MIT ART DESIGN AND TECHNOLOGY, UNIVERSITY, PUNE

Department of Information Technology

UNIT NO : 1

"INTRODUCTION TO SOFTWARE ENGINEERING"

CONTENTS

CONTENTS

Sr. No.	Name of the Content	Theory / Application
PART 2	SOFTWARE QUALITY ASSURANCE (SQA)	Theory
	8.1 : Verification and Validation	
	8.2 : SQA Plans	
	8.3 : Software Quality Frameworks	
	8.4 : ISO-9000 Models	

PART 1 : INTRODUCTION

1. Nature of Software :

- ✓ Dual : Both a product and a vehicle for delivering a product
  - Product :
    - Software is an Information Transformer.
    - Produces, manages, acquires, modifies, display, or transmits information
  - Vehicle :
    - Supports or directly provides system functionality
    - Controls other programs (e.g., operating systems)
    - Effects communications (e.g., networking software)
    - Helps build other software (e.g., software tools)
- ✓ Ubiquitous : used in variety of applications like engineering ,scientific , business
- ✓ Simple →Complex
- ✓ Internal → Public
- ✓ Single Function →Distributed
- ✓ Informational →Mission-Critical

[Continue..]

Questions (?) About Software Haven't Changed Over the Decades -

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all errors before we give the software to our customers?
- Why do we spend so much time and effort maintaining existing programs?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

Software [Definition] :

-Instructions (computer programs) that when executed provide desired features, function, and performance.

-Data structures that enable the programs to adequately manipulate information.

-Documents that describe the operation and use of the programs.

Characteristics of Software as Compared To Hardware :

✓ It is Developed/Engineered, Not Manufactured In The Classical Sense.

-Impacts the management of software projects

✓ It Does Not Wear Out

- Hardware bathtub curve compared to the software ascending spiked curve

✓ Although the Industry is moving towards component-based construction, most software continues to be custom built.

- Software is still complex to build.

[Continue..]

SOFWARE DOESN’T Wear Out ,But It Does Deteriorate..

FAILURE CURVE [BATHTUBE] : H/W

FAILURE CURVE : S/W

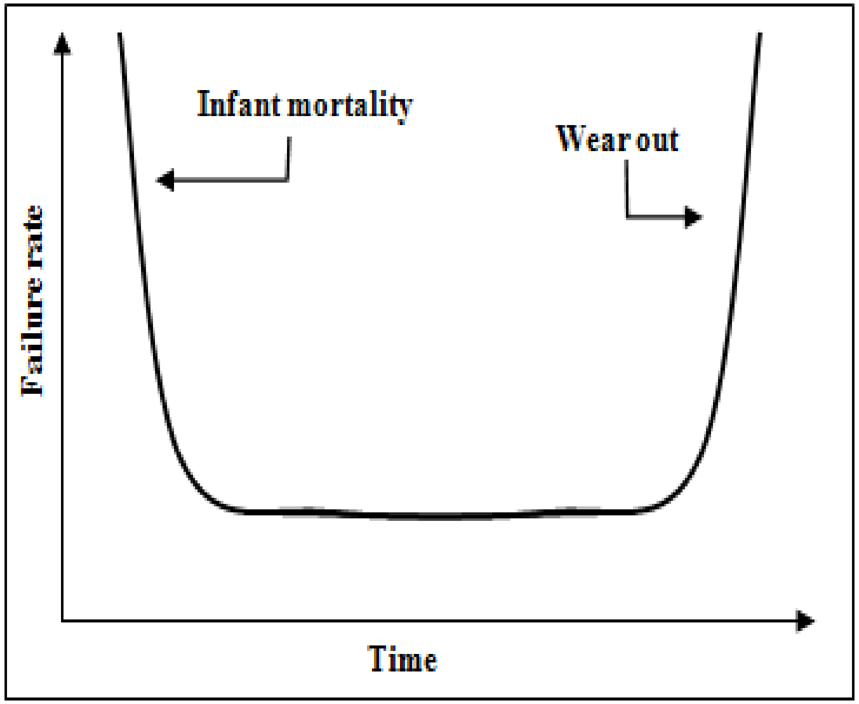


Figure 01: Failure curve for hardware

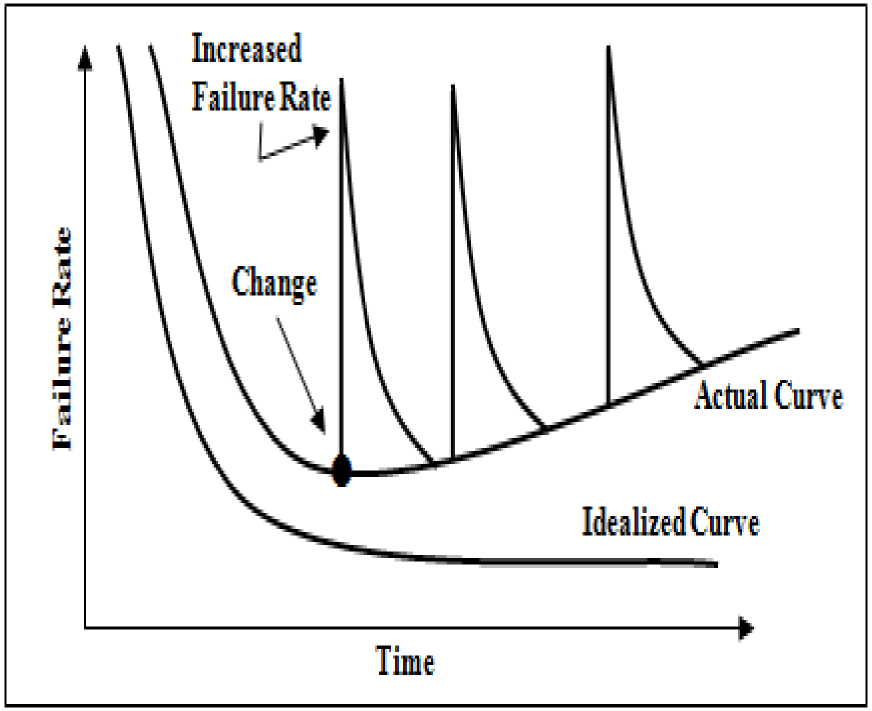


Figure 02: Failure Curves for software

Software Application Domains

S. No	Domain	Description	Examples
1.	System	Collection of Programs written to service other programs	Compilers ,Editors, Drivers etc
2.	Application	Stand-alone programs that solve a specific business need.	VLC, MySQL
3.	Scientific/Engineering	Characterized by "Number Crunching Algorithms "	CAD,CFD
4.	Embedded	Embedded in Hardware to make it SMARTER	Keypad Control [Microwave Oven]
5.	Product-Line	Design to Provide Specific Capability for use by many different customers.	Samsung, Maggie
6.	Webapps	Network Centric S/w spans a wide array of applications.	UDeserve
7.	Artificial Intelligence	Perform tasks requiring Human Intelligence, such as Visual Perception, Speech Recognition, Decision-Making and Translation between languages.	GTA,FIFA , Biometric : Pattern Recognition

Legacy Software

- Legacy software systems were developed decades ago and have been continually modified to meet changes in business requirements and computing platforms.

- The proliferation (rapid growth) of such systems is causing headaches for large organizations who find them costly to maintain and risky to evolve.

- Support core business functions

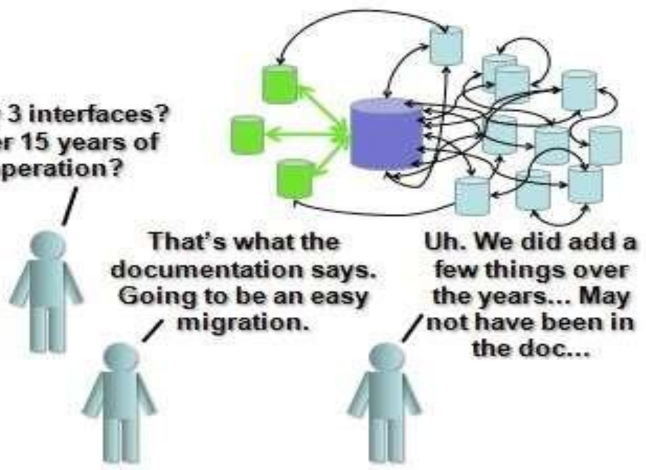
- Exhibit poor quality
  - Convoluted code, poor documentation, poor testing, poor change management

- Example : Microsoft Office -2003 is a Legacy Software

  [Current Version : 2017]

[Continue..]

Legacy Software


Only 3 interfaces? After 15 years of operation?
That's what the documentation says. Going to be an easy migration.
Uh. We did add a few things over the years... May not have been in the doc...

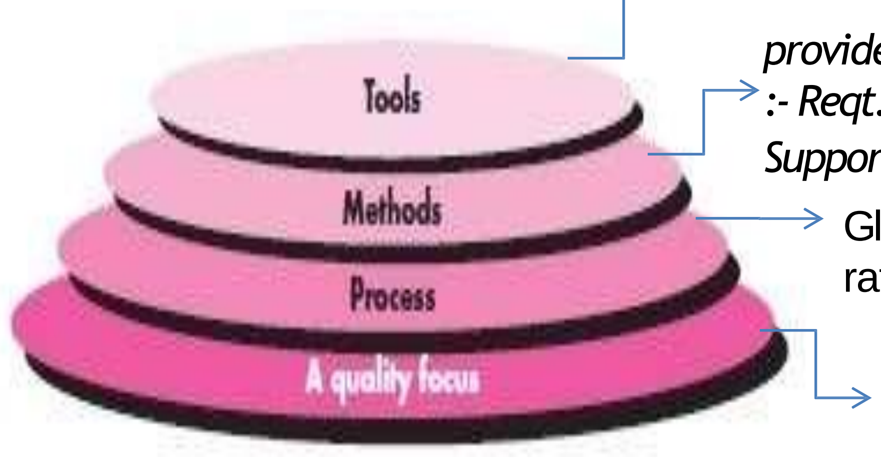
- Reasons For Evolving Legacy Software :

1. (Adaptive) -Must be adapted to meet the needs of new computing environments or more modern systems, databases, or networks
2. (Perfective)- Must be enhanced to implement new business requirements
3. (Corrective) -Must be changed because of errors found in the specification, design, or implementation

Software Engineering [Definition] :

It is defined as systematic, disciplined and quantifiable approach for the development, operation and maintenance of software.

Software Engineering [Layered Technology] :



Provides Automated / Semi-Automated Support for process & Methods

provide the technical how-to's for building software . Tasks :- Reqt. Analysis , Design,Construction,Deployment & Support.

Glue that holds the technology layers together and enables rational and timely development of computer software

The bedrock that supports software engineering

PART 1 : INTRODUCTION

2. Software Engineering Practice :

Software Engineering Practice [Definition] :

-It is a collection of concepts, methods and tools that a software engineer used on a daily basis for the development of software.
-It allows managers to manage software projects and software engineers to build computer program.

The Essence of Software Practice :

a] Understand The Problem :

- Who has a stake in the solution to the problem?
- What are the unknowns?
- Can be problem compartmentalized?
- Can the problem be represented graphically?

[Continue..]

PART 1 : INTRODUCTION

The Essence of Software Practice :

b] Plan a Solution :

- Have you seen similar problems before?
- Has a similar problem been solved?
- Can sub problems be defined?
- Can you represent a solution which leads to effective communication

c] Carry Out The Plan :

- Does the solution conform to the plan?
- Is each component part of solution be correct?

d] Examine the Result :

- Is it possible to test each component part of the solution?
- Does the solution produce result that conforms to the data, function, features and behavior that are required ?

General [Core] Principles of Software Practice
-David Hooker

1) Remember the reason that the software exists

-The software should provide value to its users and satisfy the requirements

2) Keep it simple, stupid ! (KISS)

-All design and implementation should be as simple as possible

-Software developed should be more maintainable n less error-prone.

3) Maintain the vision

-A clear vision is essential to the project's success

4) What You Produce, Others will consume

-Always specify design and implement knowing that someone else will later have to understand and modify what you did

5) Be open to the future

-Never design yourself into a corner; build software that can be easily changed and adapted

6) Plan ahead for software reuse

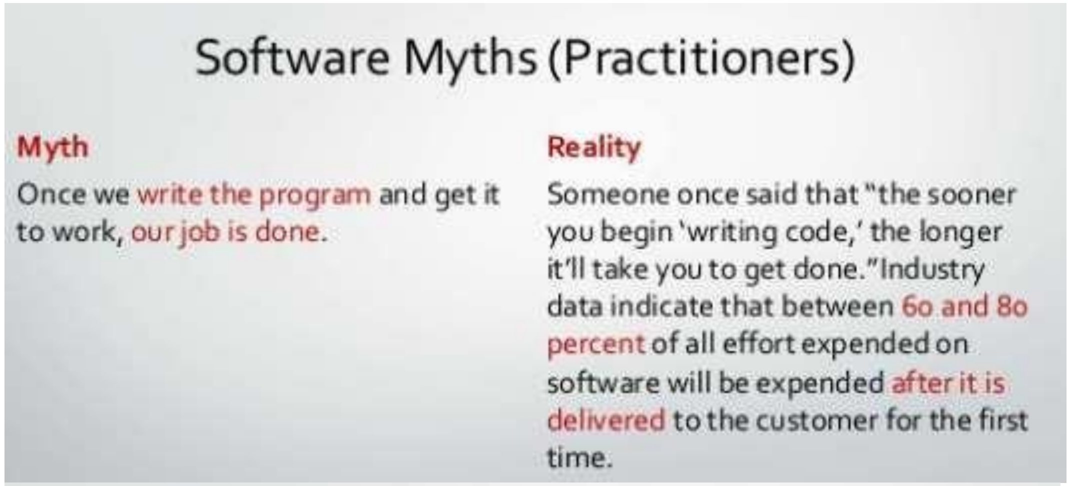
-Reuse of software reduces the long-term cost and increases the value of the program and the reusable components

7) Think, then act

-Placing clear, complete thought before action will almost always produce better results

PART 1 : INTRODUCTION

3. Software Myths: a] What Practitioner Thinks ?



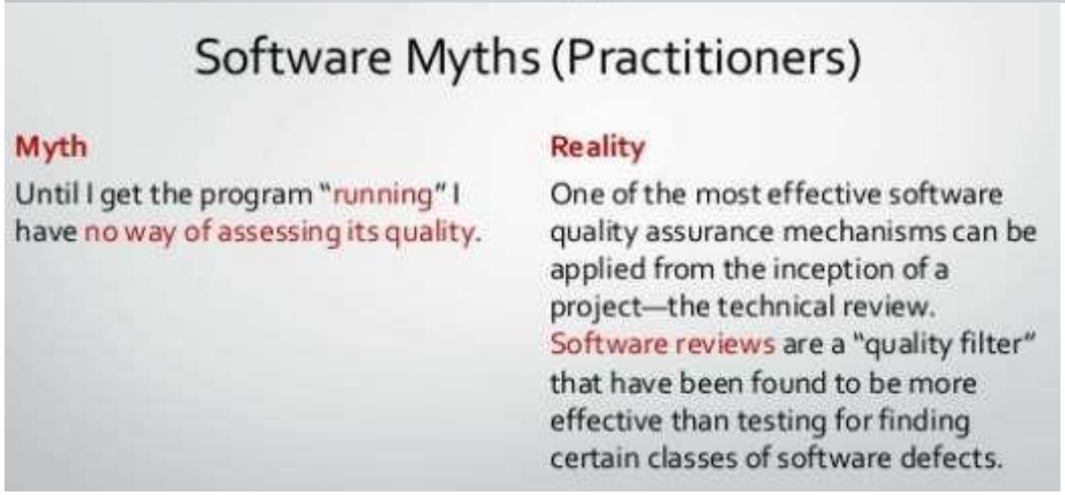
Software Myths (Practitioners)

Myth

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

Reality

Someone once said that "the sooner you begin 'writing code,' the longer it'll take you to get done." Industry data indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.



