, and optimize.

Different types of Process flow available such as,

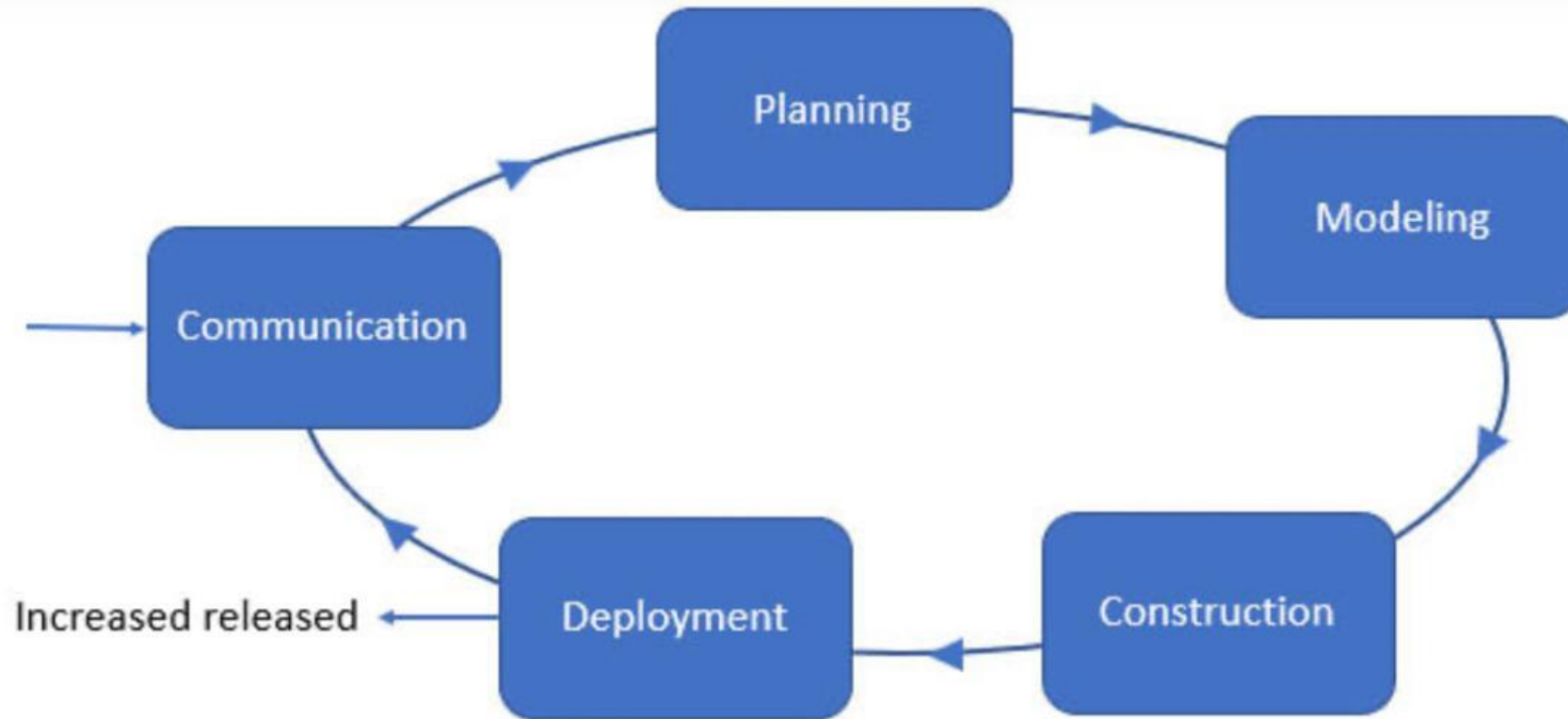
- **Linear process flow:** It executes each activity listed above in sequence form.



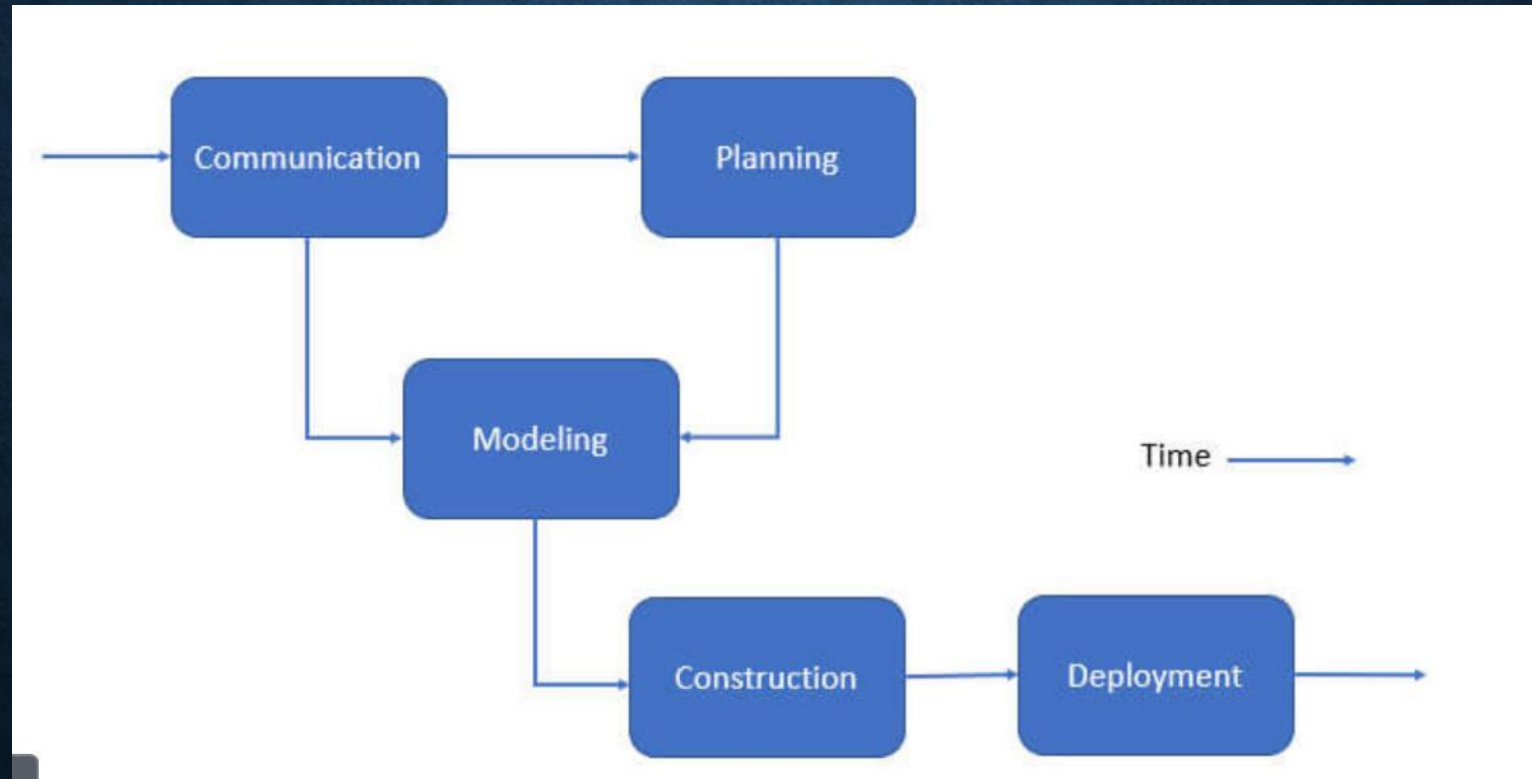
- **Iterative Process flow:** This flow repeats one or more activities that are listed above before starting the next activity.



- **Evolutionary Process flow:** This flow carries out the activities listed above in a circular manner. Each circle leads to the more complete version of the software



Parallel process flow: This flow executes the activities listed above in parallel with each other activities.



How does a framework activity change as the nature of the project changes?

1. Make contact with stakeholder via telephone.
2. Discuss requirements and take notes.
3. Organize notes into a brief written statement of requirements.
4. E-mail to stakeholder for review and approval.

IDENTIFYING A TASK SET

- A task set defines the actual work to be done to accomplish the objectives of a software engineering action.
 - A list of the task to be accomplished
 - A list of the work products to be produced
 - A list of the quality assurance filters to be applied

PROCESS PATTERNS

- A *process pattern*
 - describes a process-related problem that is encountered during software engineering work,
 - identifies the environment in which the problem has been encountered, and
 - suggests one or more proven solutions to the problem.
- Stated in more general terms, a process pattern provides you with a *template* [Amb98]—a consistent method for describing problem solutions within the context of the software process.

PROCESS PATTERN TYPES

- *Stage patterns*—defines a problem associated with a framework activity for the process.
- *Task patterns*—defines a problem associated with a software engineering action or work task and relevant to successful software engineering practice
- *Phase patterns*—define the sequence of framework activities that occur with the process, even when the overall flow of activities is iterative in nature.

PROCESS ASSESSMENT AND IMPROVEMENT

- **Standard CMMI Assessment Method for Process Improvement (SCAMPI)** — provides a five step process assessment model that incorporates five phases: **initiating, diagnosing, establishing, acting and learning.**
- **CMM-Based Appraisal for Internal Process Improvement (CBA IPI)**
—provides a diagnostic technique for assessing the relative maturity of a software organization; uses the SEI CMM as the basis for the assessment [Dun01]
- **SPICE—The SPICE (ISO/IEC15504)** standard defines a set of requirements for software process assessment. The intent of the standard is to assist organizations in developing an objective evaluation of the efficacy of any defined software process. [ISO08]
- **ISO 9001:2000 for Software**—a generic standard that applies to any organization that wants to improve the overall quality of the products, systems, or services that it provides. Therefore, the standard is directly applicable to software organizations and companies. [Ant06]

PREScriptive MODELS

- Prescriptive process models advocate an orderly approach to software engineering

That leads to a few questions ...

- If prescriptive process models strive for structure and order, **are they inappropriate for a software world that thrives on change?**
- Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, **do we make it impossible to achieve coordination and coherence in software work?**

Waterfall Model

- Winston Royce introduced the Waterfall Model in 1970.
- Also called as Linear Sequential Development Model.
- Waterfall approach was first SDLC Model, used widely in Software Engineering to ensure success of project.
- Each phase must be completed before the next phase can begin. Output of one phase will be input of next phase.
- This model is named "Waterfall Model", because its diagrammatic representation of cascading waterfalls.



When to use Waterfall Model?

1. When the requirements are constant and not changed regularly.
2. Application is not complicated and big.
3. A project is short & simple.
4. Where the tools and technology used is consistent and is not changing.
5. When resources are well prepared and are available to use.



Waterfall Model Phases

1. Requirement Analysis:

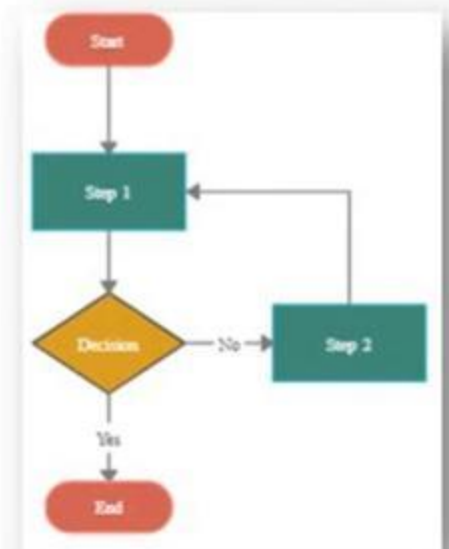
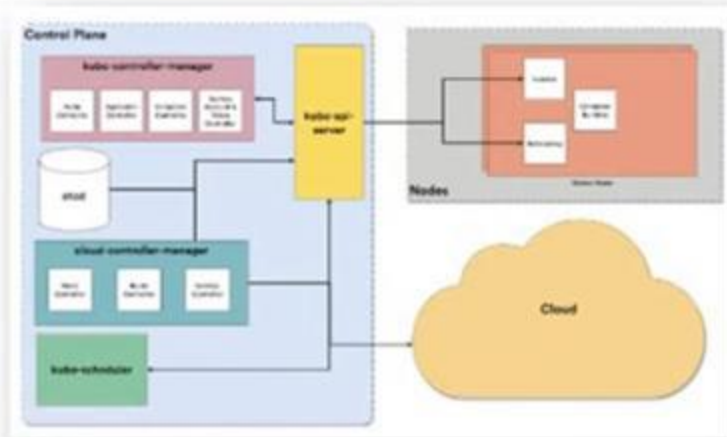
- The aim of this phase is to understand the exact requirements of the customer and to document them properly.
- They discuss functions, performance and interfacing requirement of the software.
- In this phase, Software Requirement Specification (SRS) document is created .
- It contained a detailed description of what the system will do in the common language.
- SRS document serves as a contract between the development team and customers.



Waterfall Model Phases

2. Design Phase:

- All gathered requirements are converted into suitable design.
- It defines the overall software architecture together with high level and detailed design.
- High level design include Algorithm, Flow charts, Decision tree, Database design etc.
- Low level design include Rough paper design, User interface components etc.
- Also finalized Programming languages, Database or other Hardware & Software requirements.
- All this work is documented as a Software Design Document (SDD).



Waterfall Model Phases

3. Development Phase:

- In this phase software design is translated into source code using any suitable programming language.
- The developer first developed in small programs called units after that integrated it.
- Unit testing is done in this phase to check whether each module is working properly or not.



Waterfall Model Phases

4. Testing Phase:

- In this phase, Tester perform all the testing activities to make sure that the system meets the client requirements or not.
- After combining all the unit modules, Integration testing is done for any faults and failures.
- In case of any anomaly, report it.
- It generates Test cases & Test reports.



Waterfall Model Phases

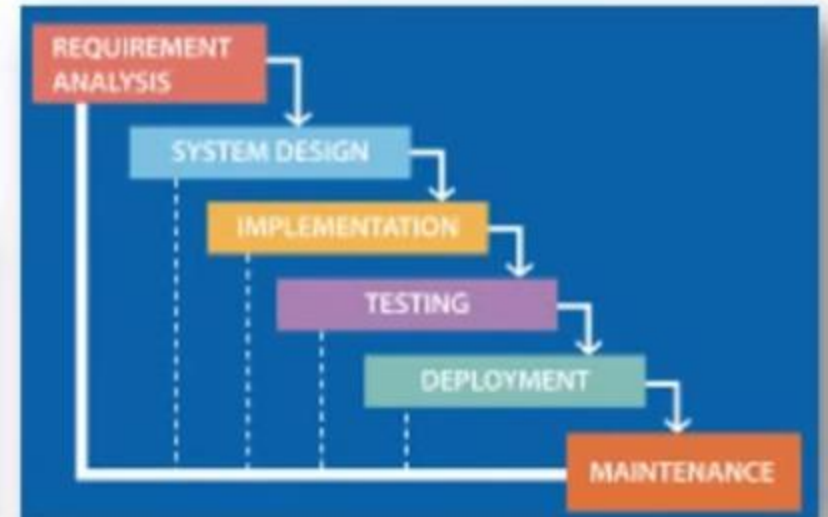
5. Deployment & Maintenance Phase:

- The product is deployed in the customer environment or released into the market.
- In maintenance phase, There are some issues which come up in the client environment.
- To fix those issues, patches are released.
- Also to enhance the product some better versions are released.
- Maintenance is done to deliver these changes in the customer environment.



Advantages of Waterfall Model

1. Simple and easy to understand and use
2. Before the next phase of development, each phase must be completed
3. Works well for smaller projects.
4. Requirements are very well understood & fixed.
5. Any changes in software is made during the process of the development
6. Process and results are well documented.

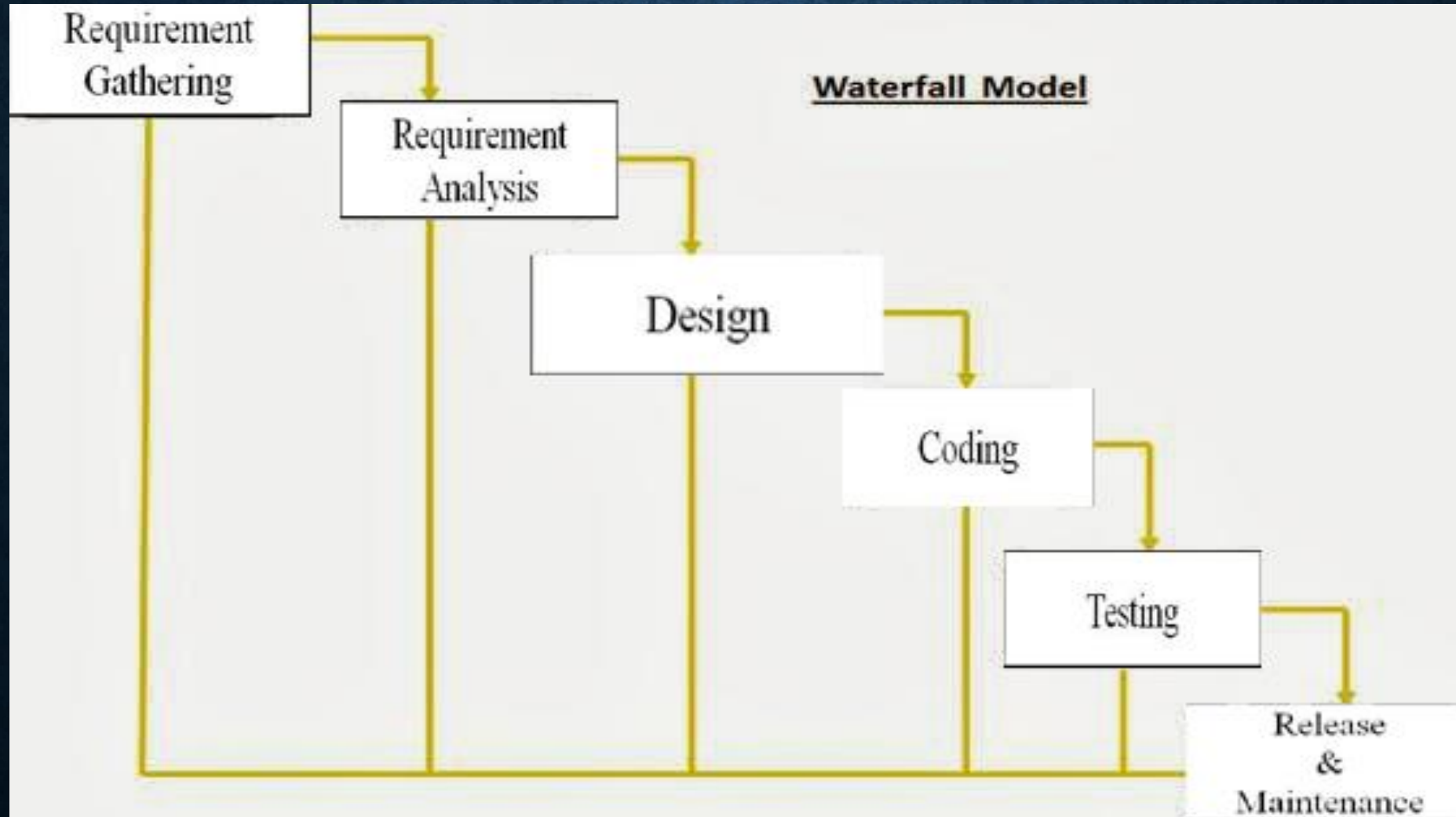


Disadvantages of Waterfall Model

1. High amounts of risk and uncertainty.
2. Error can be fixed only during the phase.
3. Not suitable for the projects where requirements are changing.
4. It becomes very difficult to move back to the phase.
5. Not a good model for complex and object-oriented projects.
6. Clients valuable feedback cannot be included with ongoing development phase



WATERFALL MODEL



WHAT IS V-MODEL

- V- model means Verification and Validation model.
- V-Shaped life cycle is a sequential path of execution of processes.
- Each phase must be completed before the next phase starts.
- Testing of the product is planned in parallel with a corresponding phase of development.

WHEN TO USE THE V-MODEL?

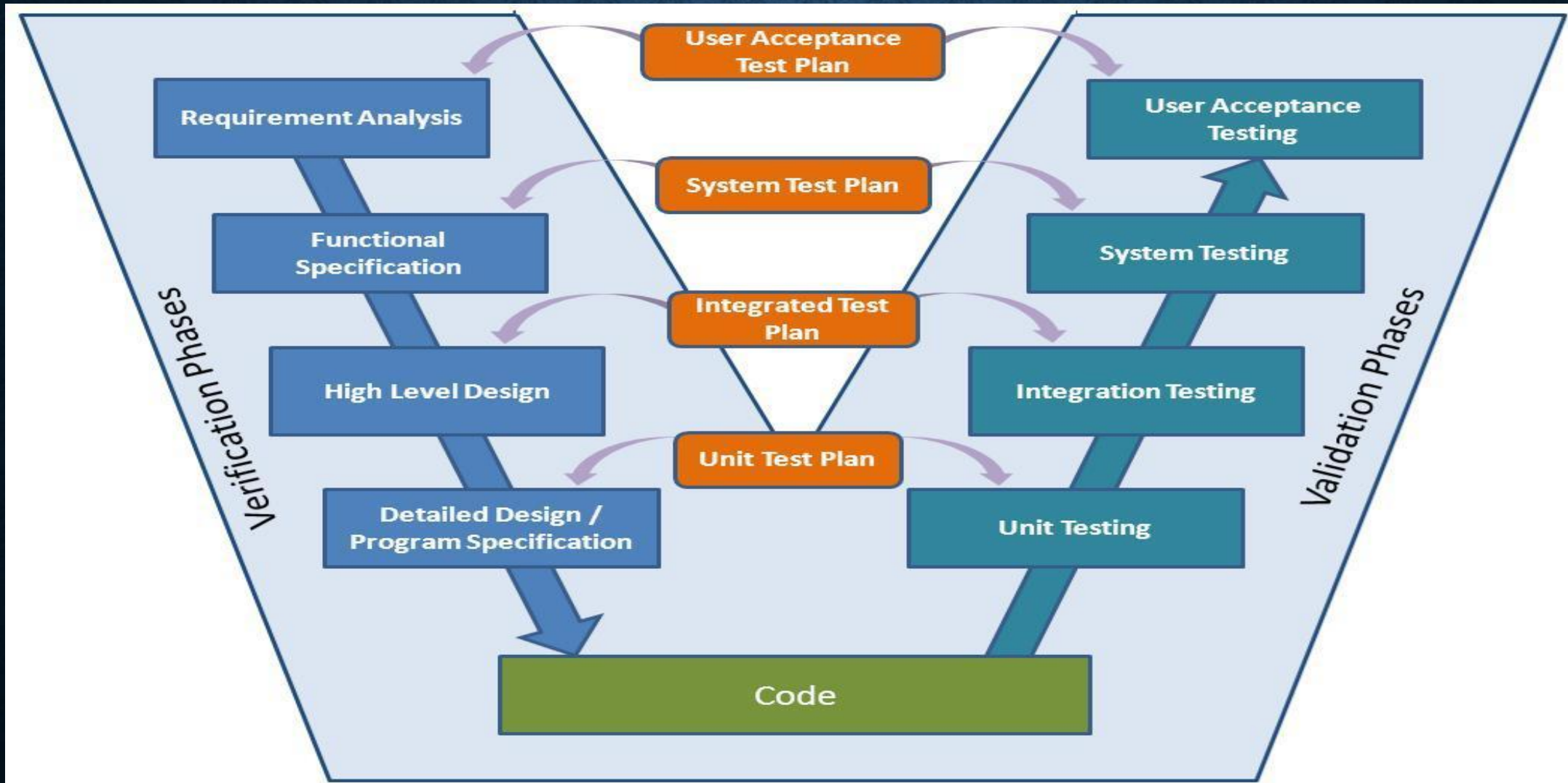
- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when sample technical resources are available with needed technical expertise.

PHASES

There are two phases

- Verification Phase
- Validation Phase

PICTORIAL VIEW OF V-MODEL



VERIFICATION PHASE

- **Requirements Analysis:**

the first step in the verification process, the requirements of the system are collected by analyzing the needs of the user.

- **System design:**

In this phase system engineers analyze and understand the business of the proposed system by studying the user requirements document.

VERIFICATION PHASE

- **Architecture design**

The baseline in selecting the architecture is that it should realize all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology details etc. The integration testing design is carried out in the particular phase.

VERIFICATION PHASE

- **Module design**

The module design phase can also be referred to as **low-level design**. The designed system is broken up into smaller units or modules and each of them is explained so that the programmer can start coding directly. The low level design document or program specifications will contain a detailed functional logic of the module, in pseudo-code:

- ❖ database tables, with all elements, including their type and size
- ❖ all interface details with complete API references
- ❖ all dependency issues
- ❖ error message listings
- ❖ complete input and outputs for a module.

CODING

- This is at the bottom of the V-Shape model. Module design is converted into code by developers.

VALIDATION PHASE

- Unit testing

- Unit Test Plans (UTPs) are developed during **module design phase**. These UTPs are executed to **eliminate bugs at code level or unit level**.
- A unit is the **smallest entity which can independently exist**, e.g. a program module. Unit testing verifies that the **smallest entity can function correctly when isolated from the rest of the codes/units**.

VALIDATION PHASE

- **Integration testing**

Integration Test Plans are developed during the **Architectural Design Phase**. These tests **verify that units created and tested independently can coexist and communicate among themselves.**

- **System testing**

System Tests Plans are developed during **System Design Phase**. Unlike Unit and Integration Test Plans, System Test Plans are composed by client's business team. System Test ensures that **expectations from application developed are met.**

VALIDATION PHASE

- User acceptance testing

User Acceptance Test (UAT) Plans are developed during the Requirements Analysis phase. Test Plans are composed by business users. UAT is performed in a user environment that resembles the production environment, using realistic data.