Software Myths (Practitioners)

Myth

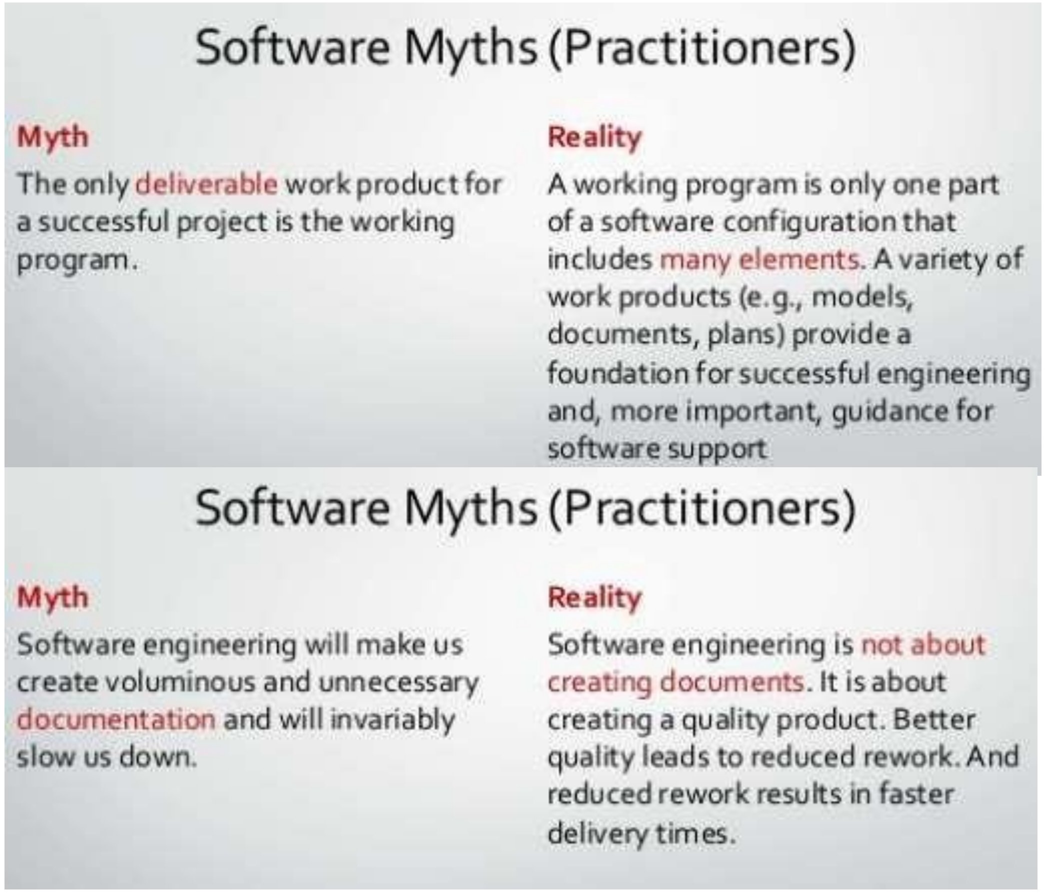
Until I get the program "running" I have no way of assessing its quality.

Reality

One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the technical review. Software reviews are a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

[Continue..]

3. Software Myths: a] What Practitioner Thinks ?



Software Myths (Practitioners)

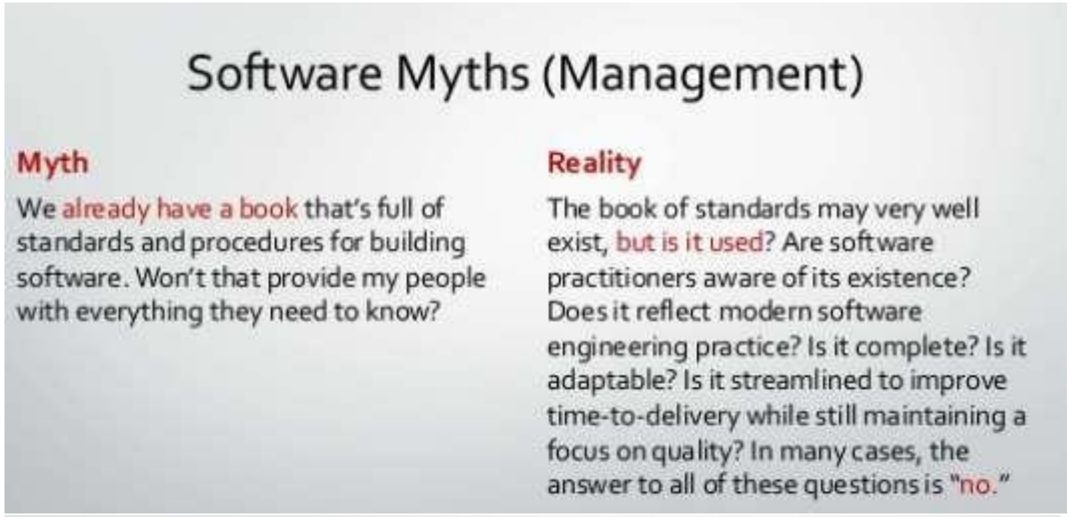
Myth	Reality
The only deliverable work product for a successful project is the working program.	A working program is only one part of a software configuration that includes many elements. A variety of work products (e.g., models, documents, plans) provide a foundation for successful engineering and, more important, guidance for software support

Software Myths (Practitioners)

Myth	Reality
Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.	Software engineering is not about creating documents. It is about creating a quality product. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

[Continue..]

3. Software Myths: b] What Management Thinks ?



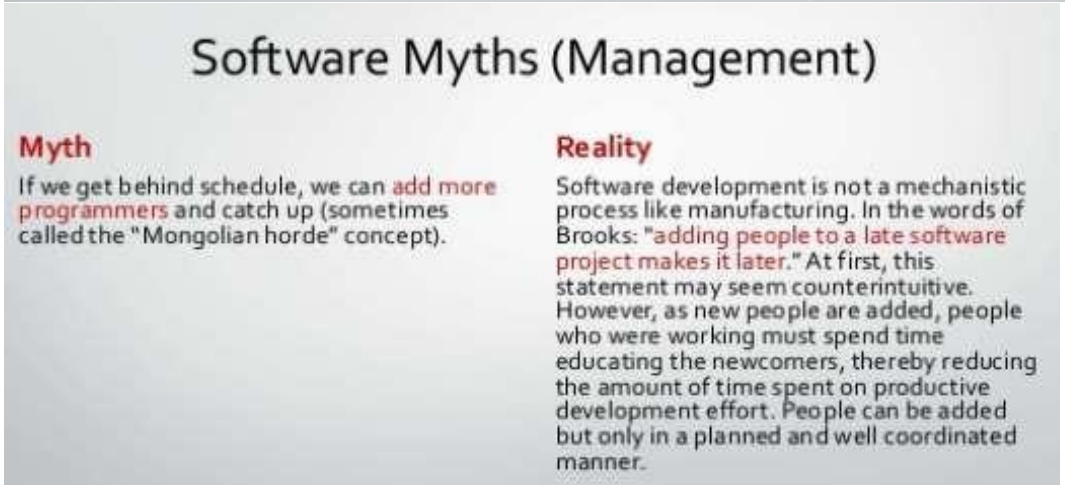
Software Myths (Management)

Myth

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

Reality

The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it adaptable? Is it streamlined to improve time-to-delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."



Software Myths (Management)

Myth

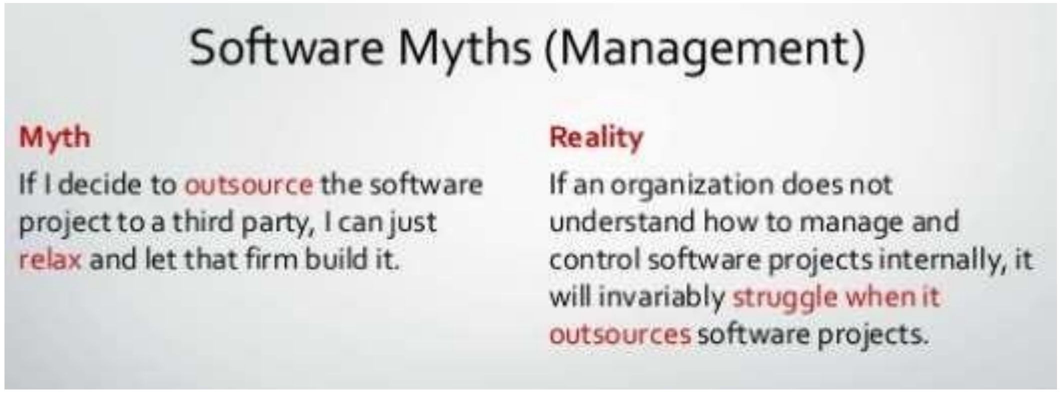
If we get behind schedule, we can add more programmers and catch up (sometimes called the "Mongolian horde" concept).

Reality

Software development is not a mechanistic process like manufacturing. In the words of Brooks: "adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well coordinated manner.

[Continue..]

3. Software Myths: b] What Management Thinks ?

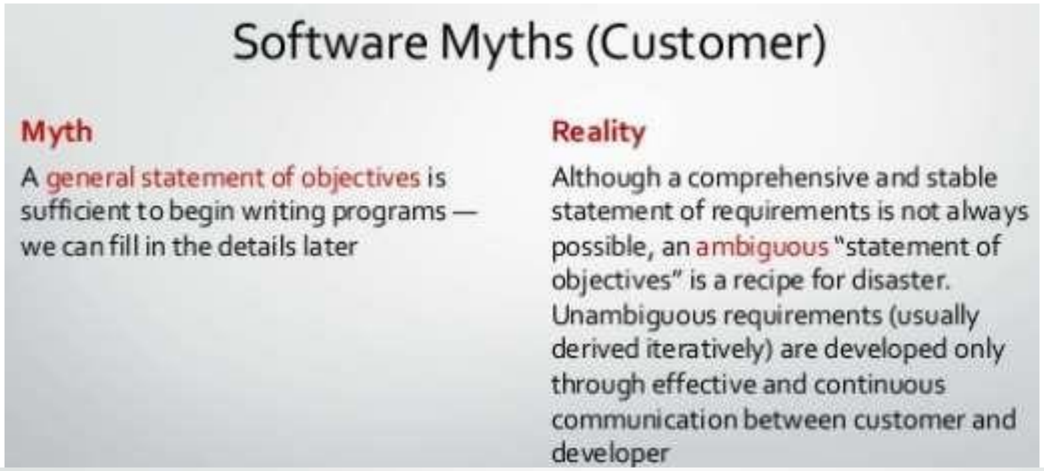


Software Myths (Management)

Myth	Reality
If I decide to outsource the software project to a third party, I can just relax and let that firm build it.	If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

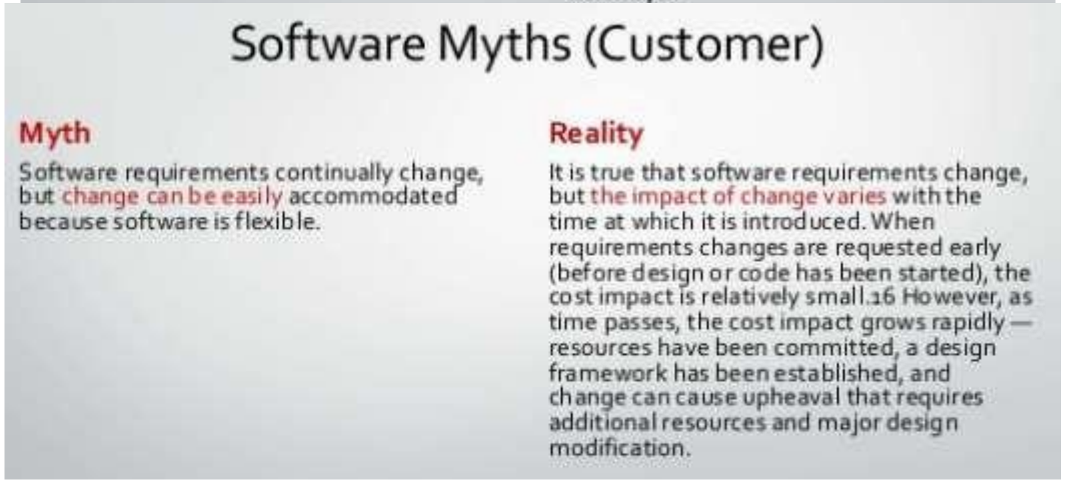
[Continue..]

3. Software Myths: c] What Customer Thinks ?



Software Myths (Customer)

Myth	Reality
A general statement of objectives is sufficient to begin writing programs — we can fill in the details later	Although a comprehensive and stable statement of requirements is not always possible, an ambiguous "statement of objectives" is a recipe for disaster. Unambiguous requirements (usually derived iteratively) are developed only through effective and continuous communication between customer and developer



Software Myths (Customer)

Myth	Reality
Software requirements continually change, but change can be easily accommodated because software is flexible.	It is true that software requirements change, but the impact of change varies with the time at which it is introduced. When requirements changes are requested early (before design or code has been started), the cost impact is relatively small.16 However, as time passes, the cost impact grows rapidly — resources have been committed, a design framework has been established, and change can cause upheaval that requires additional resources and major design modification.

PART 1 : INTRODUCTION

4. Software Process :

Process:

- ❖ It is a collection of activities, actions and tasks that are performed when some work product is created.
- ❖ Software Process can be defined as the structured set of activities that are required to develop the software system.

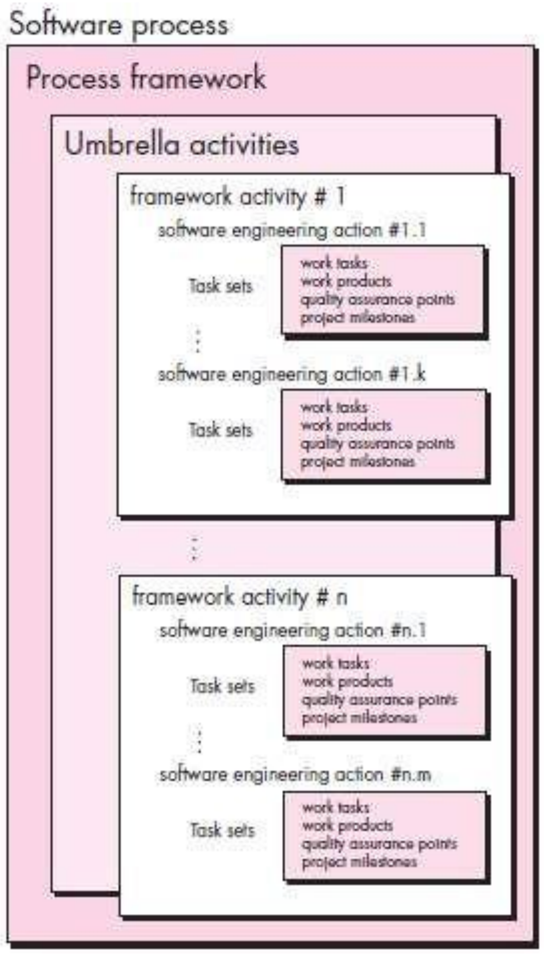
❑ The Fundamental activities are :

1. Specification (comes under Communication)
2. Design & Implementation (comes under Modelling & Construction)
3. Validation (comes under Construction Part : Testing)
4. Evolution (comes under deployment)

- A software process model is an abstract representation of a process.

PART 1 : INTRODUCTION

5. Generic Process Model :



[Continue..]

PART 1 : INTRODUCTION

5. Generic Process Model [1]:

Framework Activities	Umbrella Activities
Communication	Project Tracking & Control
Planning	Risk Management
Modelling	Quality Assurance
Construction	Configuration Management
Deployment	Technical Reviews

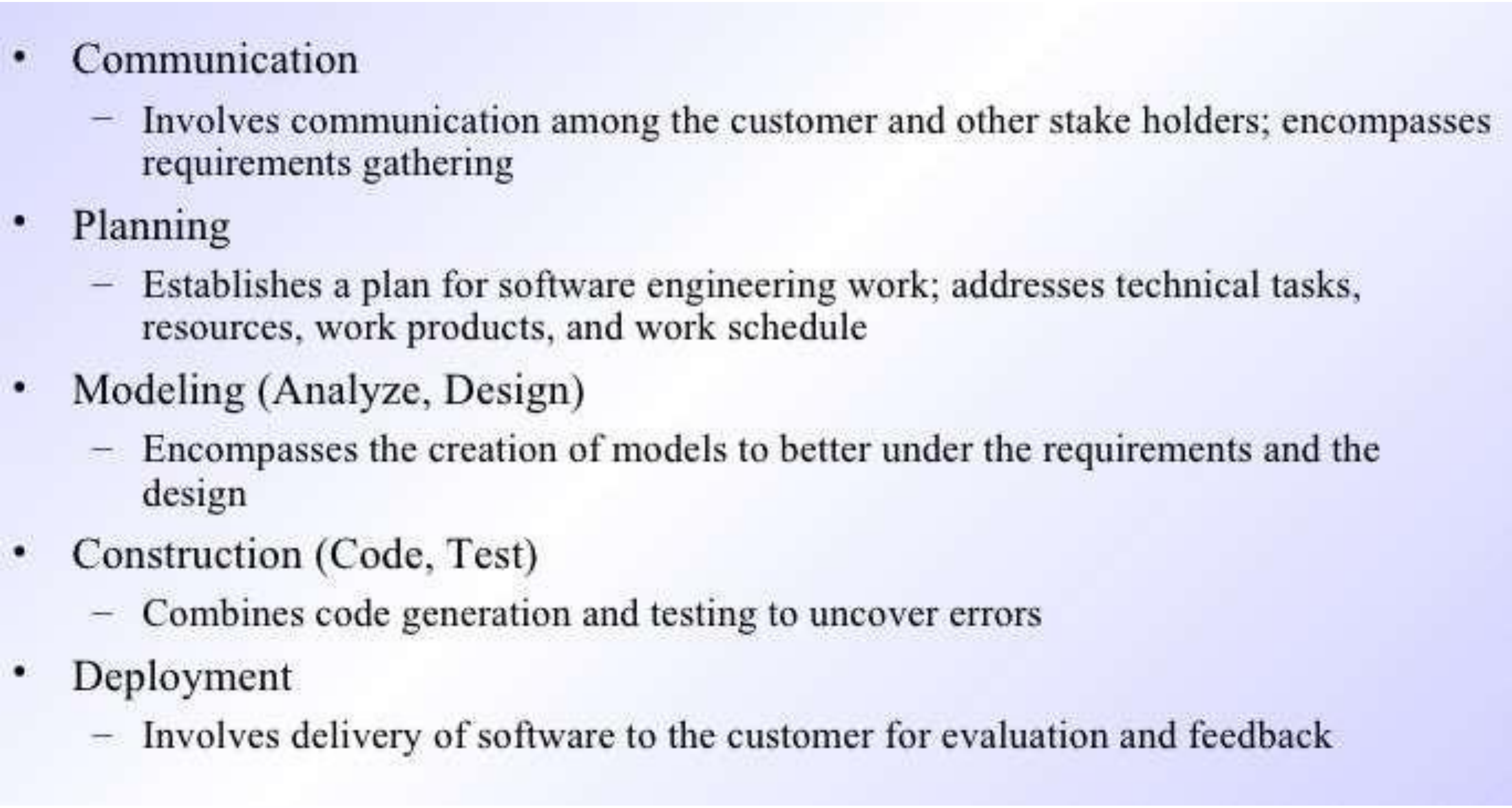
-Each of the Activity , Action , Task reside within framework/model that defines their relationship with the process and with one another.