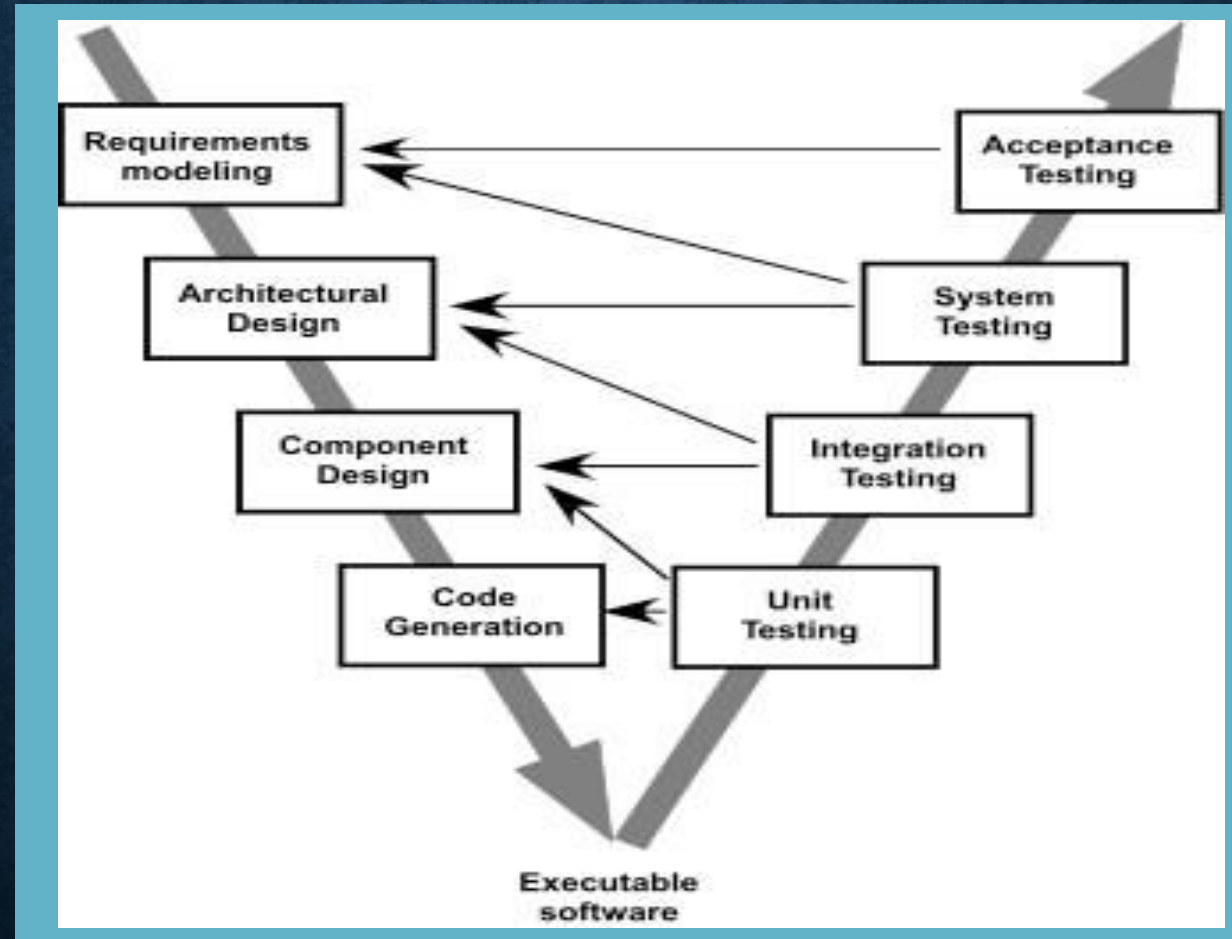
MERITS

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

DEMERITS

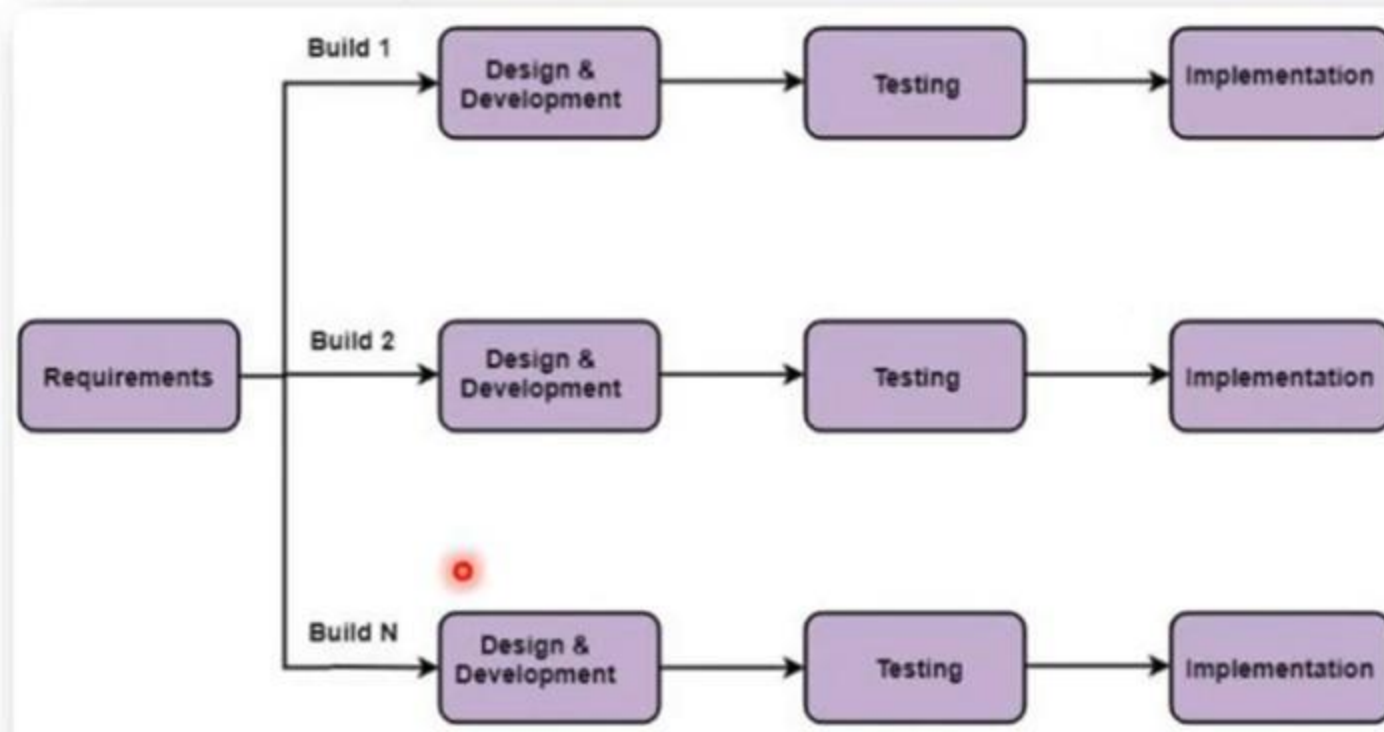
- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

THE V-MODEL



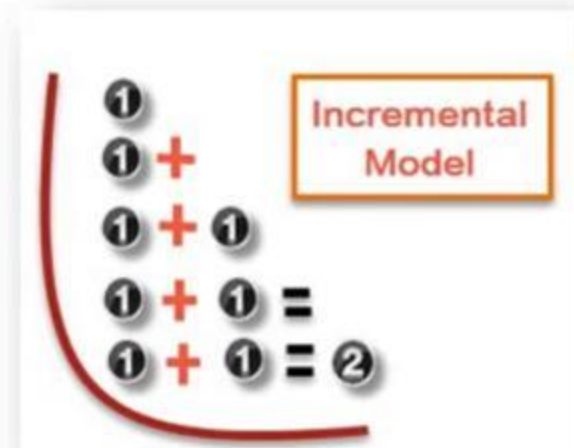
Incremental Process / Development Model

- In Incremental Model, requirements divided into multiple standalone modules of the software development cycle.
- Each module goes through the requirements, design, implementation and testing phases.
- Every subsequent release of the module adds function to the previous release.
- The process continues until the complete system achieved.



When to use Incremental Process Model?

1. Requirements are clearly specified, understood and are known up-front.
2. When the customer demands a quick release of the product.
3. You can develop prioritized requirements first.
4. A project has a long development schedule.
5. When Software team are not very well skilled or trained.
6. Lowers initial delivery cost.
7. Such methodology is more in use for web application and product based companies



Incremental Process Model Phases

1. Requirement Gathering & Analysis:

- Business Analyst & Product Manager collect all the Functional & Non Functional requirements of customers.
- This phase performs a crucial role in Incremental Model.

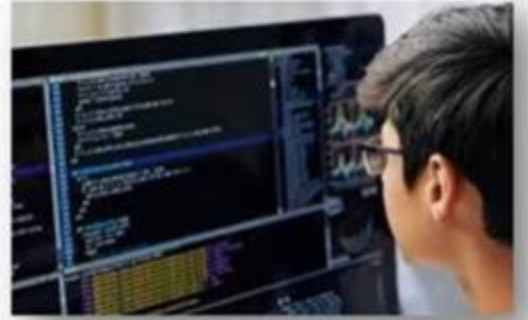


2. Design Phase:

- It design the overall software architecture together with high level and detailed design.
- Designer design Algorithm, Data Flow diagram, Activity diagram, Class diagram, State Transition diagram, Database design etc.



Incremental Process Model Phases



3. Implementaion Phase:

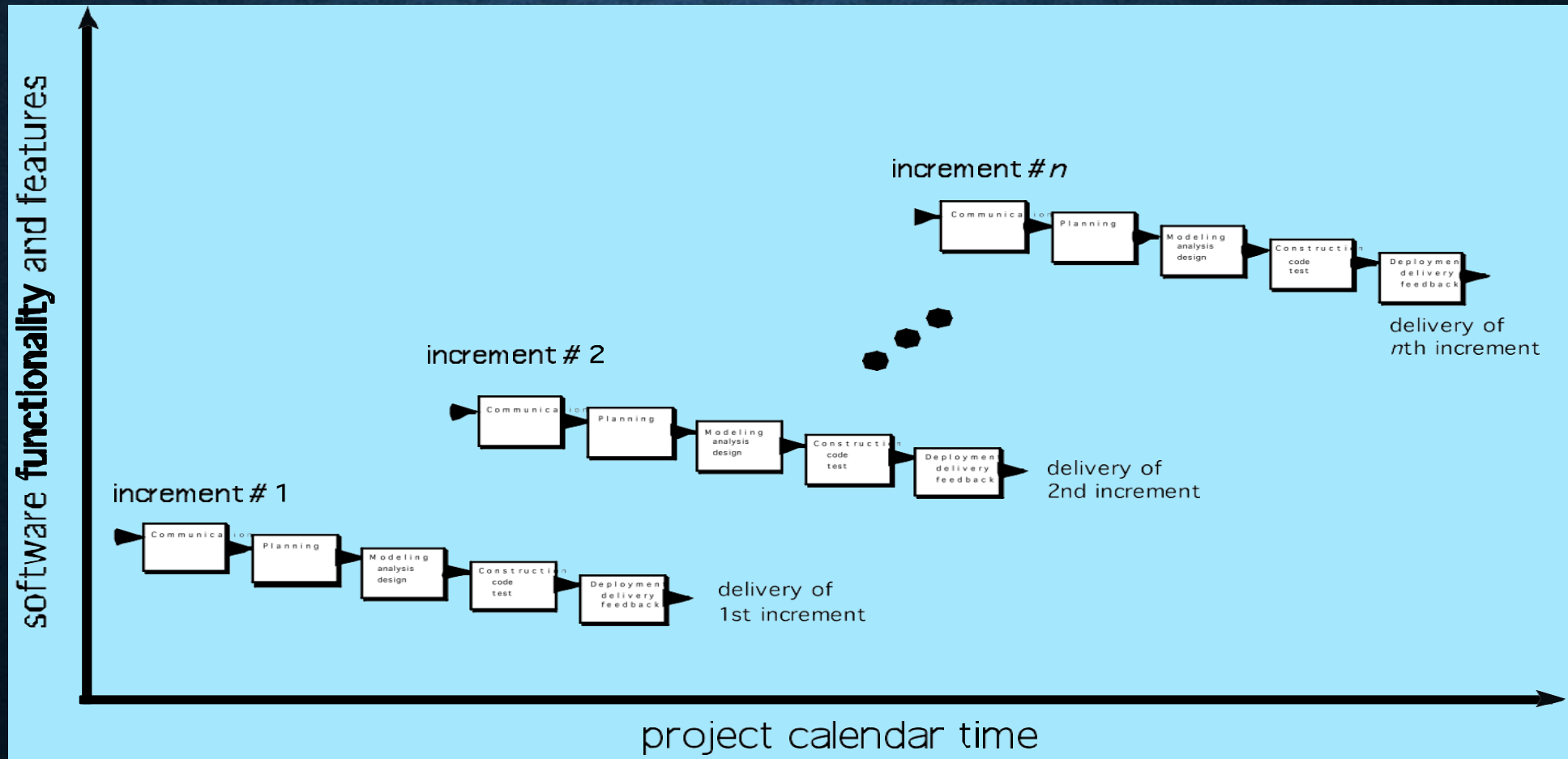
- In this phase software design is translated into specific source code.
- Programmers use suitable programming languages, frameworks & databases.

4. Testing Phase:

- In the testing phase, Tester checks the performance of each existing function as well as additional functionality as per customer requirements or not.
- There are many test methods, but the most common are white box, black box and grey box test methods.

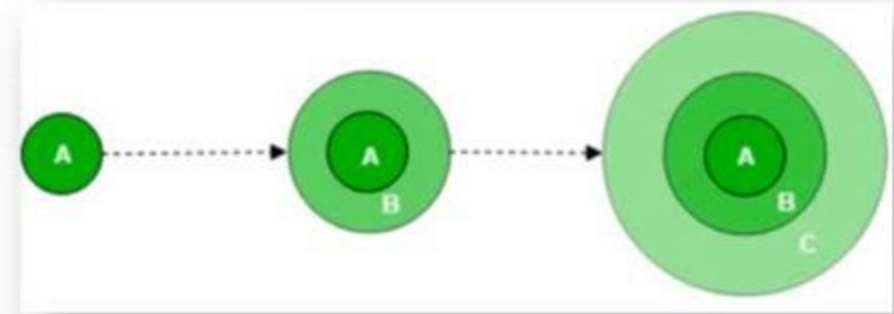


THE INCREMENTAL MODEL



Advantages of Incremental Process Model

1. The Client gets important functionality early.
2. Clients have a clear idea of the project.
3. Errors are easy to be recognized.
4. Easier to test and debug
5. Simple to manage risk because it handled during its iteration.
6. Changes are easy to implement.
7. It is flexible and less expensive to change requirements and scope.
8. Partial systems are successively built to produce a final total system.



Disadvantages of Incremental Process Model

1. Well defined module interfaces are needed.
2. A good team and proper planned execution are required because of its continuous iterations.
3. Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle.
4. Each iteration phase is rigid and does not overlap each other.

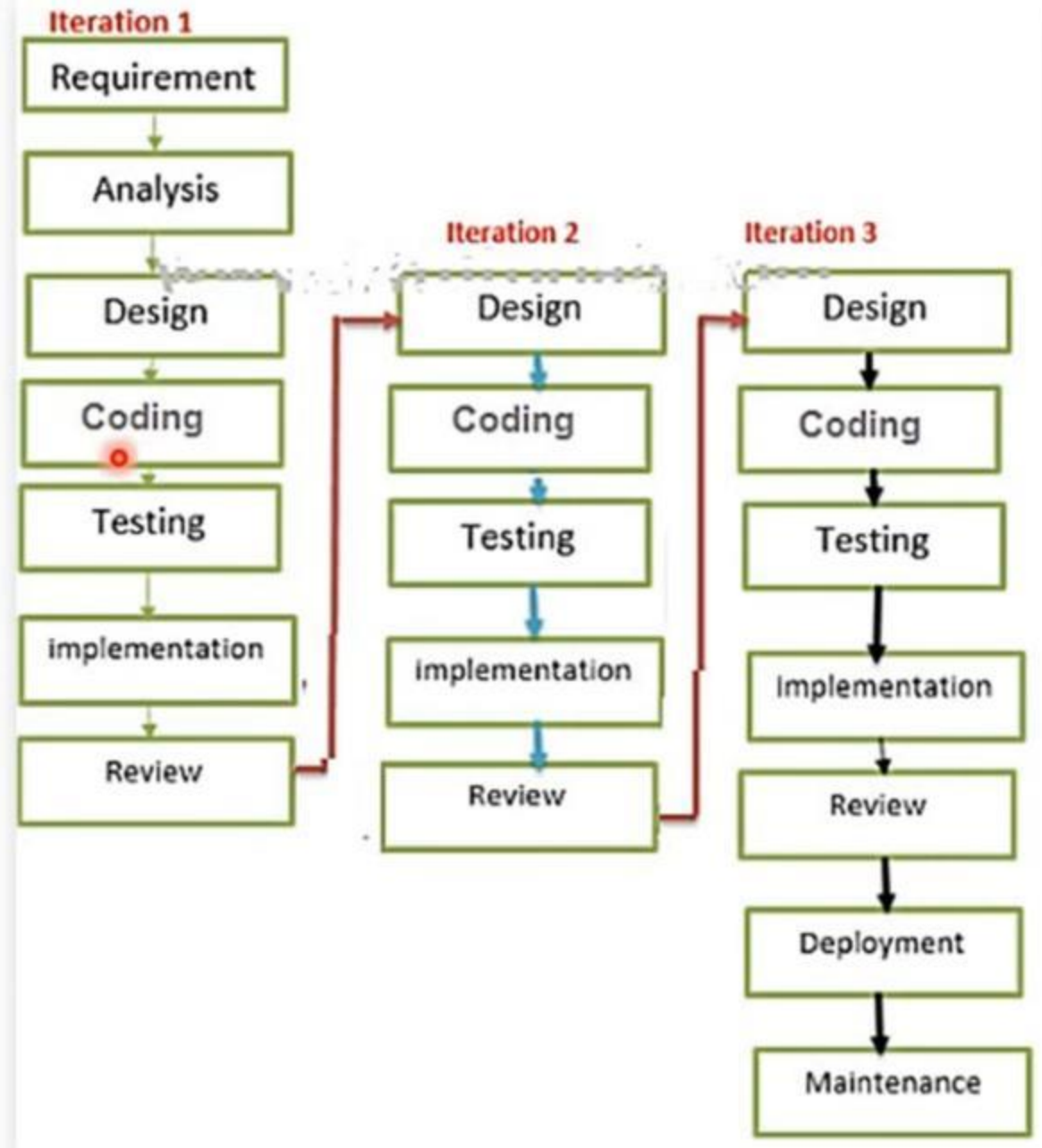


Iterative Model

➤ SDLC Cycle start.

- Start with some Requirements & Analysis.
- Develop first version of S/W.
- If any changes, new version created.
- Again If any changes, new version created.
- Called as Iterations.
- After finalized, Deploy the product.

➤ SDLC Cycle completed.



When to use Iterative Model?

1. When requirements are defined clearly and easy to understand.
2. When there is a requirement of changes in future.
3. When the software application is large.
4. A new technology is being used and is being learnt by the development team while working on the project.
5. There are some high-risk features and goals which may change in the future.

Example for just understanding:



Iterative Model Phases



1. Requirement Gathering & Analysis:

- The aim of this phase is to understand the exact requirements of the customer.
- The system analyst analyses the requirements that whether the requirement can be fulfilled or not.
- If customer requirements can be fulfilled than whether these requirements are fulfilled within the budget or out of the budget
- If all okay, Then collected requirements are then planned accordingly for developing the system.

2. Design Phase:

- It design the overall software architecture together with high level and detailed design.
- Designer design Algorithm, Data Flow diagram, Activity diagram, Class diagram, State Transition diagram, Database design etc.



Iterative Model Phases



3. Coding Phase:

- In this phase software design is translated into specific source code.
- Programmers use suitable programming languages, frameworks & databases.

4. Testing Phase:

- In this phase, Tester perform all the testing activities to make sure that the system meets the client requirements or not.
- There are many test methods, but the most common are white box, black box and grey box test methods.



Iterative Model Phases

5. Review Phase:

- The review phase is where the software is estimated and checked as per the current requirement.
- Then, further requirements are reviewed discussed and reviewed to propose for an update in the next iteration.



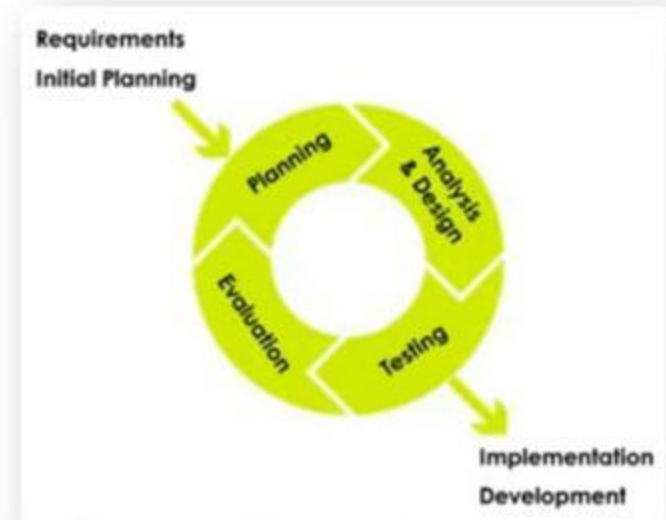
6. Deployment & Maintenance Phase:

- After completing, The product is deployed in the customer environment.
- After deployment of the software in the working environment there may be some bugs, some errors or new updates are required that solved in Maintenance Phase.



Advantages of Iterative Model

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



Disadvantages of Iterative Model

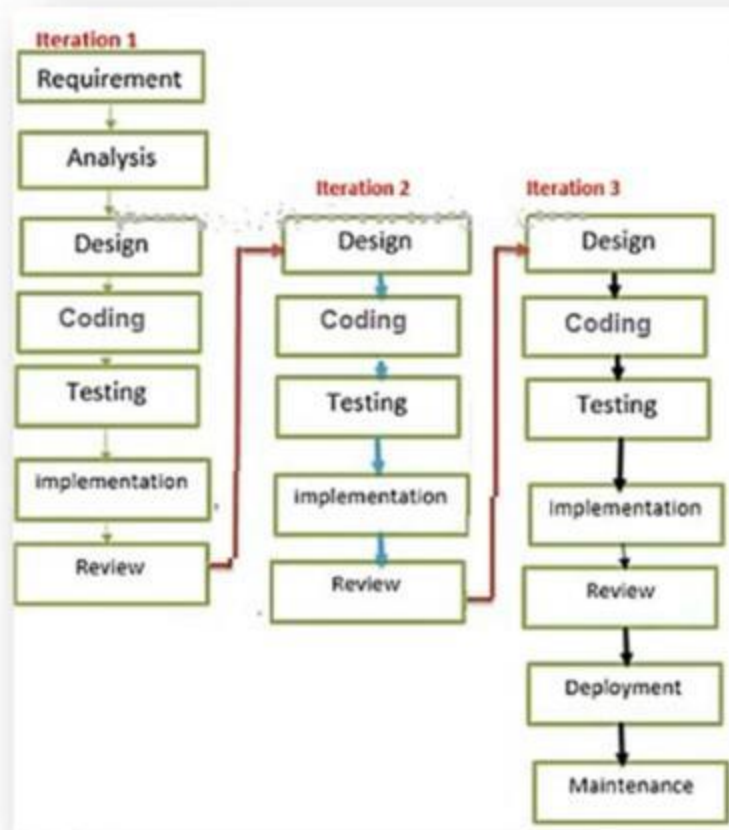
1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.



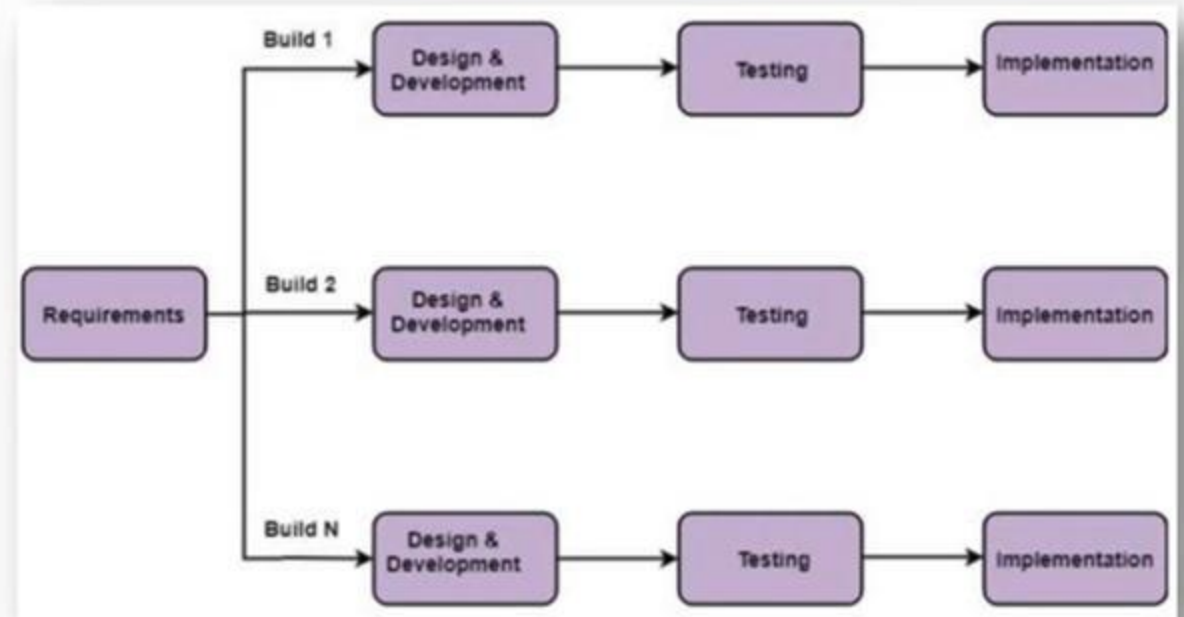
Evolutionary Process Model

- Evolutionary model is also referred to as the successive versions model.
- It is a combination of Iterative & Incremental model.