-**Generic Process Framework** for software Engineering comprise with **five framework activities + A set of Umbrella Activities.**

[Continue..]

5. Generic Process Model [2]:

Framework Activities :



- Communication
  - Involves communication among the customer and other stake holders; encompasses requirements gathering
- Planning
  - Establishes a plan for software engineering work; addresses technical tasks, resources, work products, and work schedule
- Modeling (Analyze, Design)
  - Encompasses the creation of models to better under the requirements and the design
- Construction (Code, Test)
  - Combines code generation and testing to uncover errors
- Deployment
  - Involves delivery of software to the customer for evaluation and feedback

[Continue..]

5. Generic Process Model [3]:

Umbrella Activities :

1. Software Project Tracking & Control.
   - Assessing Progress against a Project Plan.
   - Take Adequate Action to maintain schedule.

2. Risk Management
   - Assesses Risk that may affect the outcome of project or quality of software product.

3. Quality Assurance
   - Defines and conducts the activities required to ensure software quality.

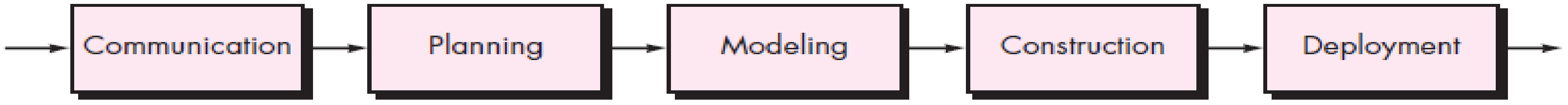
4. Configuration Management
   - Manages the Effect of change.

5. Technical Reviews
   - Assessing software product in an effort to uncover and remove errors before goes into next action or activity.

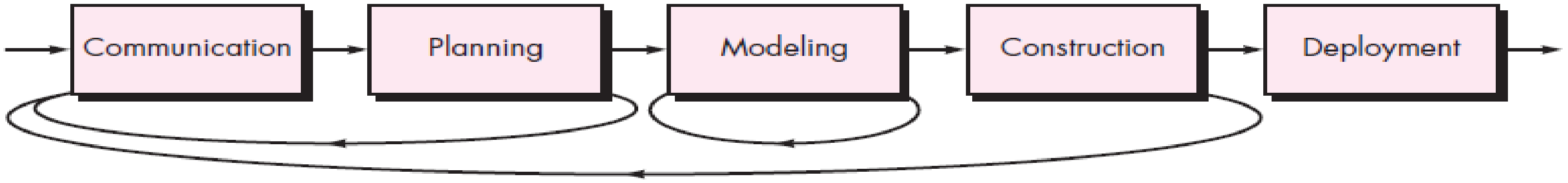
5. Generic Process Model [4]:

PROCESS FLOW :

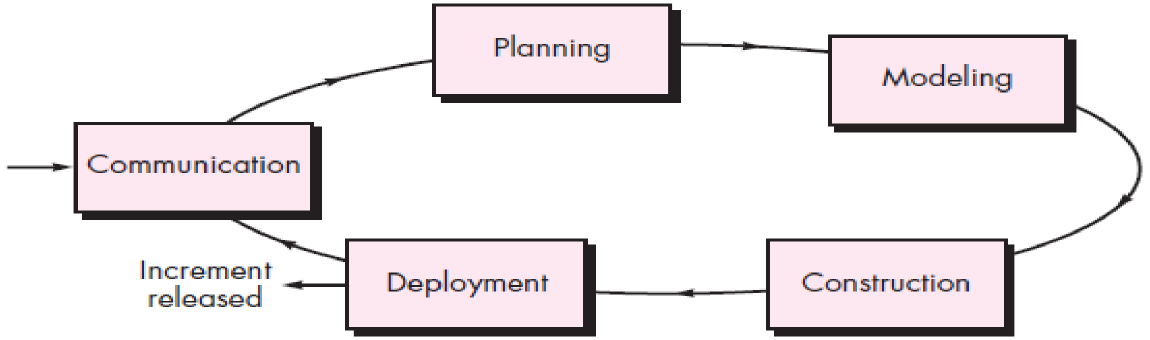
-Describes how framework activities and the actions and the tasks that occur within each framework activity are organized with respect to sequence and time.



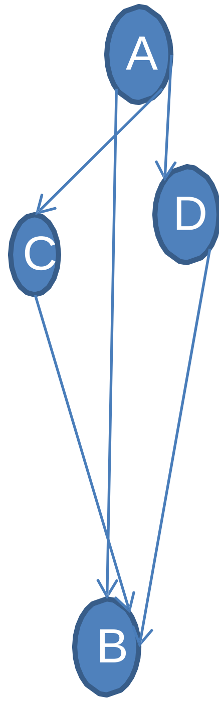
(a) Linear process flow



(b) Iterative process flow



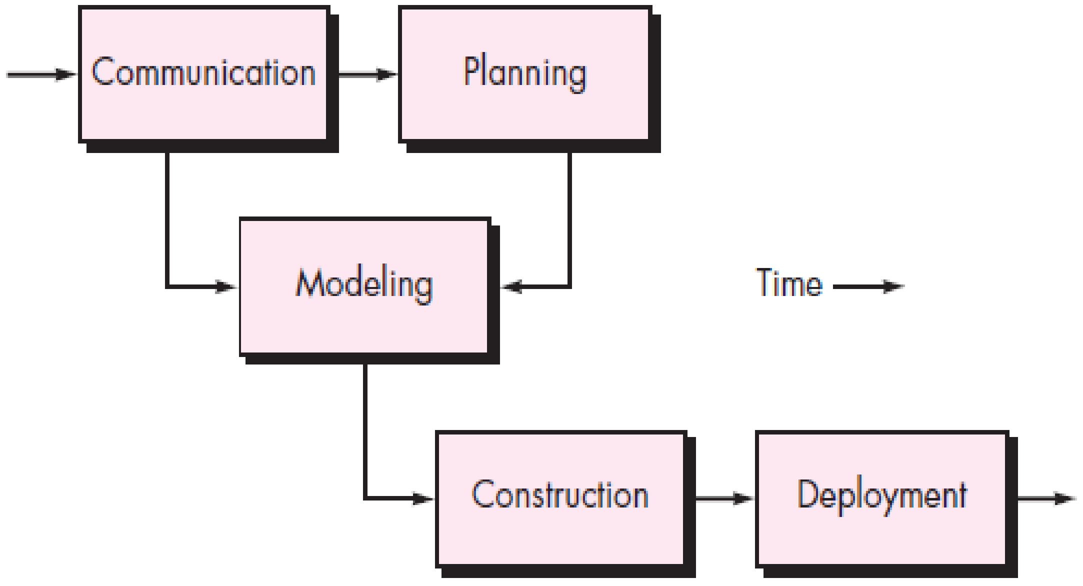
(c) Evolutionary process flow



[Continue..]

5. Generic Process Model [5]:

PROCESS FLOW :

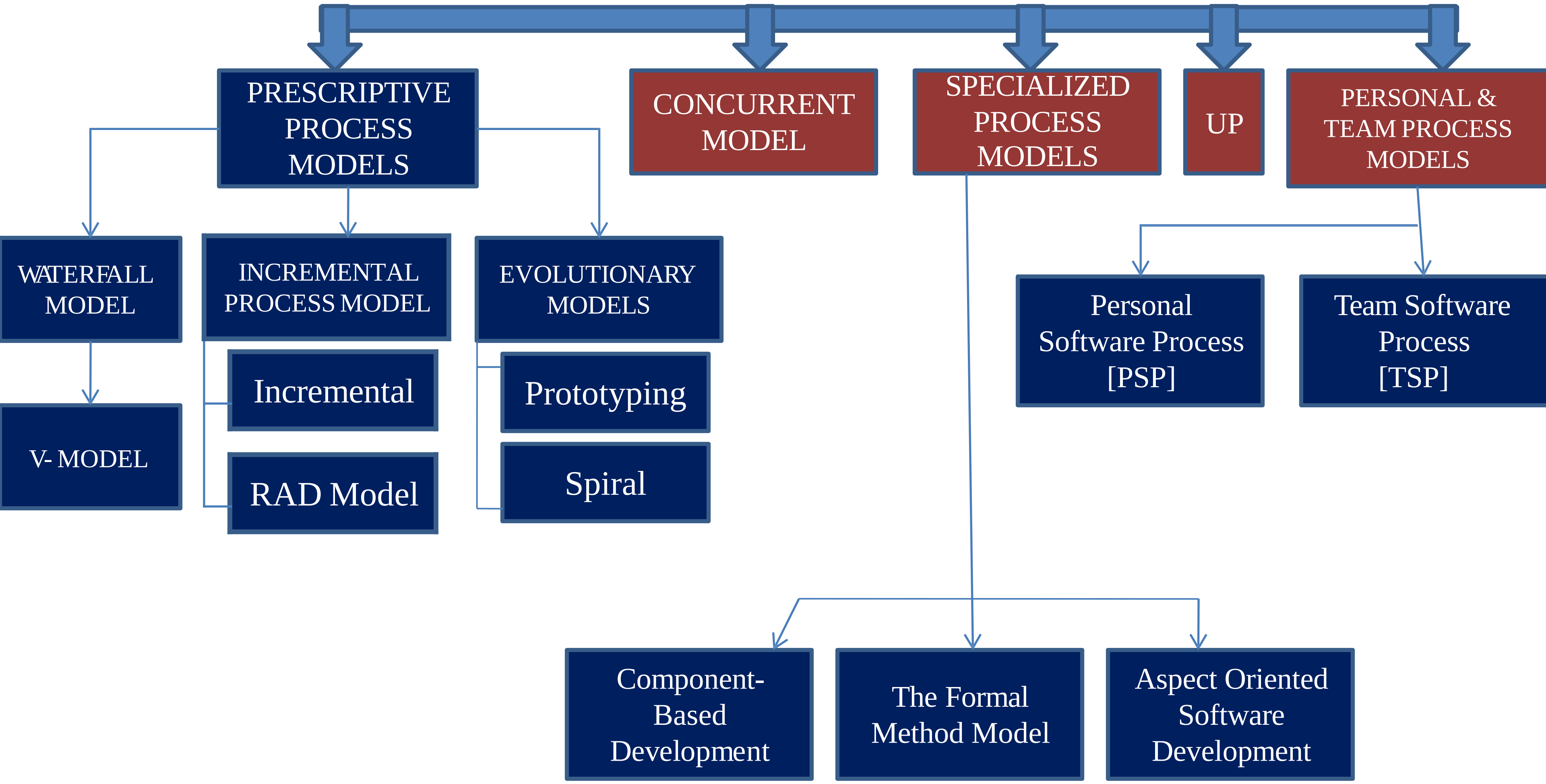


(d) Parallel process flow

PROCESS PATTERN TYPES :1. Stage Pattern 2. Task Pattern 3. Phase Pattern

PART 1 : INTRODUCTION

6. Analysis & Comparisons of Process Models

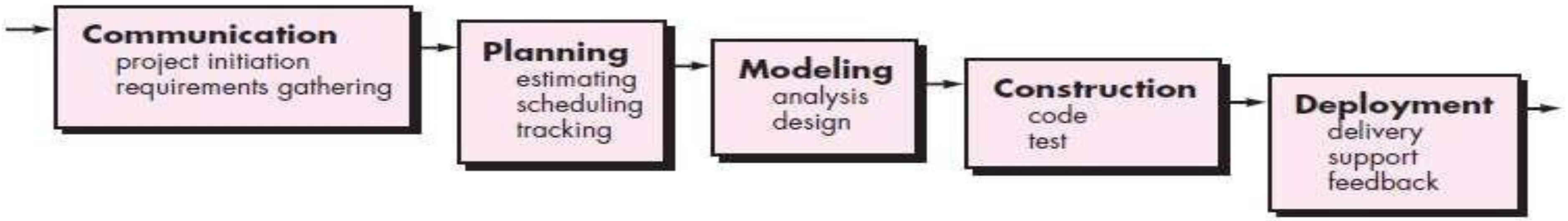

PRESCRIPTIVE PROCESS MODELS
CONCURRENT MODEL
SPECIALIZED PROCESS MODELS
UP
PERSONAL & TEAM PROCESS MODELS
WATERFALL MODEL
INCREMENTAL PROCESS MODEL
EVOLUTIONARY MODELS
Personal Software Process [PSP]
Team Software Process [TSP]
Incremental
Prototyping
V- MODEL
RAD Model
Spiral
Component-Based Development
The Formal Method Model
Aspect Oriented Software Development

PRESCRIPTIVE PROCESS MODEL

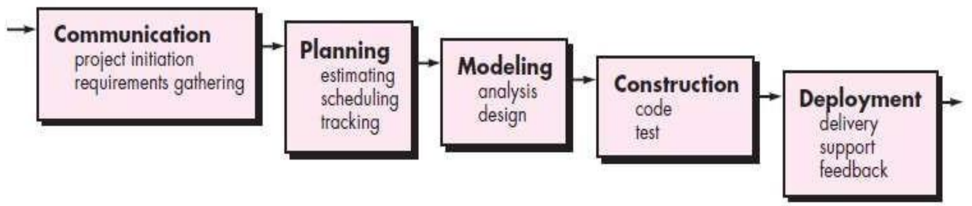
WATERFALL MODEL

-Also Known as “Linear-Sequential Model” & “Classic Life cycle Model”.

-OLDEST Model & Suggest systematic, sequential approach to Software Development.



WATERFALL MODEL (2)



- Features :

- Customer “Must” state all the requirements at the beginning of the project. All these requirements are well documented and discussed further with customer for reviewing.

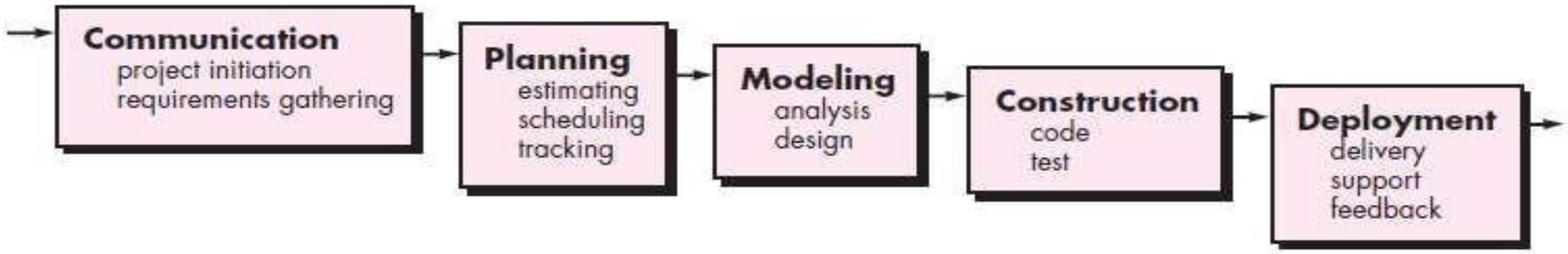
- Supports “Blocking State Mechanism” i.e. Programmer cannot start coding step unless and until the design of project is completed.

- Maintenance/Deployment is the “Longest life cycle phase” in order to correct installation error as well as Enhancing System Services as new requirements are discovered during maintenance of system.

WATERFALL MODEL (3)

- When to choose ?

- Requirements are very well known , clear and fixed.
- Product Definition is stable.
- Technology is understood.
- There is no ambiguous requirement.
- Ample resources with required expertise are available freely.
- The Project is short.



WATERFALL MODEL (4)

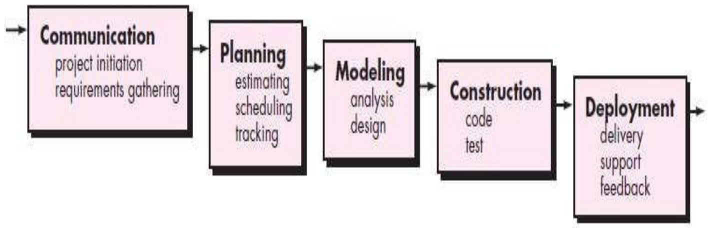
- ADVANTAGES :

✓ Simple to implement.