Revision:



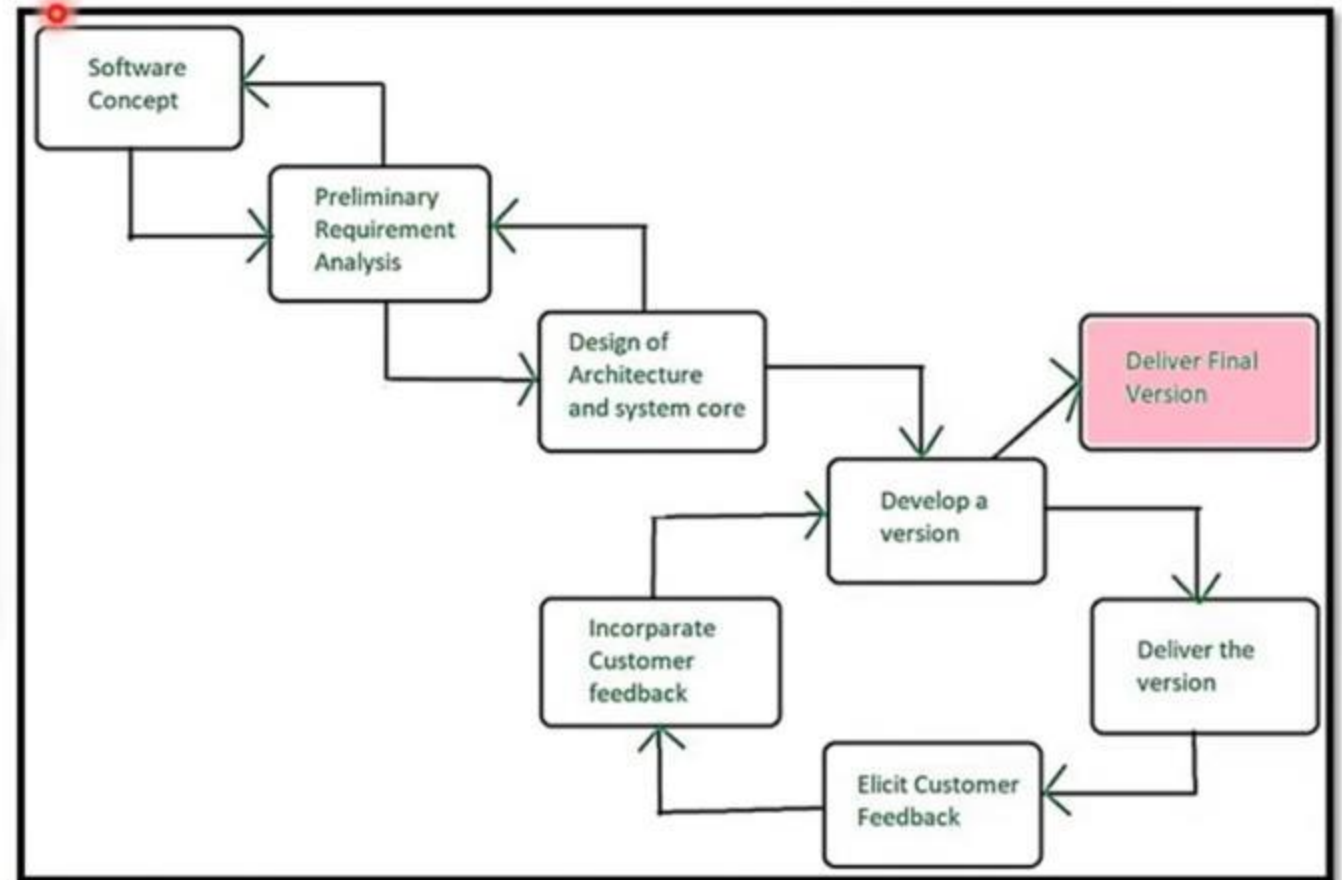
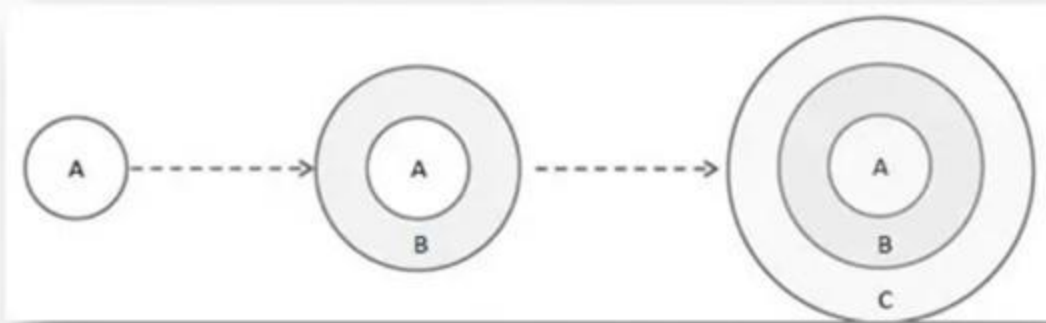
Iterative Model



Incremental Model

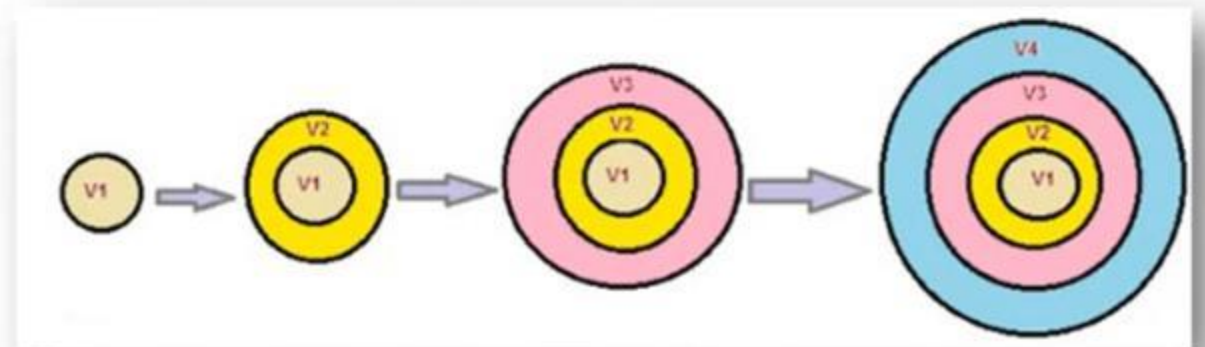
Evolutionary Process Model

- In Evolutionary model, the software requirement is first broken down into several modules
- That can be incrementally constructed means it take customer feedback of each module.
- Delivered product module by module to the customer.



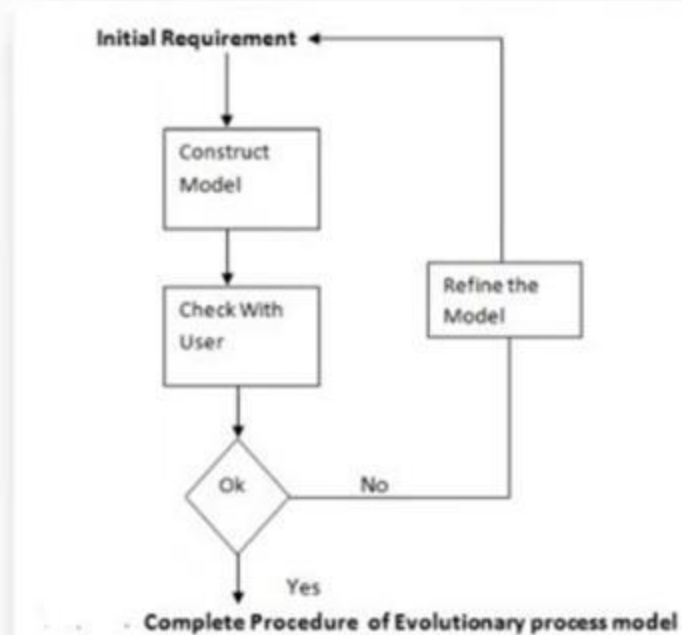
When to use Evolutionary Process Model?

1. Evolutionary model is commonly used when the customer wants to start using the core features instead of waiting for the full software.
2. It is used in large projects because of step by step development.
3. Customer requirements are not fixed. But clear concept.
4. Small changes required in separate modules.
5. Useful in object-oriented software development because all the development is divided into different units.



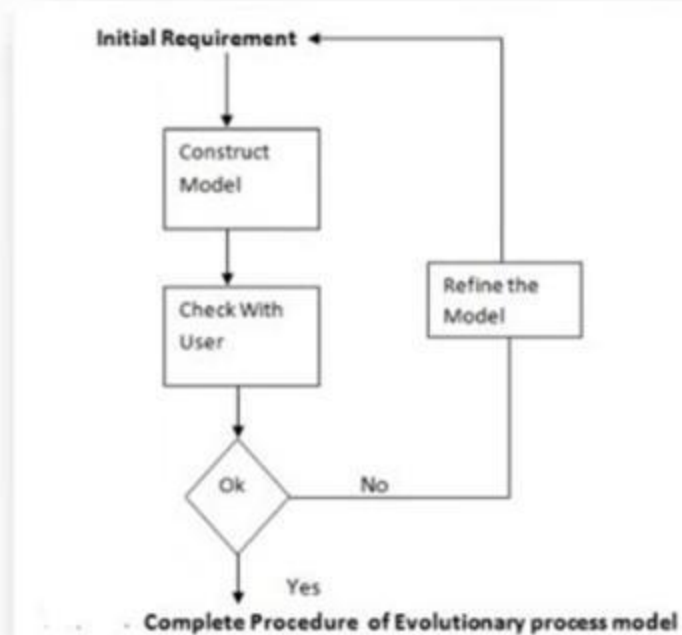
Advantages of Evolutionary Model

1. It is useful for very large products.
2. User has checked every stage during the development and it is helpful in achieving customer confidence.
3. User gets a chance to experiment with a partially developed software much before the complete version of the system is released.
4. It reduces the error because the core modules get tested thoroughly.
5. It also reduces the cost of development.



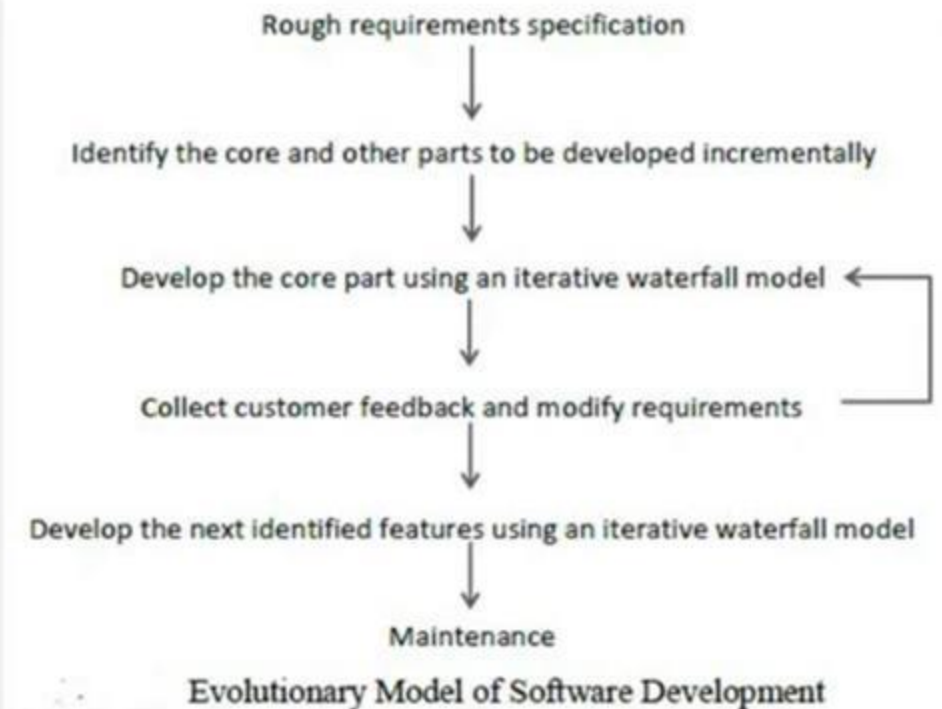
Advantages of Evolutionary Model

1. It is useful for very large products.
2. User has checked every stage during the development and it is helpful in achieving customer confidence.
3. User gets a chance to experiment with a partially developed software much before the complete version of the system is released.
4. It reduces the error because the core modules get tested thoroughly.
5. It also reduces the cost of development.



Disadvantages of Evolutionary Model

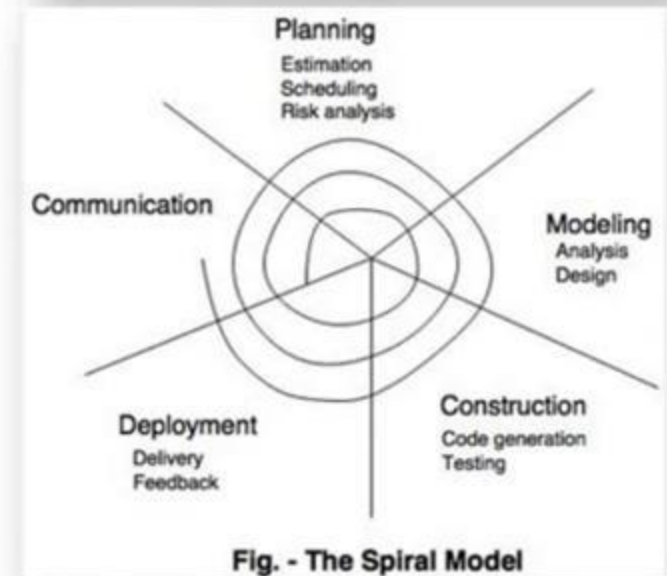
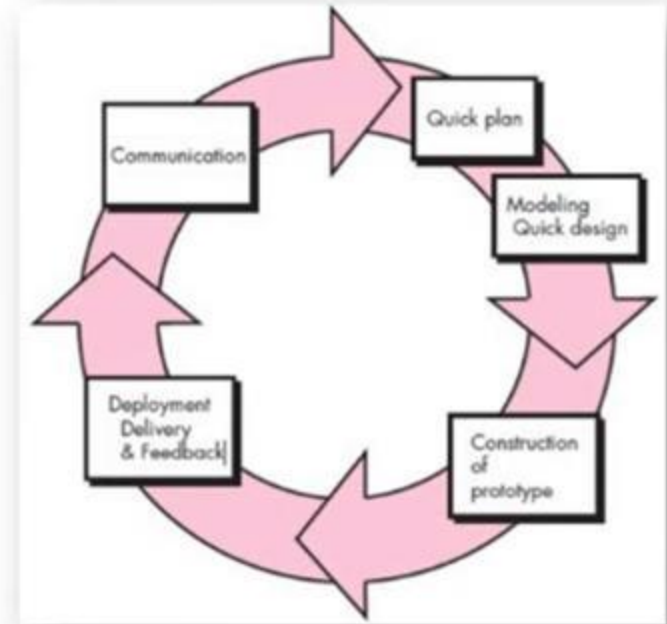
1. Constant and clear interaction between the development team members is required.
2. The delivery of full software can be late due to different changes by customers during development.
3. It is difficult to divide the problem into several parts.



Types Evolutionary Process Model

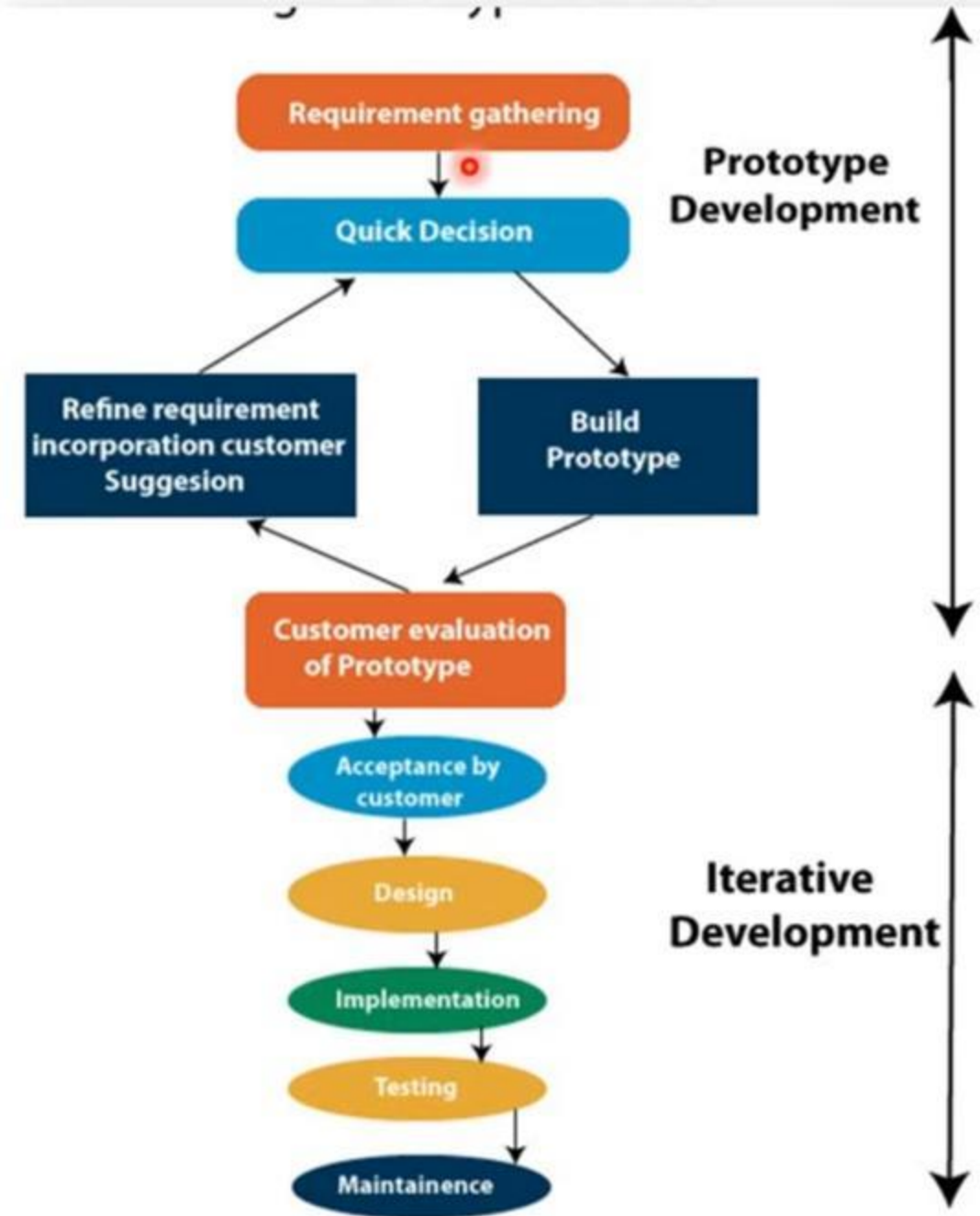
There are total three types of Evolutionary Model:

1. The prototyping model
2. The spiral model
3. Concurrent development model



Prototyping Model

- Prototype is not a complete product it is just functional replica or toy implementation of any idea, software or system.
- It is applied when customers do not know the exact project requirements.
- It generate before actual development of software.
- It is an iterative, trial and error method which takes place between developer and client.

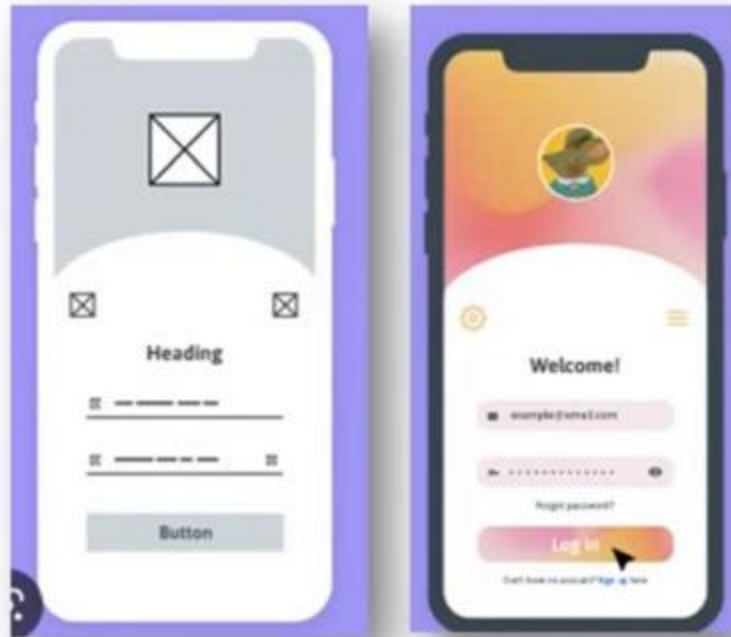


Example Prototype VS Actual



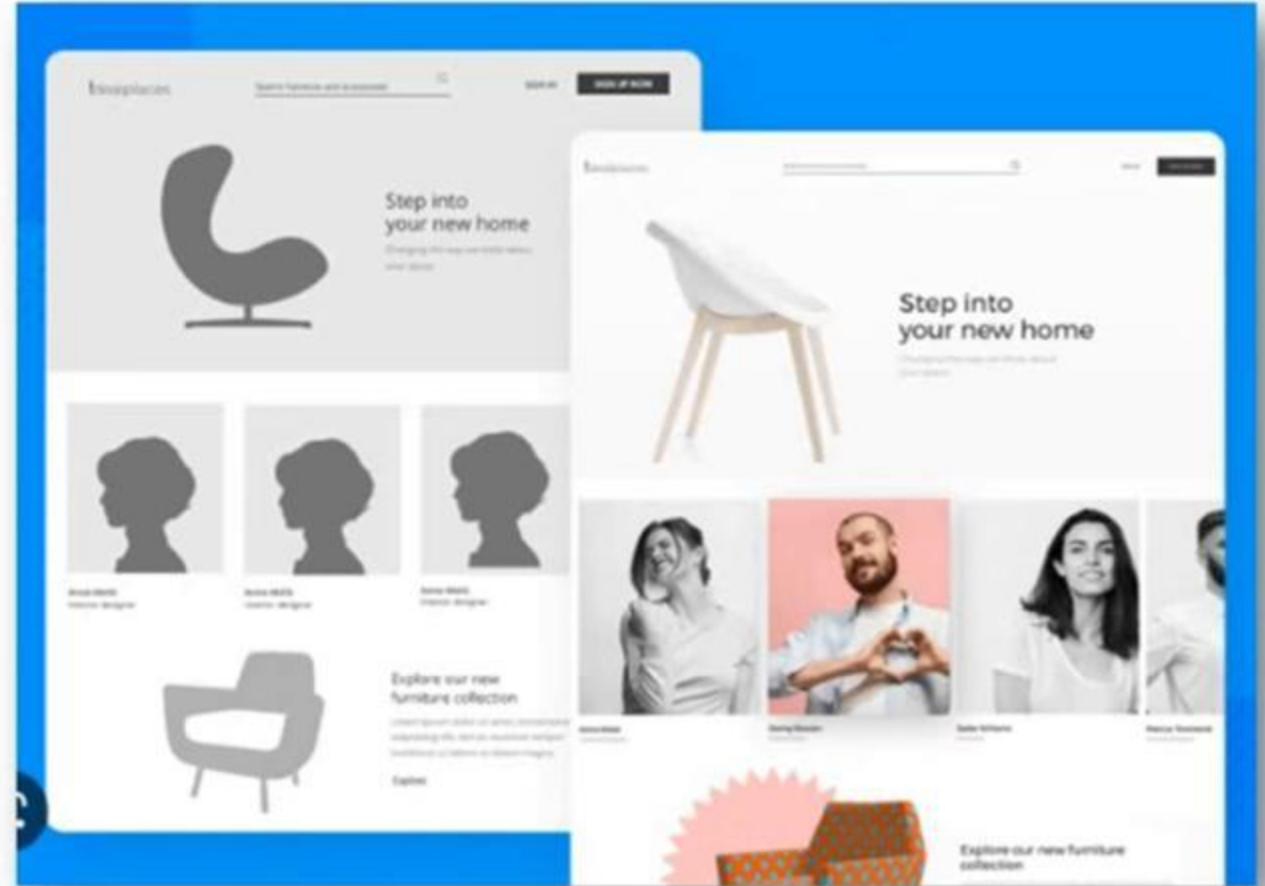
Prototype

Actual



Prototype

Actual

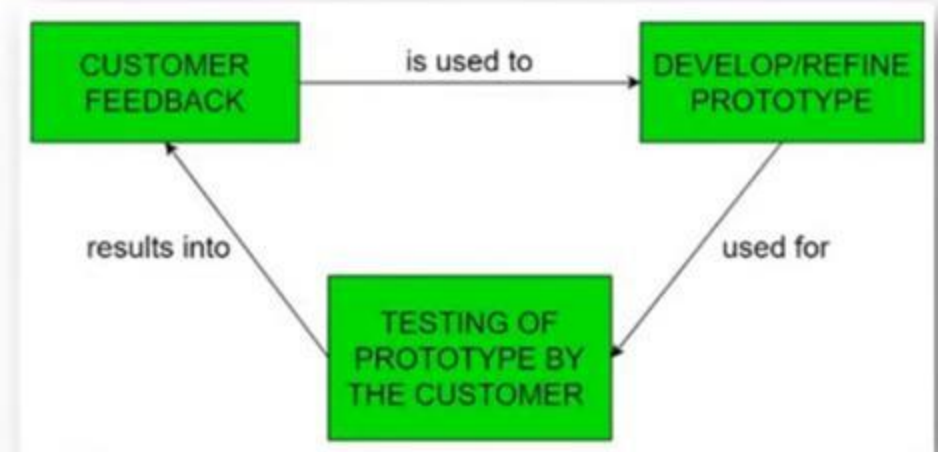


Prototype

Actual

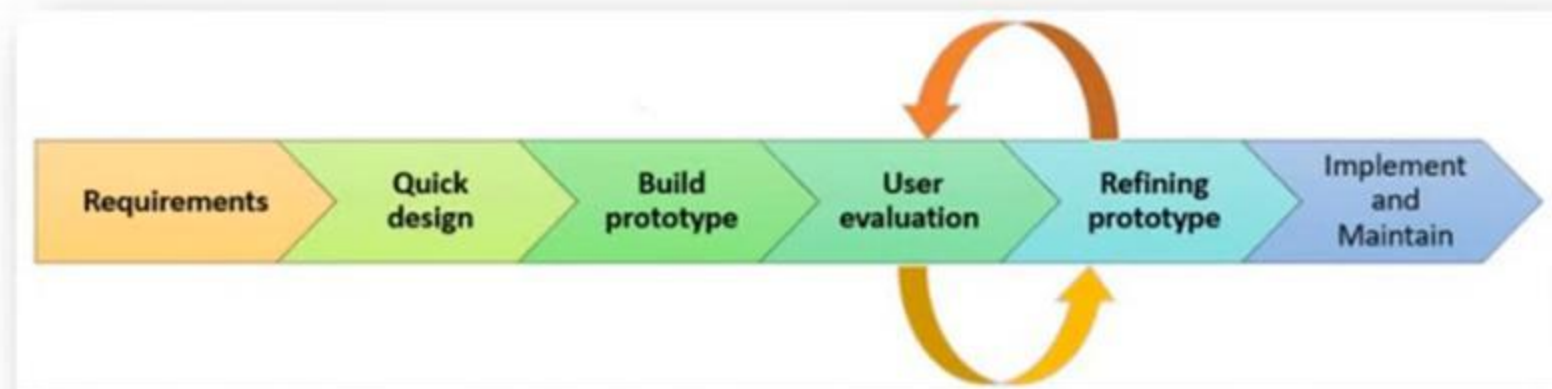
Advantages of Prototyping Model

1. Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
2. Good where requirement are changing.
3. Customers are actively involved in development.
4. Prototypes can be changed and even discarded.
5. Errors can be detected much earlier as the system is made side by side.
6. Flexibility in design.
7. Helps team member to communicate effectively



Disadvantages of Prototyping Model

1. The client involvement is more and it is not always considered by the developer.
2. It is a slow process because it takes more time for development.
3. Many changes can disturb the rhythm of the development team.
4. Documentation is poor as the requirements change frequently.
5. It is a thrown away prototype when the users are confused with it.
6. Costly with respect to time & money.
7. Prototyping tools are expensive.