✓ useful for implementation of small system.

✓ Easy to manage due to rigidity of model.

✓ Phases are processed & Completed one at a time.



- DISADVANTAGES :

× Problems in this model remains uncovered until software Testing.

× Waiting State Mechanism , huge amount of time spent in waiting rather than spending it on productive work.

× You can not go back 2 steps ; if design phase goes wrong , things can get very complicated in implementation phase.

× High amount of risk and uncertainty.

× Not a good model for complex & object oriented projects.

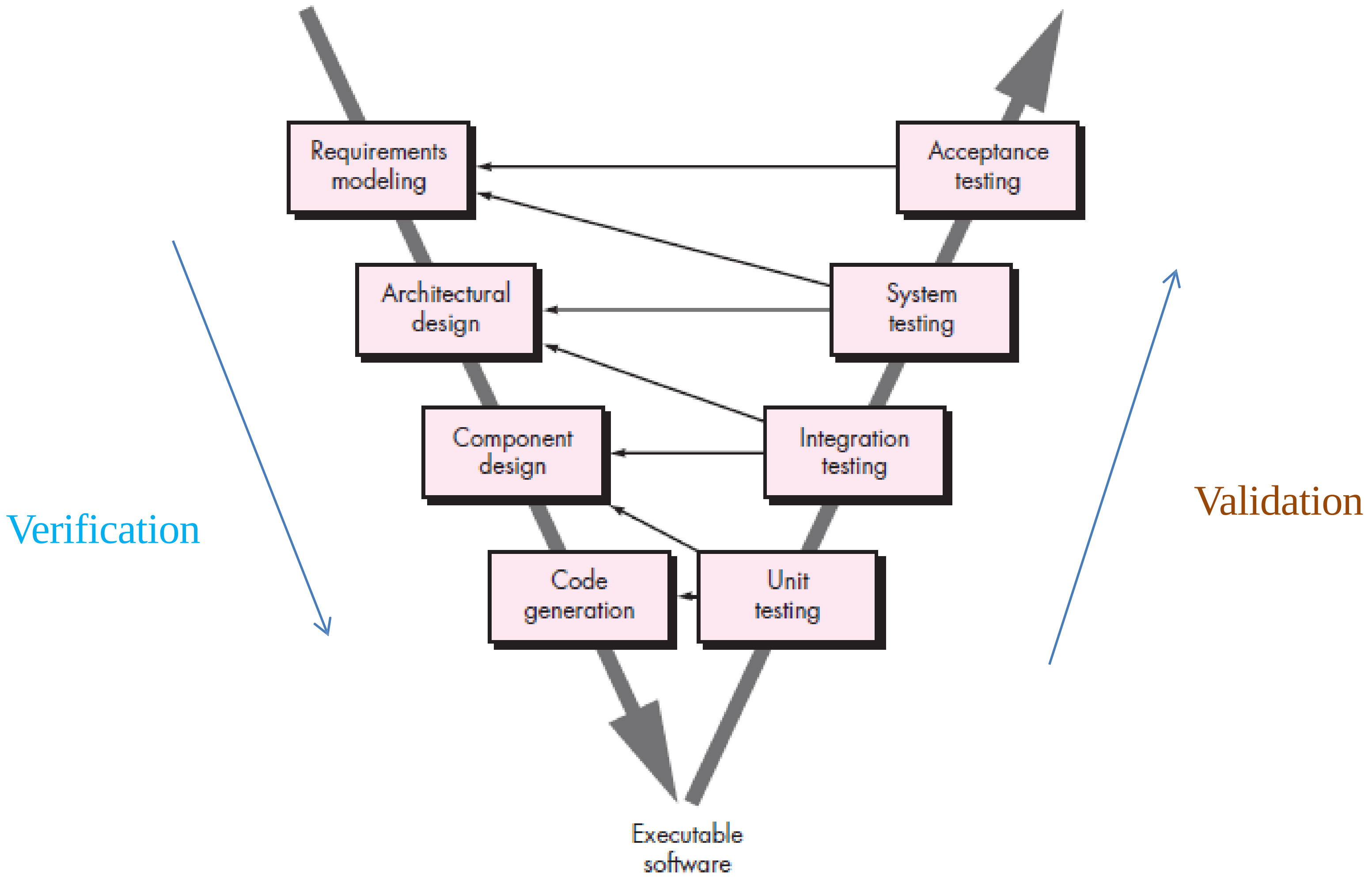
× Poor Model for long and on-going projects.

× Not suitable for projects where Requirements are at a moderate to high risk of changing

× Customer must have patience as working version of the program will not be available until late in project time span.

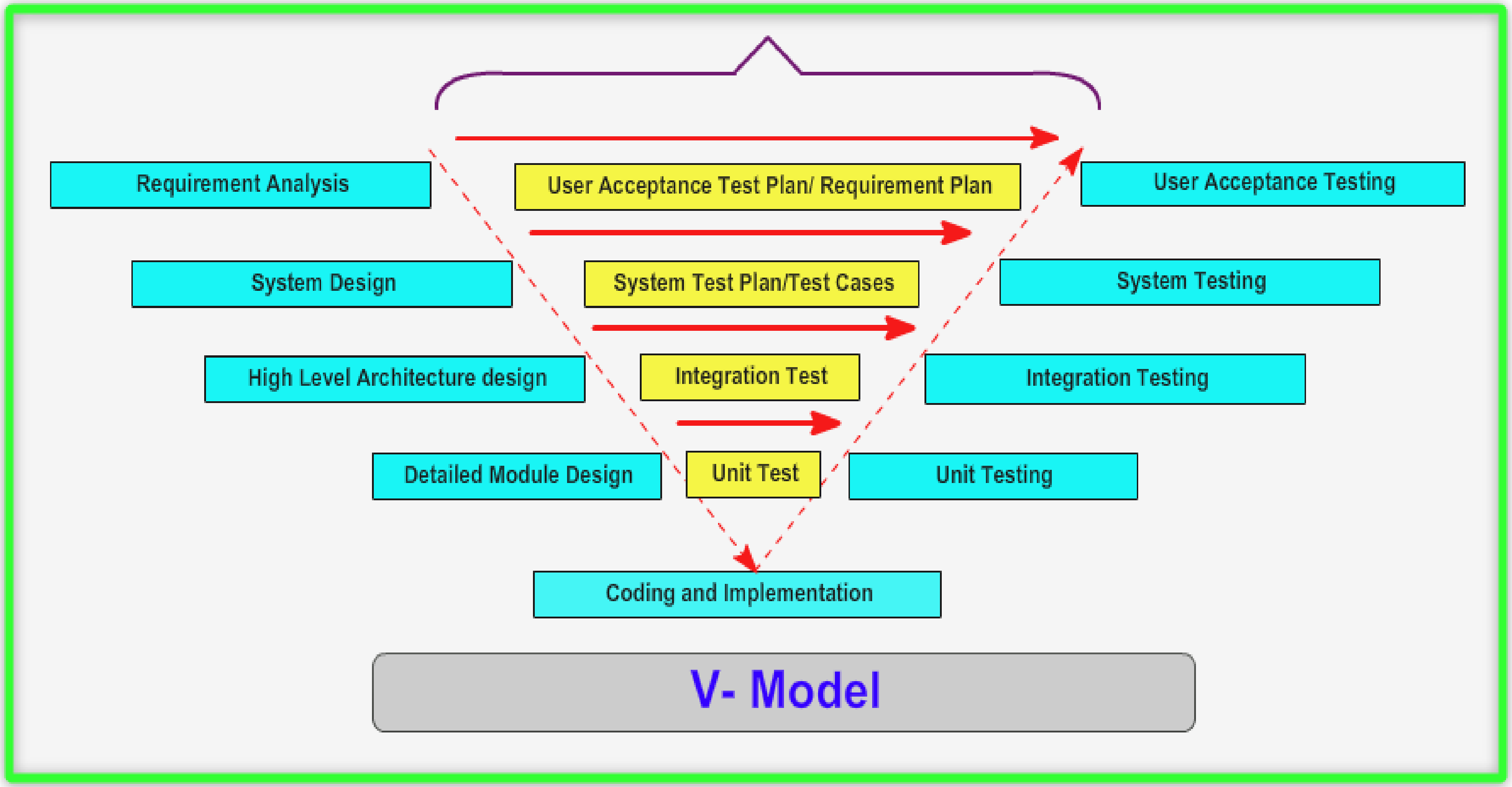
V - MODEL

- It is an Extension of the Waterfall Model.

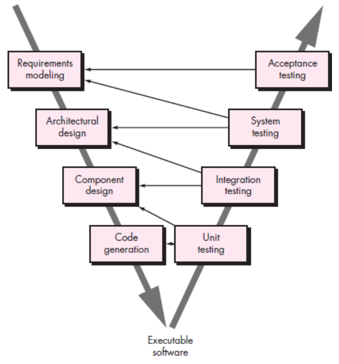


V – MODEL (1)

- The Left Arm of V-Model is a Conventional waterfall model of the development & Right Arm represents corresponding validation or testing levels.



V- MODEL (2)



- Features :

- Deliverables of each phase shown on left arm undergo verification. Validation is conducted at different levels namely unit , Integration, System and acceptance testing.

- Each phase of development provides input to the respective test plan used in validation. E.g. Acceptance test plan can be made ready once user requirements are captured & verified.

- Each verification activity has its corresponding Validation activity.

V- MODEL (3)

- ADVANTAGES :

✓ Highly Disciplined model. Phases completed one at a time

✓ Works well with smaller projects where requirements are very well understood.

✓ Simple & Easy to understand & Use.

✓ Easy to manage due to rigidity of the model. Each phase has specific deliverables and a review process.

- DISADVANTAGES :

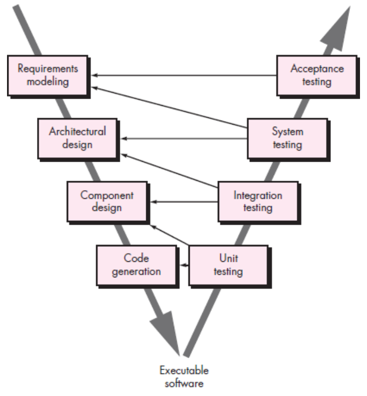
× High Risk & Uncertainty.

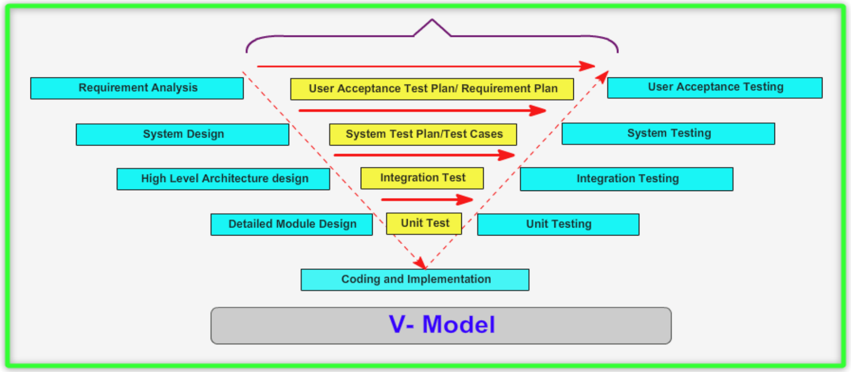
× Not a good model for complex and Object-oriented Projects

× Poor Model for Long & Ongoing Projects

× Not Suitable for the Projects where requirements are at moderate to high risk of changing.

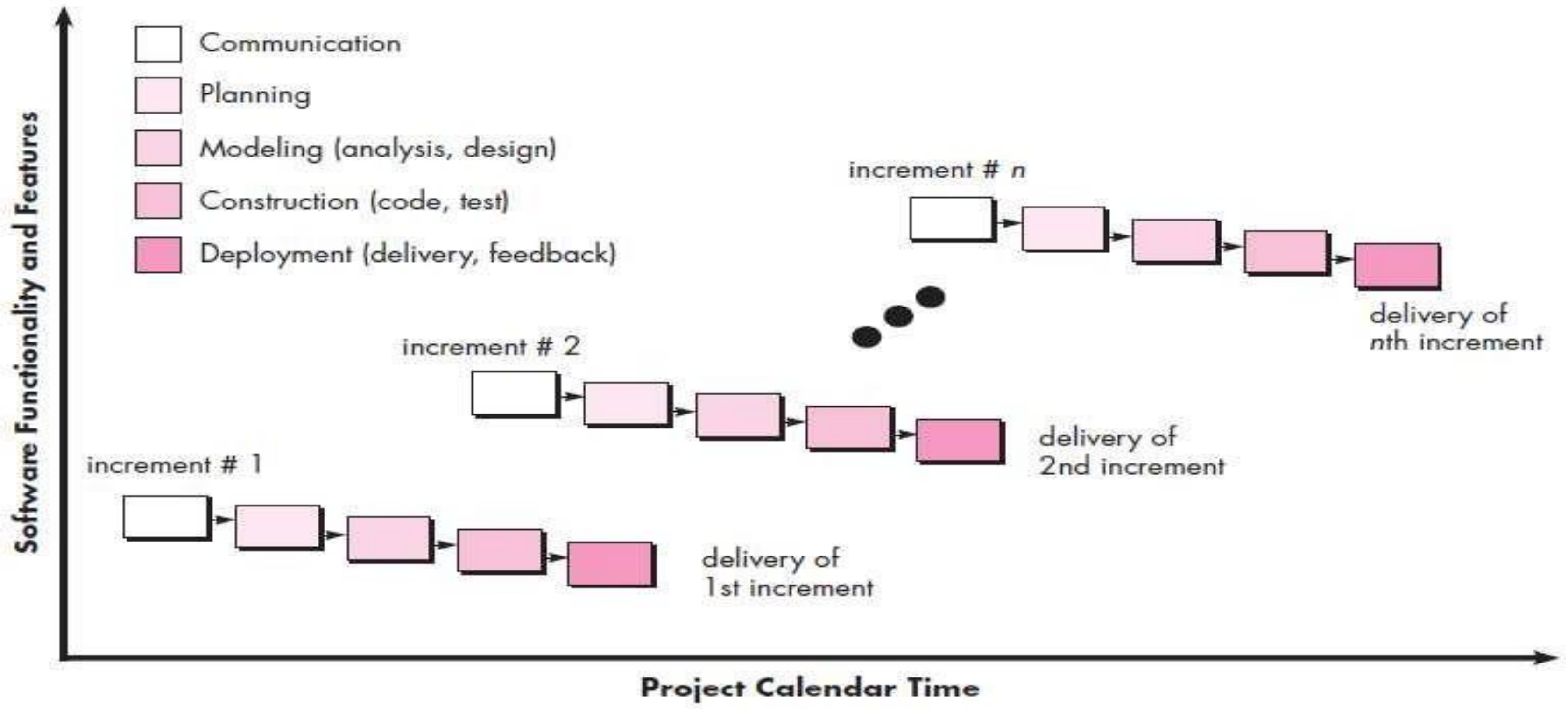
× No working software is produced until late during the life cycle.





INCREMENTAL MODEL

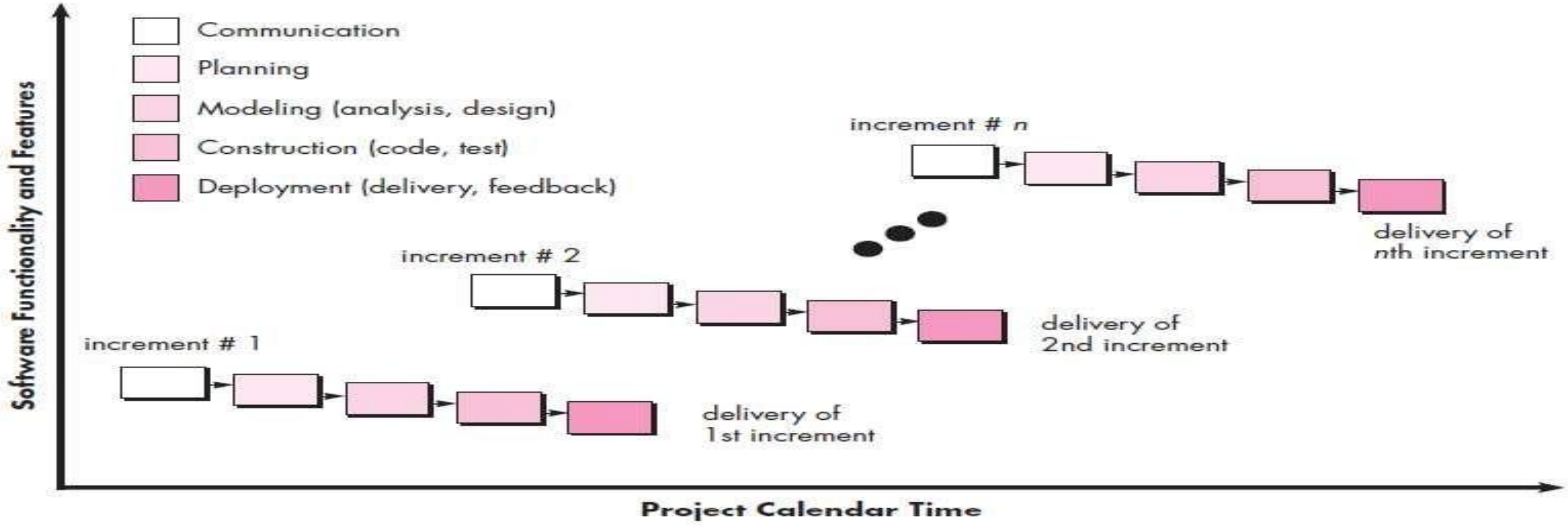
- It is an Evolution of the Waterfall Model.
- It Combines elements of waterfall model (applied repeatedly) with the Iterative philosophy of prototyping.