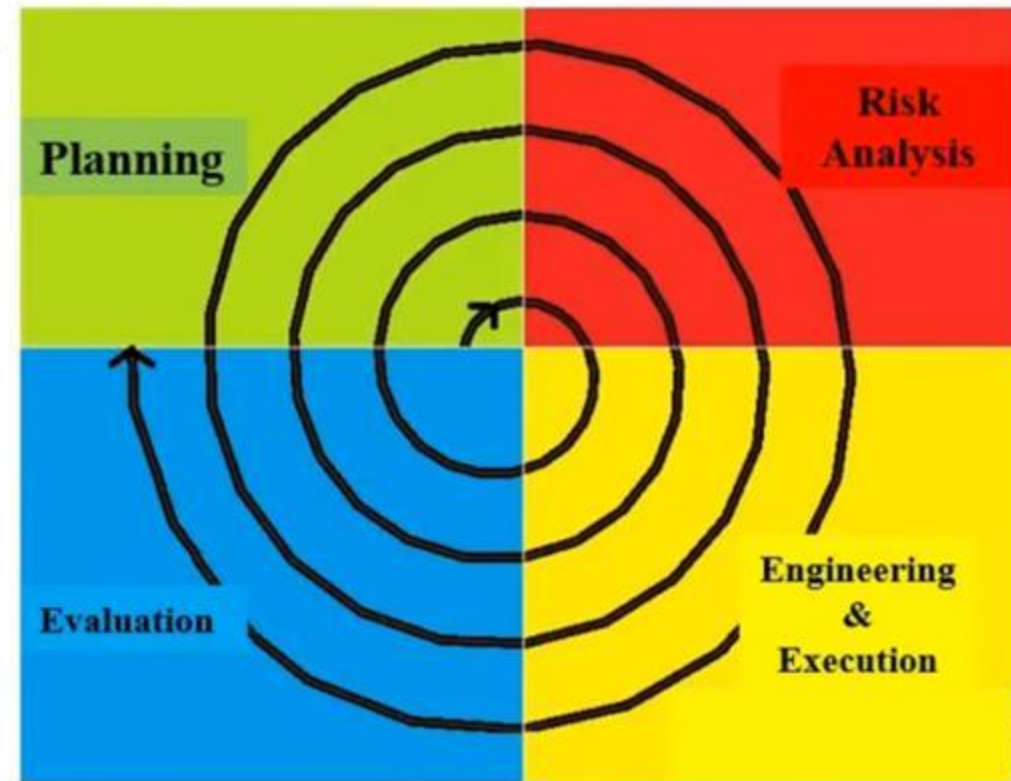


About Spiral Model

- Initially proposed by Boehm in 1986.
- Spiral Model is a **risk-driven software development process model**.
- It is a combination of Waterfall model, Iterative model & Prototyping model.
- Software is developed in a series of incremental releases as per each spiral.
- **Example**: Microsoft, OS Versions, Gaming industry ex.
- Also called as Meta model.

➤ Spiral model divided into the four parts:

1. Planning
2. Risk Analysis
3. Engineering & Execution
4. Customer Evaluation



Spiral Model Phases

1. Planning: (Requirement Gathering & Analysis)

- Communication between customers & project head.
- Collect all the requirements from customers.
- Analyses estimated cost, schedule & required recourses.

2. Risk Analysis:

- Identification of all the potential risks.
- Risk mitigation strategy is planned for solving risks.
- Design a prototype of model.



Spiral Model Phases

3. Engineering & Execution:

- Actual development start.
- Designer design the product as per final prototype.
- Developer perform actual coding or implementation.
- Tester perform all testing methods.
- Deploy or Release product to the customer environment.

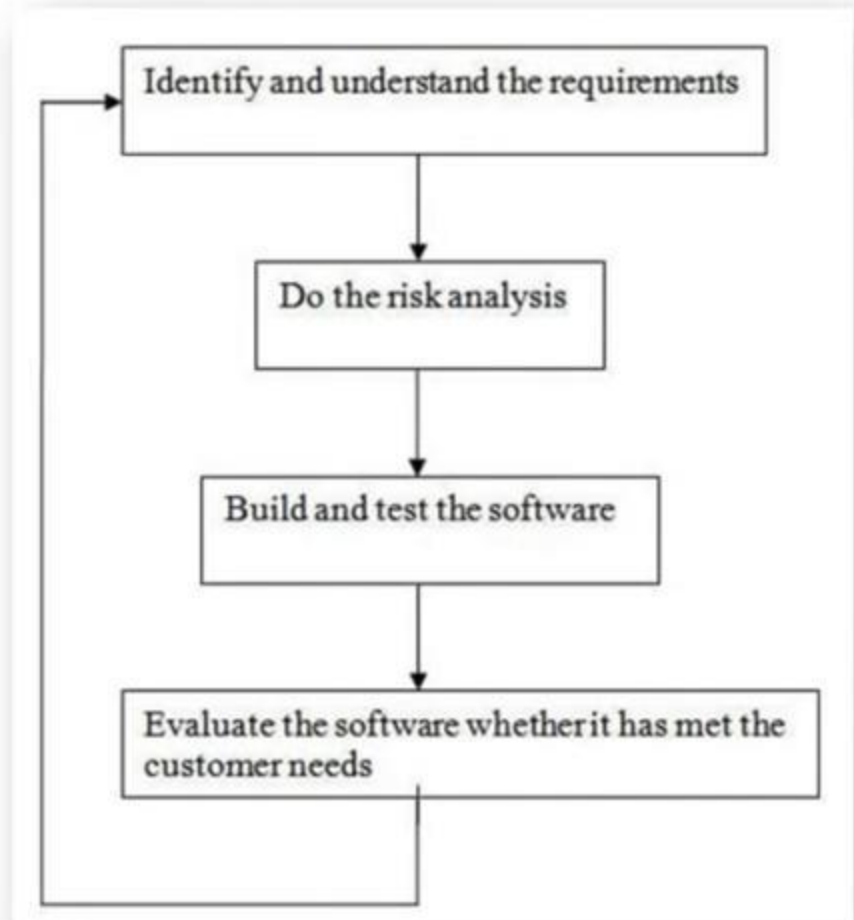
4. Evaluation:

- Take a feedback from customers.
- If customer want any changes, goes to next planning OR **next spiral iteration.**



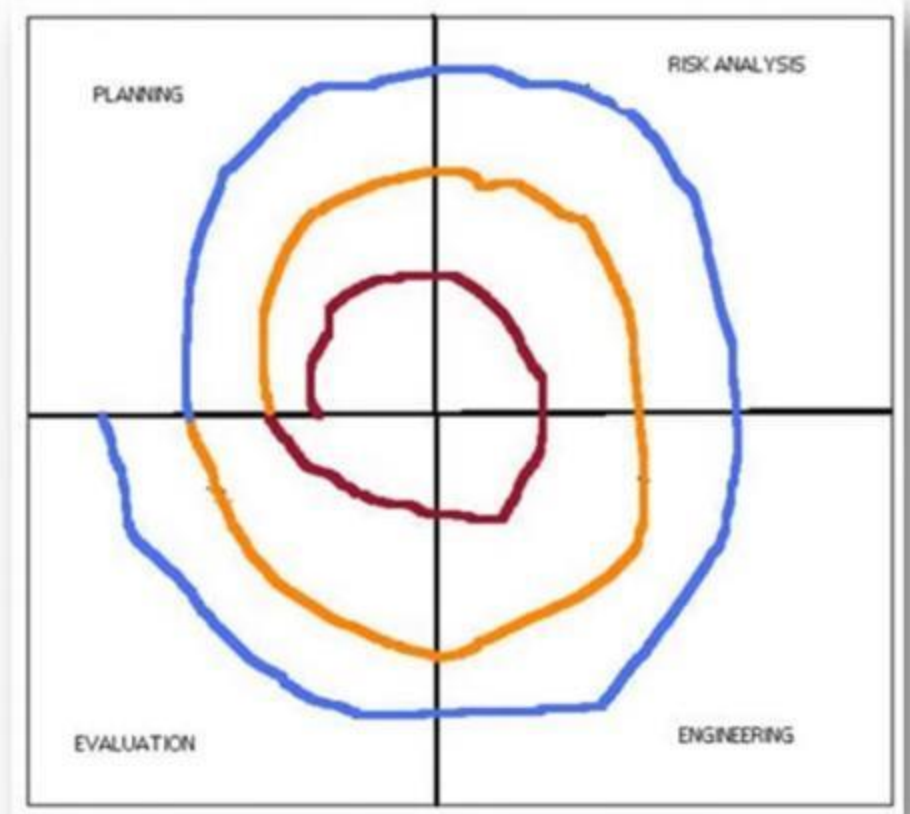
When to use Spiral Model?

1. When the project is large & high budget.
2. When requirements are unclear and complex.
3. Where the software needs continuous risk evaluation.
4. When creation of a prototype is applicable.
5. When changes may require at any time.
6. When long term project commitment is not feasible due to changes in economic priorities



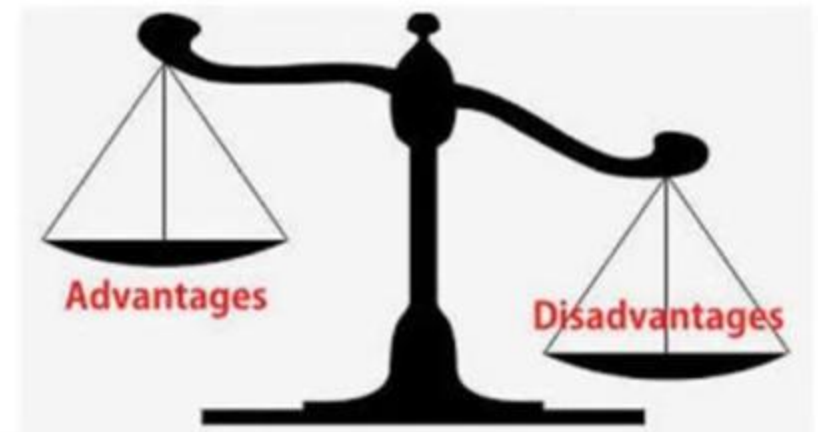
Advantages of Spiral Model

1. High amount of risk analysis.
2. Risky parts can be developed earlier which helps in better risk management.
3. Useful for large and mission-critical projects.
4. Allows extensive use of prototypes.
5. There is always a space for customer feedback
6. Changing requirements can be accommodated.
7. Development is fast.

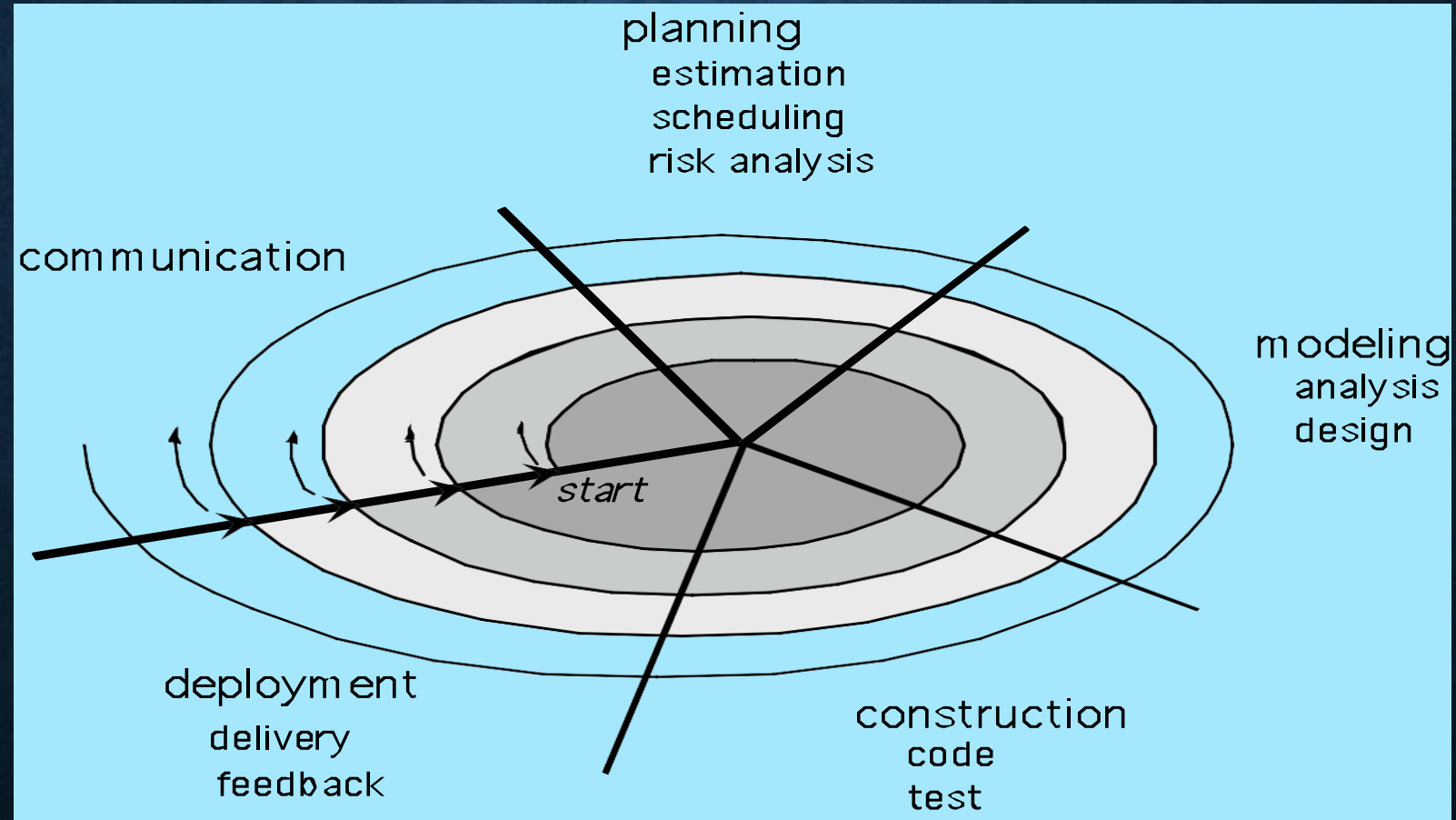


Disadvantages of Spiral Model

1. Risk analysis needed highly particular expertise.
2. Can be a costly model to use.
3. Doesn't work well for smaller projects.
4. Spiral Process is complex sometimes because Spiral may go infinitely.
5. Large number of spiral stages requires excessive documentation.

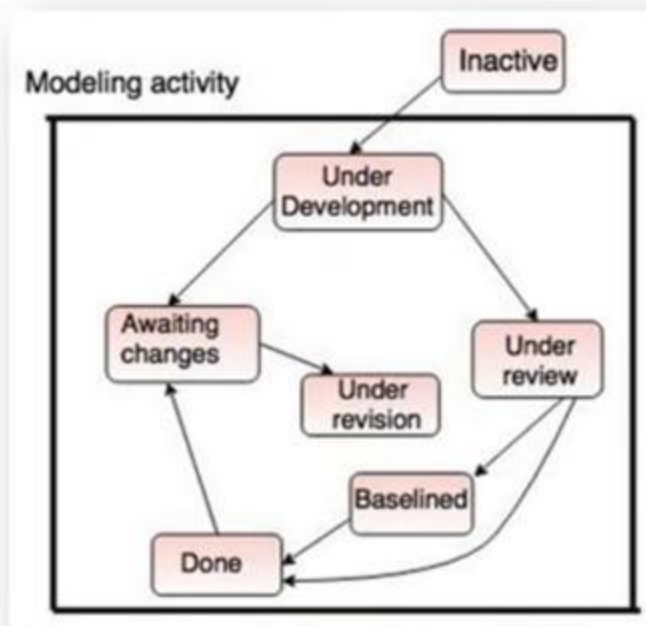


EVOLUTIONARY MODELS: THE SPIRAL



About Concurrent Model

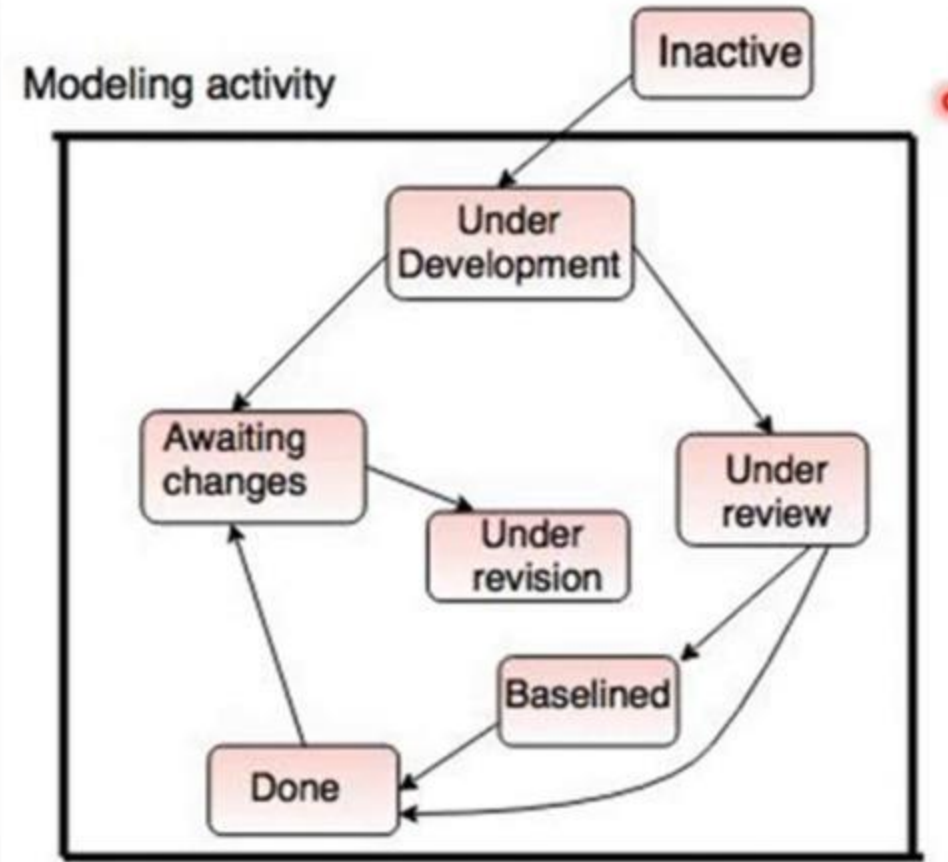
- It is a type of Evolutionary Model.
- Concurrent development model is also known as concurrent engineering.
- The term concurrent mean “done at the same time”.
- It used in all software development process model.
- Transition from state to state for each of the software engineering activities.



Concurrent Model Working

✓ SDLC Activities: Communication, Design, Coding, Testing, Deployment

- **Inactive**: No any activity / state performed.
- **Under Development**: Any activity performed.
- **Awaiting Changes**: If customer want any changes
- **Under review**: Testing activity start.
- **Under revision**: Do all required changes.
- **Baselined**: As per the SRS document
- **Done**: Project completed & Deployed.



Advantages of Concurrent Model

This model is applicable to all types of software development processes.

It is easy for understanding and use.

It gives immediate feedback from testing.

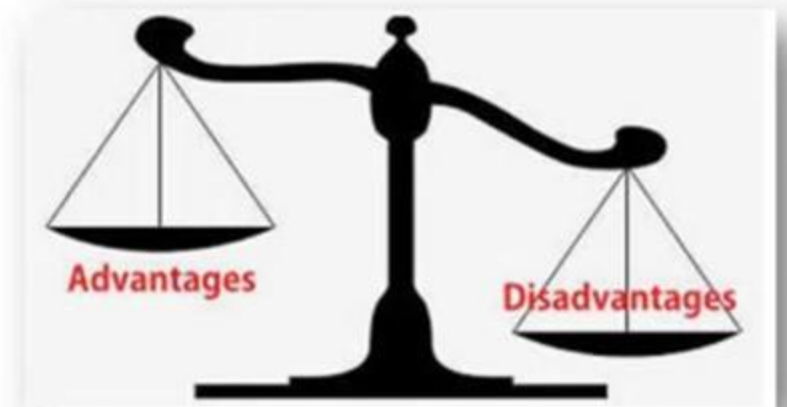
It provides an accurate picture of the current state of a project.



Disadvantages of Concurrent Model

It needs better communication between the team members. This may not be achieved all the time.

It requires to remember the status of the different activities.



Specialized Process Models

- Specialized process models may combine the characteristics of one or more of the traditional models.
- These models are applied when a specialized software engineering approach is chosen.
- Some specialized models are:
 - Component-Based Development
 - Formal Methods Model

Component-Based Development

- The component-based development model incorporates many of the characteristics of the spiral model.
- It is evolutionary in nature which uses an iterative approach to the creation of software.
- However, the component-based development model constructs applications from prepackaged software components.
- Modeling and construction activities begin with the identification of candidate components.
- These components can be designed as either conventional software modules or object-oriented classes or packages of classes.

involves the following steps :

1. Available component-based products are researched and evaluated for the application domain in question.
2. Component integration issues are considered.
3. A software architecture is designed to accommodate the components.
4. Components are integrated into the architecture.
5. Comprehensive testing is conducted to ensure proper functionality.

Advantages

- The component-based development model leads to software reuse.
- The reusability of components allows software engineering team to achieve a reduction in development cycle time as well as a reduction in project cost.

Formal Methods Model

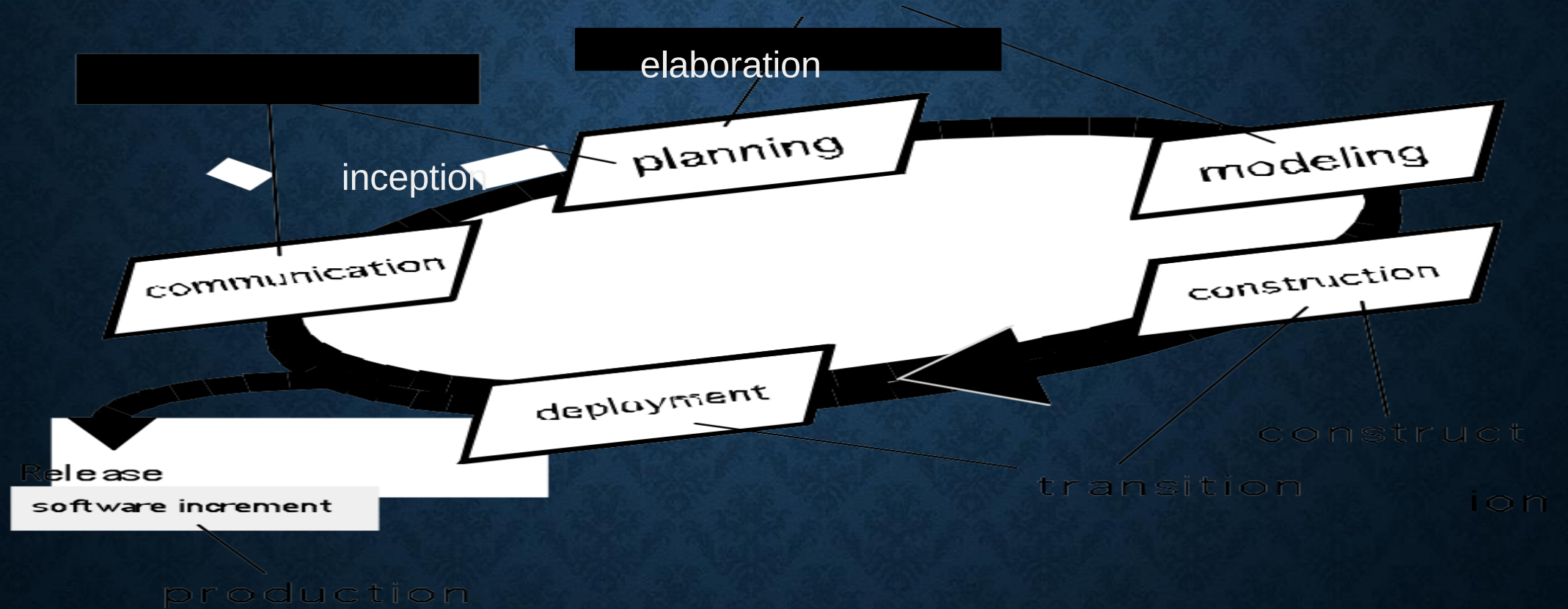
- The formal methods model includes a set of activities that leads to formal mathematical specification of computer software.
- Formal methods enable you to specify, develop, and verify a computer-based system by applying some mathematical notation.
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more efficiently through the application of mathematical analysis.
- When formal methods are used during design, they serve as a basis for program verification and therefore enable you to discover and correct errors that might otherwise go undetected.
- Thus, the formal methods model offers a defect-free software.
- The formal methods approach is used to build safety-critical software (e.g., developers of aircraft avionics and medical devices).

- Some drawbacks are:
 - The development of formal models is time consuming and expensive.
 - Because few software developers have the necessary background to apply formal methods, extensive training is required.
 - It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

STILL OTHER PROCESS MODELS

- **Component based development**—the process to apply when reuse is a development objective
- **Formal methods**—emphasizes the mathematical specification of requirements
- **AOSD**—provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
- **Unified Process**—a “use-case driven, architecture-centric, iterative and incremental” software process closely aligned with the Unified Modeling Language (UML)

THE UNIFIED PROCESS (UP)



UP WORK PRODUCTS

Inception phase

Vision document
Initial use-case model
Initial project glossary
Initial business case
Initial risk assessment.
Project plan,
phases and iterations.
Business model,
if necessary.
One or more prototypes

Elaboration phase

Use-case model
Supplementary requirements
including non-functional
Analysis model
Software architecture
Description.
Executable architectural
prototype.
Preliminary design model
Revised risk list
Project plan including
iteration plan
adapted workflows
milestones
technical work products
Preliminary user manual

Construction phase

Design model
Software components
Integrated software
increment
Test plan and procedure
Test cases
Support documentation
user manuals
installation manuals
description of current
increment

Transition phase

Delivered software increment
Beta test reports
General user feedback

PERSONAL SOFTWARE PROCESS (PSP)

- **Planning.** This activity isolates requirements and develops both size and resource estimates. In addition, a defect estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- **High-level design.** External specifications for each component to be constructed are developed and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.
- **High-level design review.** Formal verification methods (Chapter 21) are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.
- **Development.** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- **Postmortem.** Using the measures and metrics collected (this is a substantial amount of data that should be analyzed statistically), the effectiveness of the process is determined. Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

TEAM SOFTWARE PROCESS (TSP)

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
 - The Capability Maturity Model (CMM), a measure of the effectiveness of a software process, is discussed in Chapter 30.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.