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # 1
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # n
delivery of nth increment
Software Functionality and Features
Project Calendar Time

INCREMENTAL MODEL (2)

- In this Model , the initial model with limited functionality [core product] is created for user's understanding about the software product and then this model is refined and expanded in later releases.

- In this model Progressive Functionalities are obtained with each release.

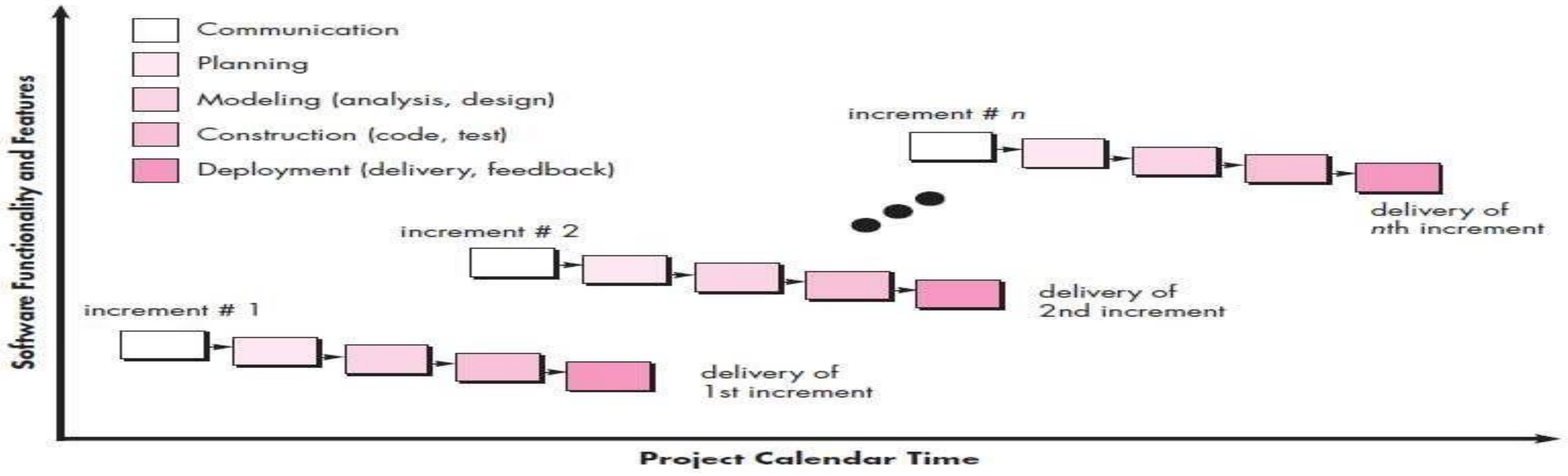


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
increment # 2
increment # 1
delivery of nth increment
delivery of 2nd increment
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

INCREMENTAL MODEL (3)

- When to choose ?

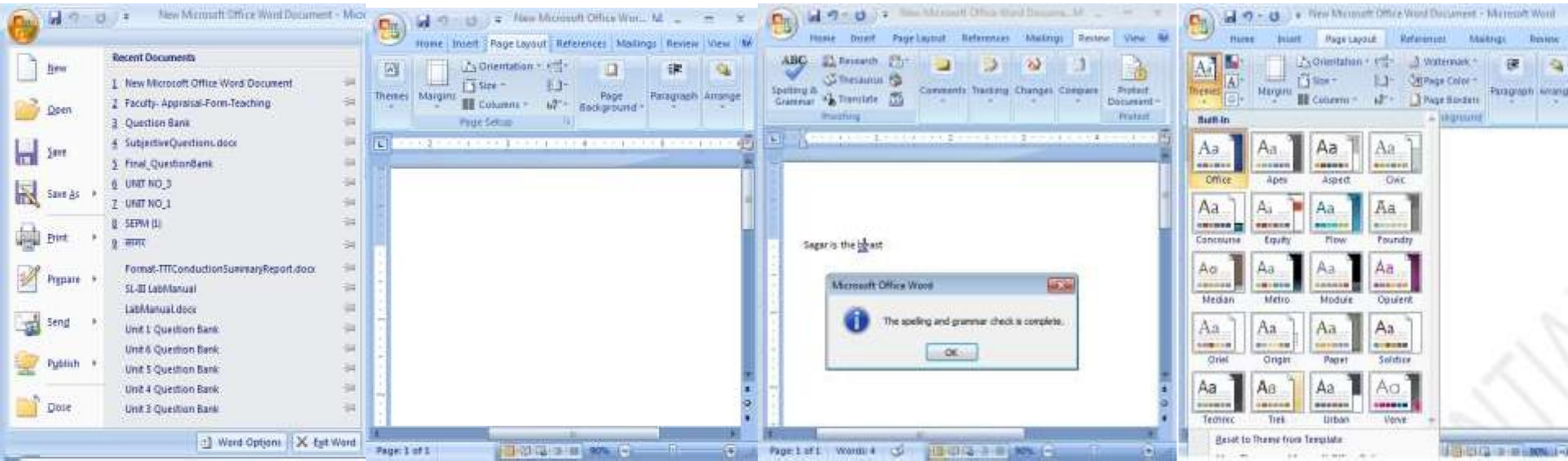
- Requirements are reasonably well defined.
- Overall scope of the development effort suggests a pure linear effort.
- When limited set of software functionality needed quickly.


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

INCREMENTAL MODEL (4)

Example : Development of Word Processing software. [MS-WORD]

ITERATION #	DESCRIPTION
1	Basic File Management Functions : New , Open , Save , Save as etc. can be implemented.
2	More Sophisticated Editing & Document Production Capability can be implemented. E.g. Rulers, Margins etc.
3	Spelling , Grammar Checking and auto correction of words is implemented.
4	The Final advanced page layout capabilities are developed. And so on till the product is finished.



1 New Microsoft Office Word Document
2 Faculty- Appraisal-Form-Teaching
3 Question Bank
4 SubjectiveQuestions.docx
5 Final_QuestionBank
6 UNIT NO_3
7 UNIT NO_1
8 SEPM (1)
9 [illegible]
Format-TTTConductionSummaryReport.docx
SL-III LabManual
LabManual.docx
Unit 1 Question Bank
Unit 6 Question Bank
Unit 5 Question Bank
Unit 4 Question Bank
Unit 3 Question Bank

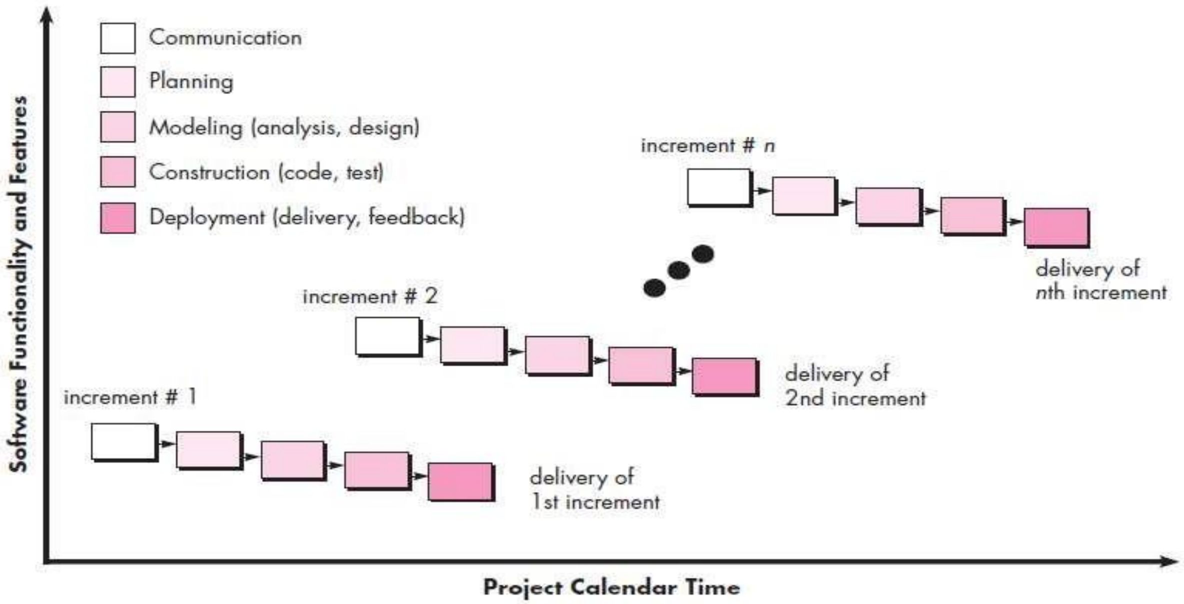
Sagar is the best

The spelling and grammar check is complete.

INCREMENTAL MODEL (5)

- ADVANTAGES :
  - ✓ Useful when the size of development team is small at beginning or some team members are not available. Thus Early Increments can be implemented with small size of team
  - ✓ if product satisfies client ,then size of team can be increased.
  - ✓ All Increments are properly planned to handle all types of technical risks.
  - ✓ The Initial product delivery is faster and cost of development is low
  - ✓ The Customers can respond to the functionalities after each increment and come up with feedback.

- DISADVANTAGES :
  - × The cost of finished Product may be Increased in the end beyond the cost estimated.
  - × After Each Increment , the customer can demand for additional functionalities that may cause serious problem to system architecture..
  - × It needs very clear planning and design before the whole system is broken into small increments.



Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of nth increment
increment # 2
delivery of 2nd increment
increment # 1
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

RAD MODEL

- It is Rapid Application Development Model. It is the type of incremental model.

- Multiple Teams work in Parallel way on developing the software system using RAD Model .

- "Extremely Short Development Life Cycle"

- Using RAD Model , fully functional system can be developed within 60-90 Days.

- "Build/Patch" concept is developed from RAD Model

RAD MODEL (2)

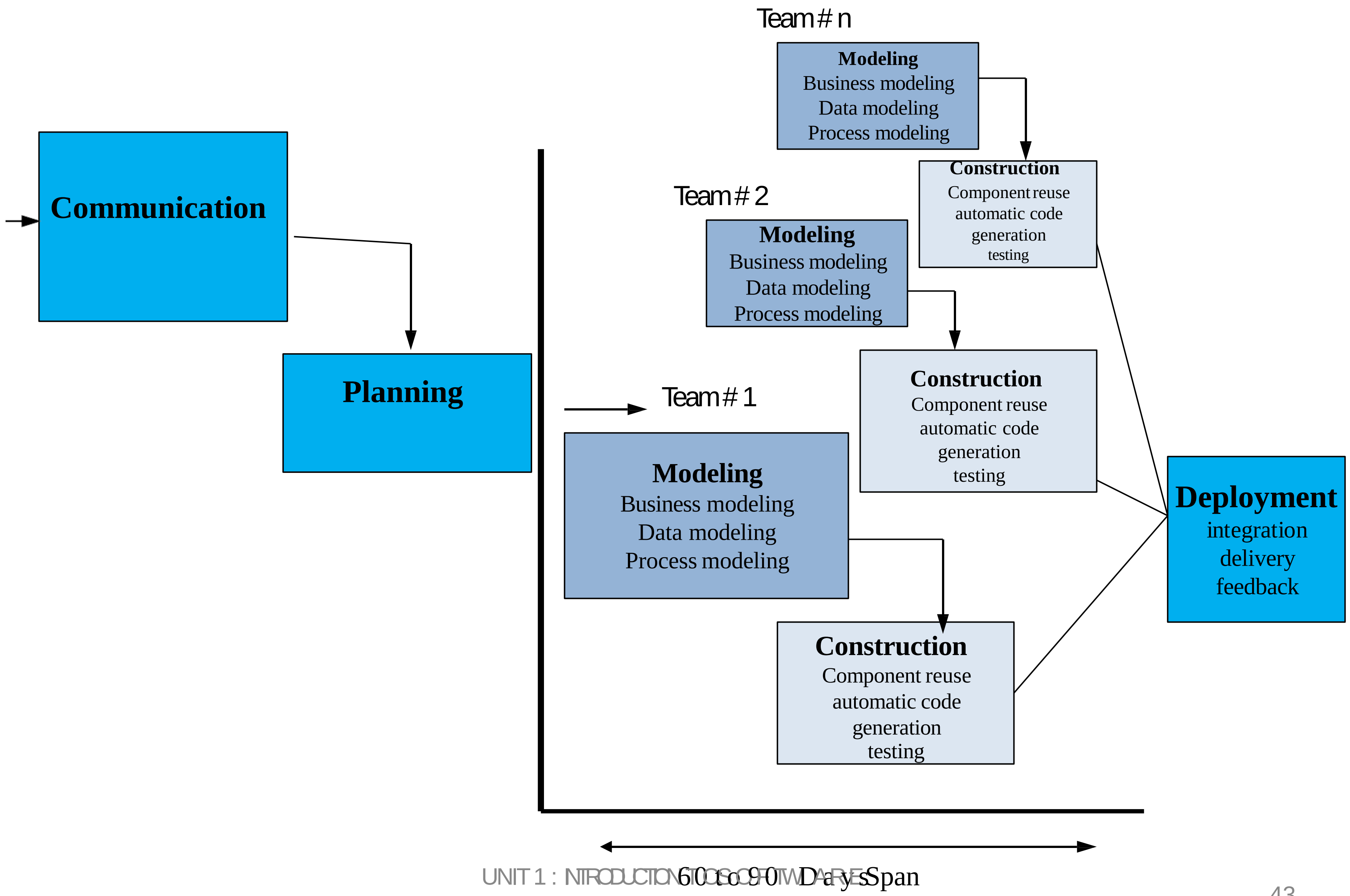
- When to choose ?

- Requirements are fully understood.
- Component-based approach is adopted.

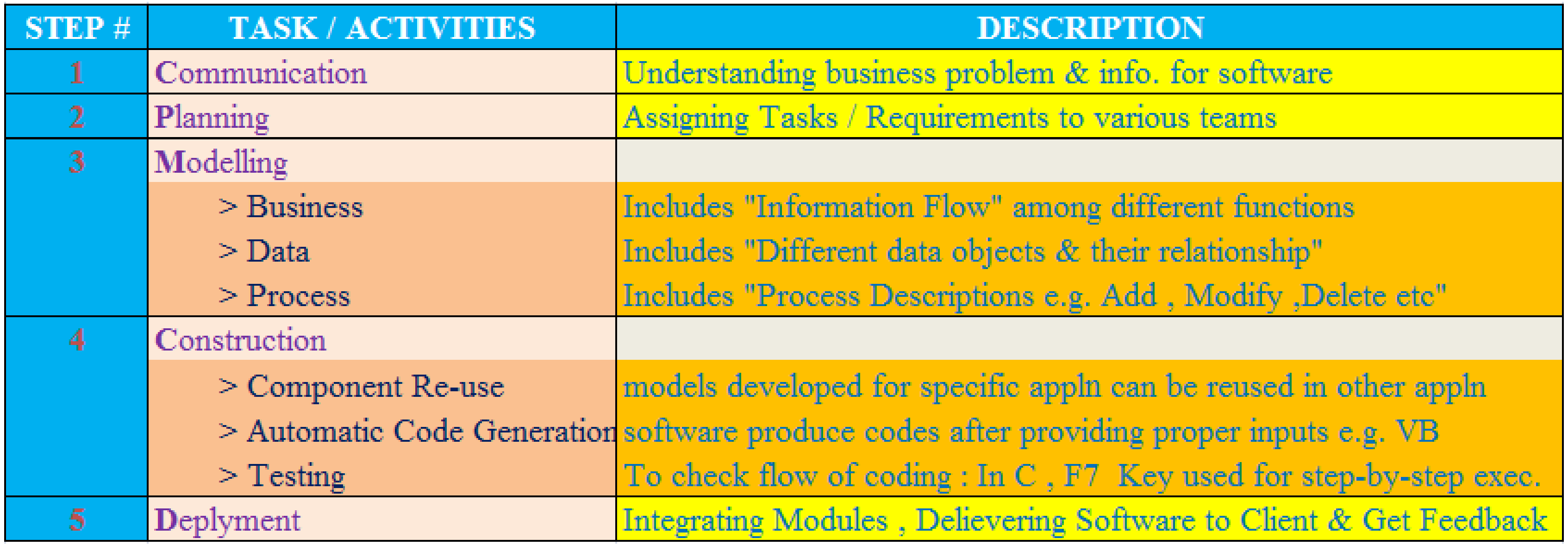
- PROCESS :

1. The initial activity starts with communication between customer and developer.
2. Depending upon initial requirements the project planning is done.
3. Now Requirements are divided into different groups & each requirement is assigned to different teams.
4. When all teams are ready with their final product , the product of each team is integrated i.e. combined to form a product as whole.

RAD MODEL (3)


Team # n
Modeling
Business modeling
Data modeling
Process modeling
Construction
Component reuse
automatic code
generation
testing
Team # 2
Modeling
Business modeling
Data modeling
Process modeling
Construction
Component reuse
automatic code
generation
testing
Communication
Planning
Team # 1
Modeling
Business modeling
Data modeling
Process modeling
Construction
Component reuse
automatic code
generation
testing
Deployment
integration
delivery
feedback
60 to 90 Days Span

RAD MODEL (4)



STEP #	TASK / ACTIVITIES	DESCRIPTION
1	Communication	Understanding business problem & info. for software
2	Planning	Assigning Tasks / Requirements to various teams
3	Modelling	
	> Business	Includes "Information Flow" among different functions
	> Data	Includes "Different data objects & their relationship"
	> Process	Includes "Process Descriptions e.g. Add , Modify ,Delete etc"
4	Construction	
	> Component Re-use	models developed for specific appln can be reused in other appln
	> Automatic Code Generation	software produce codes after providing proper inputs e.g. VB
	> Testing	To check flow of coding : In C , F7 Key used for step-by-step exec.
5	Deplyment	Integrating Modules , Delievering Software to Client & Get Feedback

RAD MODEL (5)

- ADVANTAGES :

✓ Product can developed with short period of time

✓ Supports increased re-usability of component

✓ Minimal code writing as it supports automatic code generation

✓ Encourages the customer feedback

- DISADVANTAGES :

× Requires sufficient man-power to create no. of teams

× All teams must work with equal speed . If one team lags behind the schedule results in late delivery of overall project.

× Improper Modularized results in Problematic in terms of building the components.

× This approach is useful for only larger projects.

× REDUCED SCALABILITY : as RAD application evolves from prototype to a finished application. So less scope.

× REDUCED FEATURES : Because of "Time Boxing" Concept. Due to short amount of time features are pushed to be completed in later releases.

EVOLUTIONARY PROCESS MODEL

- "Limited Version" concepts introduced to meet business/competitive pressure.

- A set of core product/system requirements is well understood , but details of product or system extensions are yet to be defined.

- Evolutionary Process are iterative in nature to develop increasingly more complete versions of software.

- Two Types : 1. Prototype Model   2. Spiral Model

PROTOTYPING MODEL

- It assist you and other stakeholders what to built when requirements are fuzzy(confused).

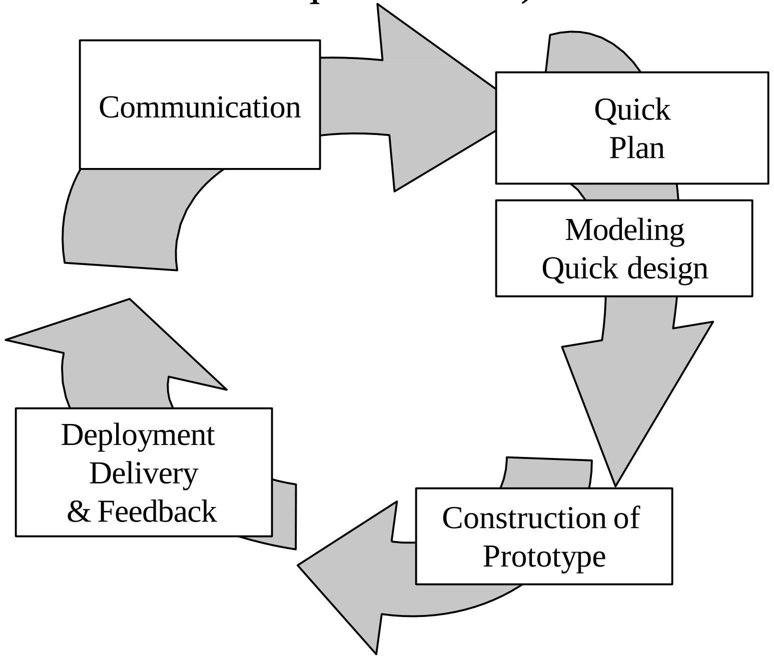
- In this Model , the planning & Modelling are “Quick” in nature.

- The Prototype can serve as “First system ” i.e. some are “Throwaway” , others are “Evolutionary” i.e. that Prototype slowly evolves into the actual system.

PROTOTYPING MODEL (1)

PROCESS :

1. Communication : Meeting stakeholders to define overall objectives of software , identify known requirements
2. Quick Plan : Prototype Iteration planned Quickly.
3. Quick Modelling (Design) : Representation of aspects of s/w that will be visible to end-user
4. Construction of Prototype : Quick Design Leads to this phase.
5. Deployment : The Prototype is Deployed & Evaluated by Stakeholders , who provide feedback (Purpose : To Further Refine Requirements)



PROTOTYPING MODEL (2)

- ADVANTAGES :

✓ Well suited for projects where Requirements are Hard to Determine at Beginning.

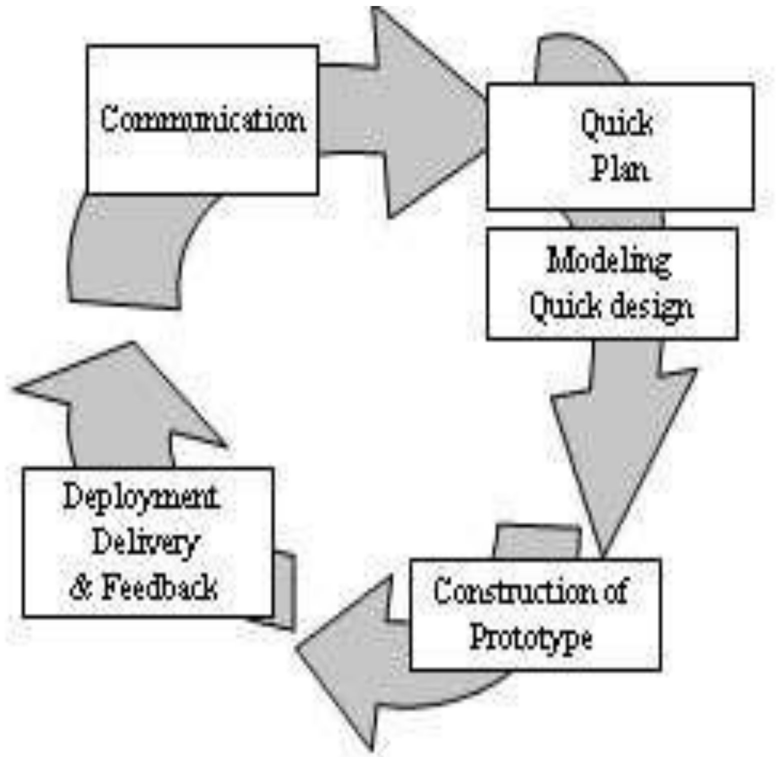
✓ It is an attractive idea for Complicated and Large system for which there is no manual process for determining the Requirements.

✓ Prototype makes system more Transparent.

- DISADVANTAGES :

× Developer often makes Implementation Compromise in order to get a Prototyping work Quickly.

× Sometimes customer may be unaware that prototype is not a product . Development cost may become large.



SPIRAL MODEL

- Its an Evolutionary Model which combines the best features of waterfall model & the iterative nature of Prototype Model.

SPIRAL MODEL = WATERFALL MODEL + PROTOTYPE MODEL.

- In this Model , more emphasis is given on "**Risk Analysis**" .

- It Consist of 4 Phases : Planning , Risk Analysis , Engineering & Evaluation.

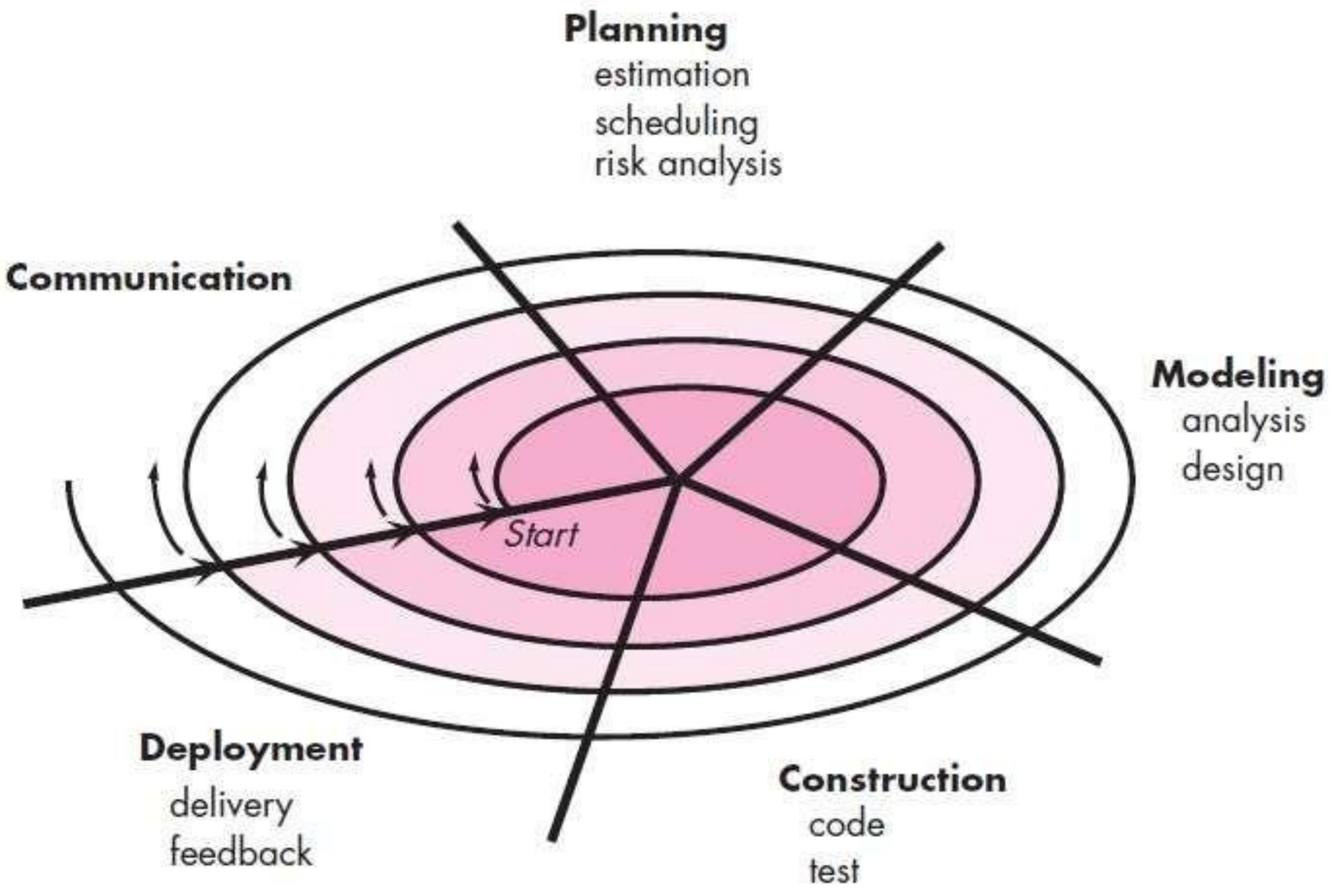
- A software project repeatedly passes through these phases in iterations (called Spirals in this model).

SPIRAL MODEL (2)

- When to choose ?

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

SPIRAL MODEL (3)

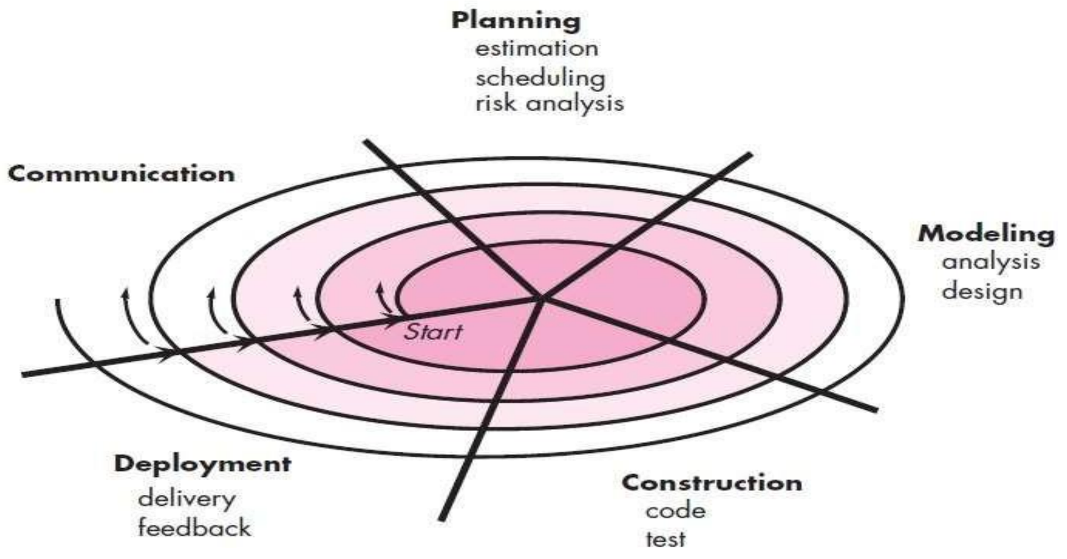


STEP #	PHASES	DESCRIPTION
1	Planning Phase	Requirements are gathered during the planning phase [SRS]
2	Risk Analysis Phase	To Identify Risk & Alternative solution. Protoype is formed at end.
3	Engineering Phase	Software Is Developed along with coding.
4	Evaluation Phase	Allows the cutomer to Evaluate ooutput of project before next spiral

SPIRAL MODEL (4)

- ADVANTAGES :
  - ✓ Supports High Amount of Risk Analysis Resulting Increase in Avoidance of Risks.
  - ✓ Good for Large & Mission-Critical Projects.
  - ✓ Strong Approval & Documentation Control.
  - ✓ Additional Functionality can be added in the Later Date.

- DISADVANTAGES :
  - × Costlier Model To Use.
  - × Risk Analysis requires Highly Specific Expertise.
  - × Projects Success is Highly Dependent on the Risk Analysis Phase.
  - × Doesn't Work Well for smaller Projects.



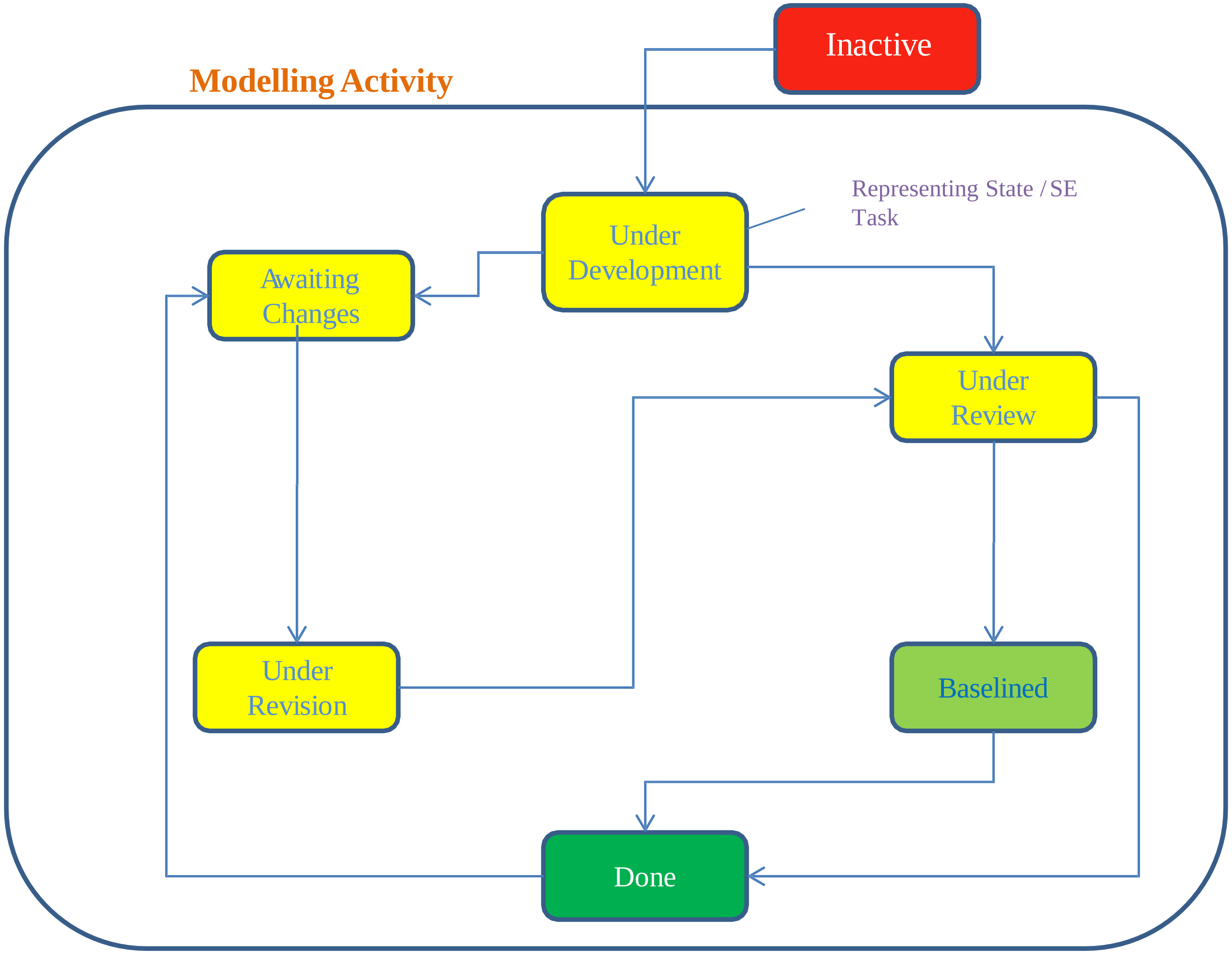
CONCURRENT MODEL

- Also known as Concurrent Engineering.
- Provides accurate view of the current state of project.
- It defines a network of software engineering activities as well as various concurrent activities.
- Changes in one activity can trigger transition among states of an activity.
- Applicable to All types of software development.

CONCURRENT MODEL (1)

-Early In a Project , Communication activity has completed its first iteration

- If Customer Indicates that Change In Requirements must be made…Then. . .
  If Inconsistency in the requirements model is uncovered. . . . . .Then. . . .



CONCURRENT MODEL (2)

- ADVANTAGES :

✓ It is Applicable to ALL types of software development processes

✓ It gives very clear picture of current state of the Project.

✓ Easy to Use & Understand

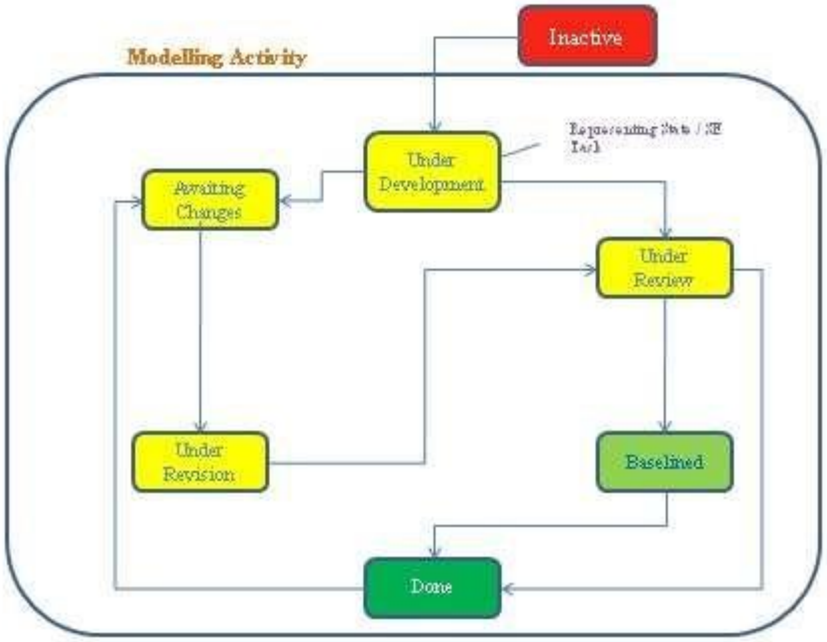
✓ Flexible

- DISADVANTAGES :

× Due to Concurrent Working , any change in Requirement from client may halt the Progress because of Dependency.

× Requires Excellent & Updated communication between Team Members. This may not be achieved all the time.

× SRS must be updated at Regular Intervals to reflect the changes.



SPECIALIZED PROCESS MODEL

- It has many characteristics of one or more traditional models.

- They are used when only collection of specialized technique or methods are expected for developing the specific software.

- Types of Specialized Models:

1. Component-Based Development
2. Formal Methods Model
3. Aspect Oriented Software Development.

Component Based Development

- COTS Components (Commercial off-the-shelves : Products Ready-Made & Available for sale to General Public) are used during software built.
- COTS having Specialized Targeted Functionalities and Well Defined Interfaces.
- Easy to integrate these components in Existing software.
- It leads to "Software Re-Use" (reduction in development cycle time as well as project cost).
- It makes use of various characteristics of Spiral Model.
- It is "Evolutionary" in nature.

Component Based Development (1)

❖ STEPS :

1. Available component-based products are researched & 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. Conduction of Comprehensive testing to ensure proper functionality.

Examples of COTS Software :

SPECIALIZED PROCESS MODEL

Formal Methods Model

- It consists of activities that leads to Formal Mathematical Specification of Computer software .
- It has provision to apply a rigorous (extremely through & careful) , mathematical notation.
- Ambiguity , Incompleteness and Inconsistency can be discovered & easily corrected by applying mathematical formulation in Design Phase (During Verification).
- It produces "Defect-Free" Software.
- Examples : Aircraft Avionics Developers , Medical Devices , Cleanroom Software Engineering.

The Formal Methods Model (1)

❖ CRITICAL ISSUES :

1. Development is quite Time-Consuming & Expensive.

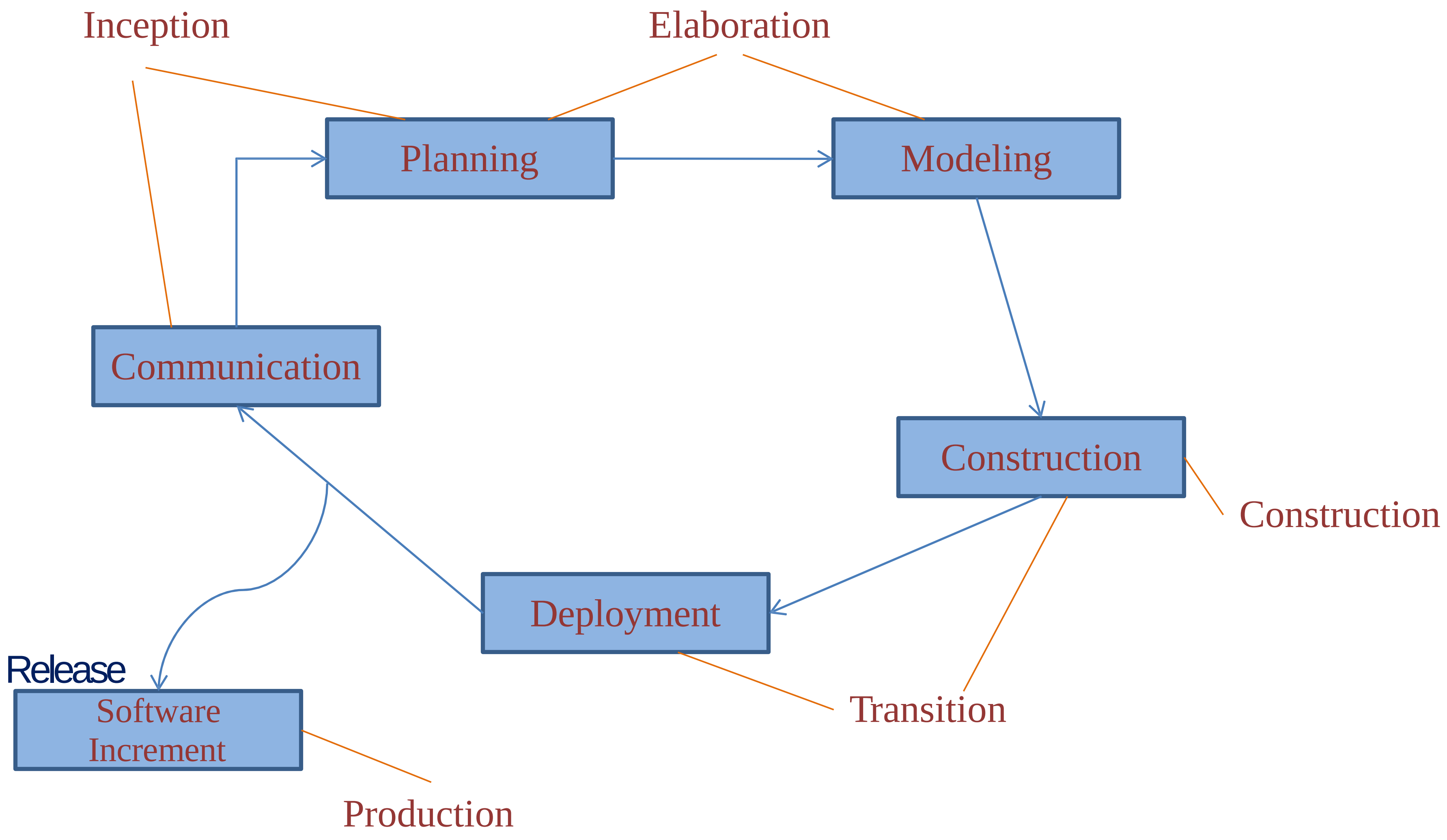
2. Requirement of Extensive training .

3. It is difficult to use Models as a communication mechanism for technically unsophisticated customers.

Aspect-Oriented Software Development [AOSD]

- It adopts both Spiral & Concurrent Process Models.
- The Modern Computer-based systems are more complex in nature resulting in Arising of Some concerns [Security , Task Synchronization , Memory Management]
- When concerns are cut across multiple system functions , features and information , they are known as cross-cutting concerns.
- Aspectual Requirements define those crosscutting functions that have impact across software architecture.
- AOSD /AOP (Aspect Oriented Programming) provides process & Methodology approach for defining , specifying , designing & Constructing aspects.

UNIFIED PROCESS [UP] MODEL



UNIFIED PROCESS [UP] MODEL (1)

- Also Known as “Rational Unified Process [RUP]”

- UP Phases are Similar to “Generic Framework Activities”

- There are 5 UP Phases :

1. Inception [Communication + Planning]
2. Elaboration [Planning + Modeling]
3. Construction [Construction]
4. Transition [Construction + Deployment]
5. Production [-]

UNIFIED PROCESS [UP] MODEL (2)

Sr. No.	Phase	Activities Involved	Description
1	Inception	Communication + Planning	➢Identification of Business Requirements ➢Proposal of rough architecture for system. ➢Creation of Plan for Increment + Iterative ➢Preliminary Use-Case.
2	Elaboration	Planning + Modeling	➢Refines & Expands Preliminary Use Cases ➢5 Models Generation : Use-Case , Analysis , Design , Implementation , Deployment ➢Baselined Architecture does not provide all features & functions required for system.
3	Construction	Construction	➢Develops the software component that makes each use case operational. ➢Analysis & Design Phase are completed to reflect the final version of the increment. ➢Use case Derives Sets of User Acceptance tests.
4	Transition	Construction + Deployment	➢Software is given to user for beta testing. ➢Creation of documentation like user manuals.
5	Production	-	➢Ongoing use of the software is monitored. ➢Defect Reports & Request for changes submitted & evaluated.

PSP v/s TSP

Sr. No.	Parameters	PSP [Personal S/W Process]	TSP [Team S/W Process]
1	Best Fitted For :	Individual	Teams
2	Framewor k Activities :	1) Planning 2) High Level Design 3) High Level Design Review 4) Development 5) Postmortem	1) Launch High Level Design 2) Implementation 3) Integration 4) Test 5) Postmortem
3	Definition :	It is SEI (Software Engineering Institute) technology that brings disciplined to the software development habits to an Individual software Engineer .	It is Complementary SEI technology that enables teams to develop software products more effectively.
4	Helps In	✓Improving Product Quality ✓Increase Cost Predictability ✓Increase Schedule Predictability ✓Reducing Development Cycle Time.	✓It helps team of engineers how to produce quality products for planned costs and on aggressive schedules. ✓PSP is like applying Six Sigma to Software Development.
5	Provide Improvemen	Individual	Software Teams.

PART 1 : INTRODUCTION

7. INTRODUCTION TO CLEANROOM ENGINEERING

- It is a process of Developing software using Formal Methods instead of classic analysis , design , code and testing cycle.

- This approach emphasizes on writing the code correctly first time & verifying its correctness before testing.

- Objective : "To Build Most reliable software"

- This Method was originally developed by **Harlan Mills** at IBM.

- "Cleanroom " Term used in Electronic Industry wherein the introduction of defects were avoided during fabrication of IC's.

7. INTRODUCTION TO CLEANROOM ENGINEERING (1)

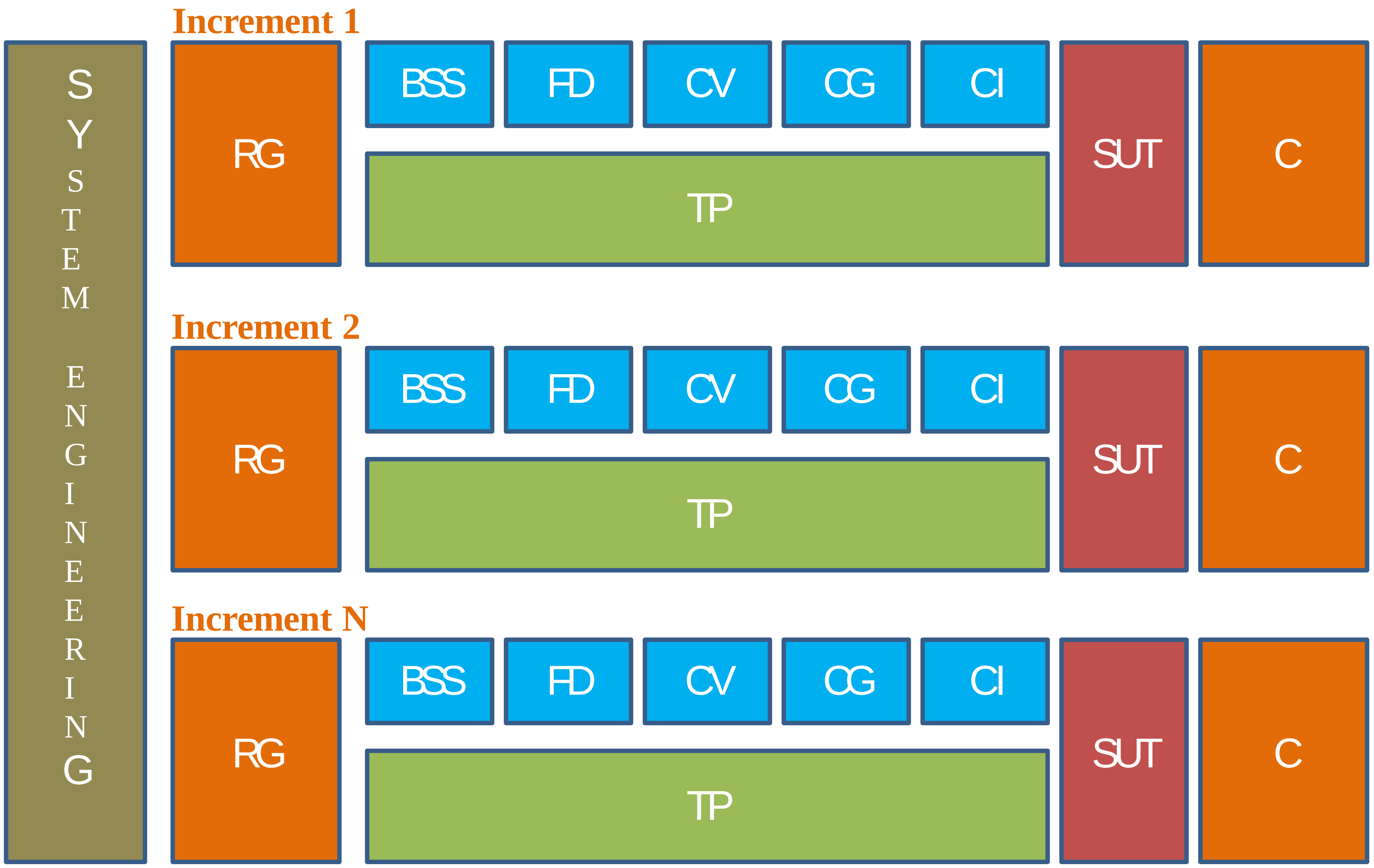
- Specialized Version of "Incremental Software Model"

- The Philosophy "To Avoid Dependence on costly defect removal processes by writing code increments right the first time and verifying their correctness before testing"

- It's a Process Model incorporates the statistical quality certification of code increments as they accumulate into system.

- The Increments are tested Individually as well as these are Retested when added with the new increments.

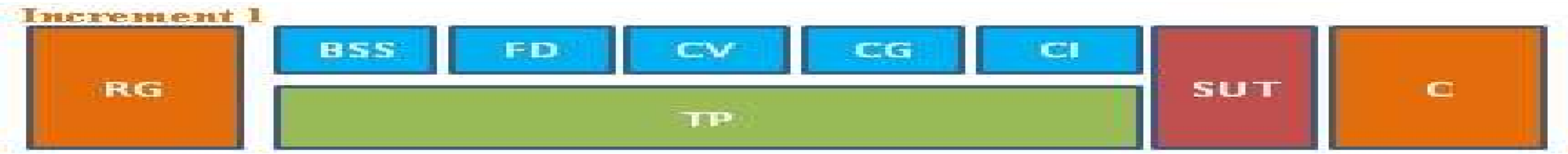
7. INTRODUCTION TO CLEANROOM ENGINEERING (2)



CLEANROOM PROCESS MODEL

7. INTRODUCTION TO CLEANROOM ENGINEERING (3)

TASKS	EXPLAINATION
Requirement Gathering [RG]	More Details of Customer-Level Requirements are Developed.
Box Structure Specification [BSS]	Used to describe Functional Specification.
Formal Design [FD]	Specifications [Black-Box] are iteratively refined to become analogous to architectural & component-level designs (clear box)
Corrective Verification [CV]	From High Level to Detailed Leveled Rigorous Verification.
Code Generation [CG]	Generation of code based on Formal Design.
Code Inception [CI]	To Check Syntactic Correctness of code.
Test Planning [TP]	The Projected Usage was Analyzed and Suite of test cases are planned.
Statistical Use Testing [SUT]	Executes series of test cases .
Certification [C]	Once Verification..Inception and usage testing completed , increment is certified as ready for integration.



INTRODUCTION TO CLEANROOM ENGINEERING (4)

- ADVANTAGES :

✓ Zero Defect Software can be developed using Cleanroom Approach

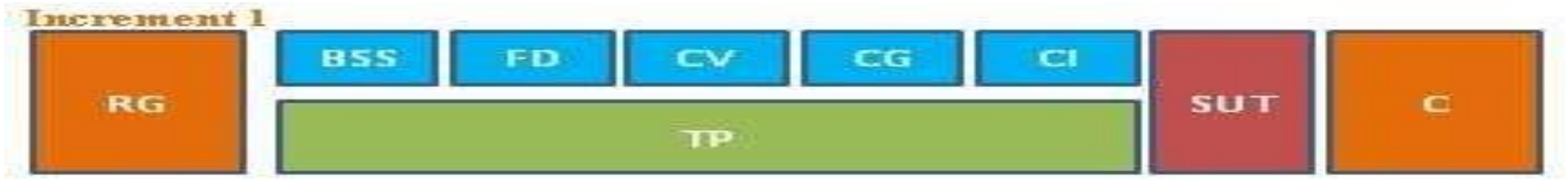
✓ Defects are discovered during development stage. Hence the cost and time of testing is reduced.

- DISADVANTAGES :

× Requires Skilled & Committed Staff.

× Formal Mathematical Proofs Required to check Correctness of Program. Developing Such Proofs w.r.t. Specification is Very Expensive.

× Requires Knowledge of mathematics and formal semantic of the program.



TYPES OF TESTING : VERIFICATION

- Focus on "Find Defects As Early As Possible.
- It Makes Sure that Software Application is getting Developed in the correct way with respect to the Requirement.
- Are we **Doing** the job Right?
- It Represents Left Arm of V-Model.
- It is also Known as "Static Testing" i.e. Before Execution.

PROCESS OF VERIFICATION :

1. Systematically Read the Contents of a SRS.
2. Find Issues/Defects
3. Get Them Solved
4. Static Testing as Software Application is not executed .

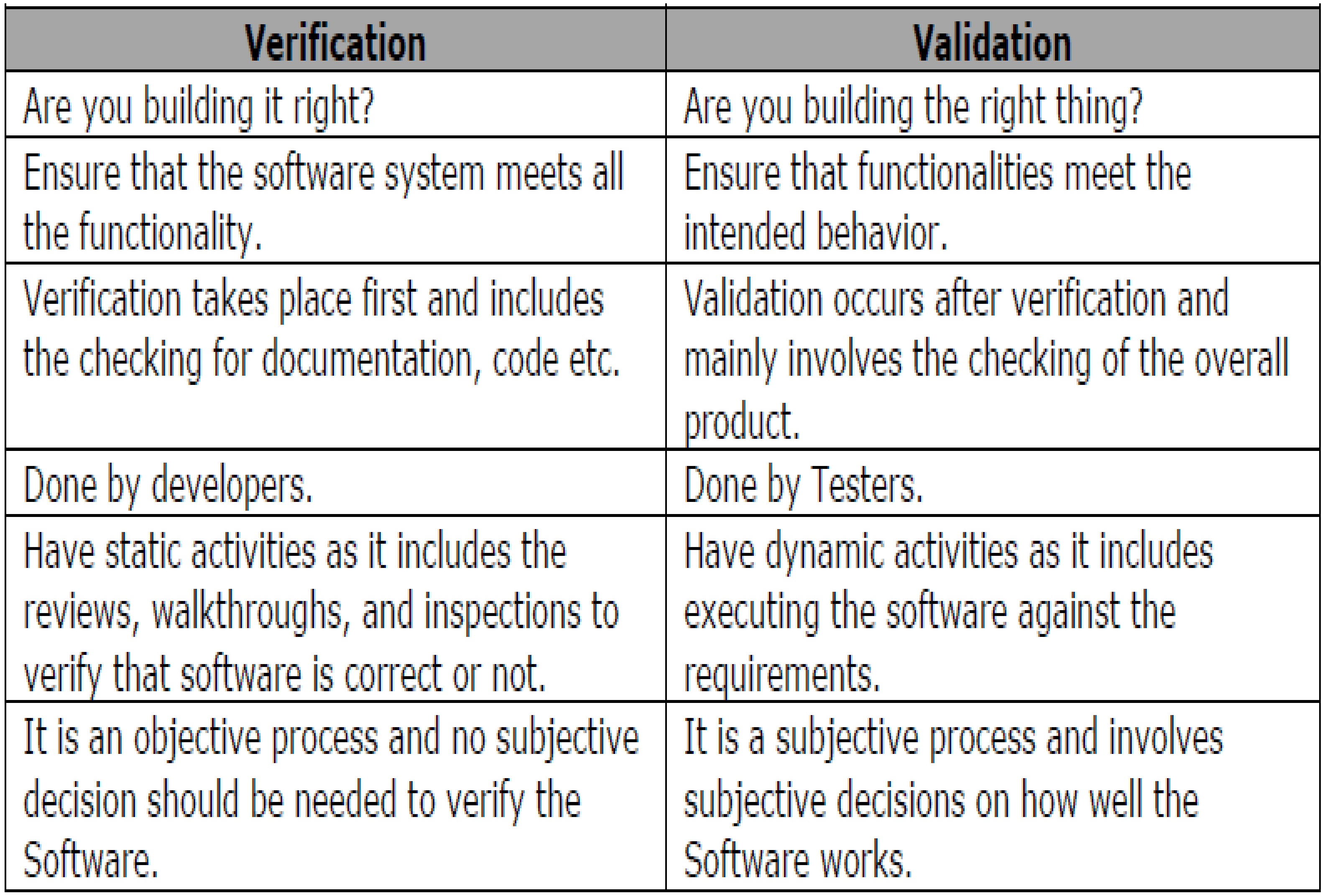
TYPES OF TESTING : VALIDATION

- A Disciplined Approach to evaluate whether the final, developed application fulfills its specific intended purpose.
- Are we **Doing** the Right job?
- It Represents Right Arm of V-Model.
- It is also Known as "Dynamic Testing" i.e. After Execution.

LEVELS OF TESTING :

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

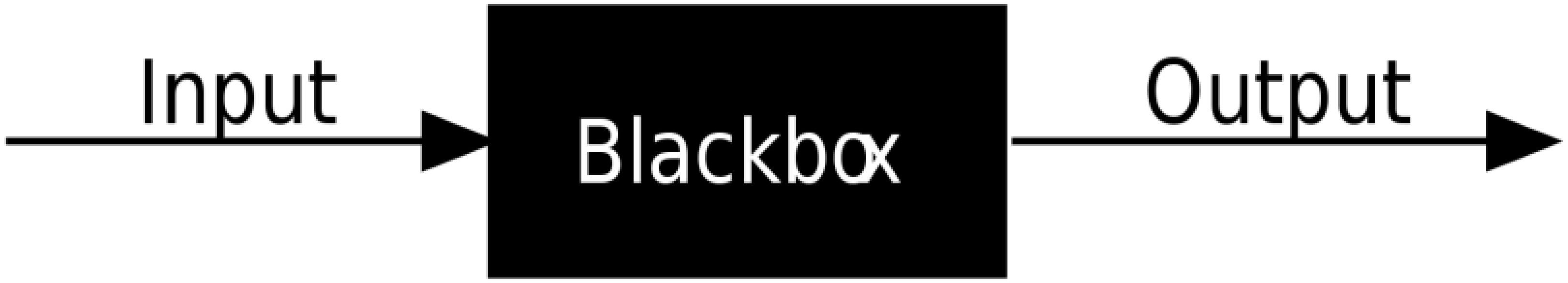
VERIFICATION Vs. VALIDATION



Verification	Validation
Are you building it right?	Are you building the right thing?
Ensure that the software system meets all the functionality.	Ensure that functionalities meet the intended behavior.
Verification takes place first and includes the checking for documentation, code etc.	Validation occurs after verification and mainly involves the checking of the overall product.
Done by developers.	Done by Testers.
Have static activities as it includes the reviews, walkthroughs, and inspections to verify that software is correct or not.	Have dynamic activities as it includes executing the software against the requirements.
It is an objective process and no subjective decision should be needed to verify the Software.	It is a subjective process and involves subjective decisions on how well the Software works.

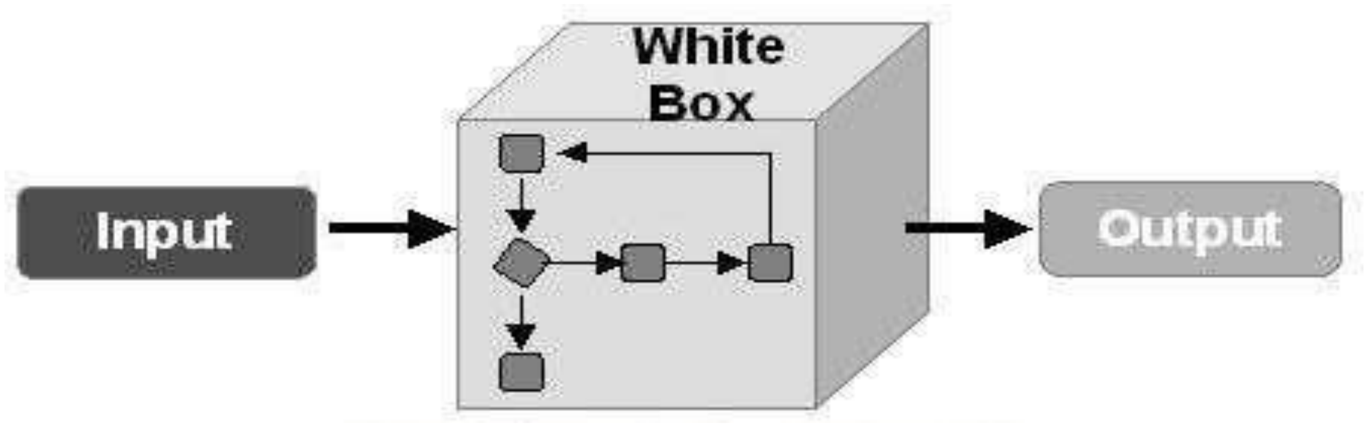
METHOD OF TESTING :BLACK BOX TESTING

- Application is treated as a black box where internal mechanism is not known.
- Do not look inside the program code.
- It is Specification-Based Testing.
- It is also known as Functional Testing.



METHOD OF TESTING : WHITE BOX TESTING

- Application is treated as a white box where internal mechanism is known.
- Test the Program Code i.e. Internal Structure.
- It is Structural-Based Testing.
- It is also known as Glass Box Testing.



Unit Testing

- Unit : The Smallest Piece of software that can be tested in Isolation to verify its behavior.

- It is usually done by Developer.

- Either Done Unit Testing Manually or using tools like Junit , Nunit.

- Purpose : Ensures that the code meets the requirements.

INTEGRATION Testing

- Purpose : To confirm that the individually tested units can work together to deliver the intended functionality.

- It is done by Developer & Tester.

- Types of Integration Testing : 1. Top-Down 2. Bottom-Up 3. Critical Part First 4. Big –Bang.

SYSTEM Testing

- It is conducted on a Complete , integrated system to evaluate system compliance with its specified features.
- It is Always done by Tester.
- It falls within a scope of Black-Box Testing.

PART 2 : SOFTWARE QUALITY ASSURANCE

8.2.1 : Quality Assurance

Goal :

To Ensure the Management of Data which is Important for Product Quality.

SQA Activities :

1. Prepare an SQA Plan for a Project.
2. Participates in the Development of the project's software process description.
3. Reviews Software Engineering activities to verify compliance with the defined software process.
4. Audits Designated Software Work Product to verify compliance with the defined software process.
5. Ensure Deviation in software work . These work products are documented & handled according to documented procedures.
6. Records any noncompliance & reports to senior management.

PART 2 : SOFTWARE QUALITY ASSURANCE

8.2.2 : SQA PLANS

- ❑ It Provides Roadmap for instituting SQA.
- ❑ It Serves as a Template for SQA Activities.

SQA 7 Sections/Elements:

1. Management Section.
2. Documentation Section
3. Standards, Practices and Conventions Section.
4. Review & Audits Section.
5. Test Section.
6. Problem Reporting & Corrective Action Section.
7. Other.

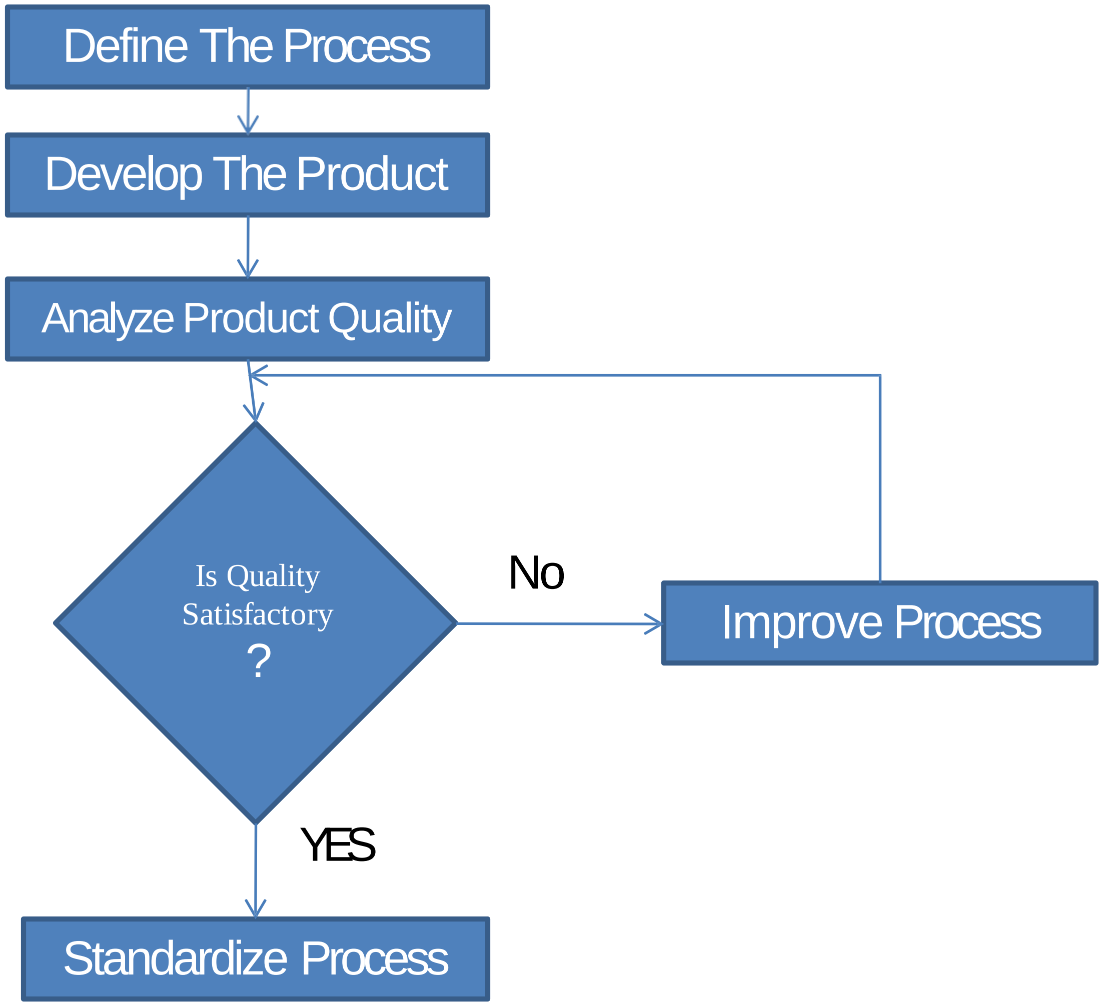
PART 2 : SOFTWARE QUALITY ASSURANCE

8.2.2 : SQA TEMPLATE [IEEE STANDARD]

1. Purpose & Scope of a Plan
2. Description of work product
3. Applicable Standards
4. SQA Activities
5. Tools and methods/standards used
6. SCM procedures for managing change.
7. Methods for maintaining SQA related records
8. Organizational roles & responsibilities.

PART 2 : SOFTWARE QUALITY ASSURANCE

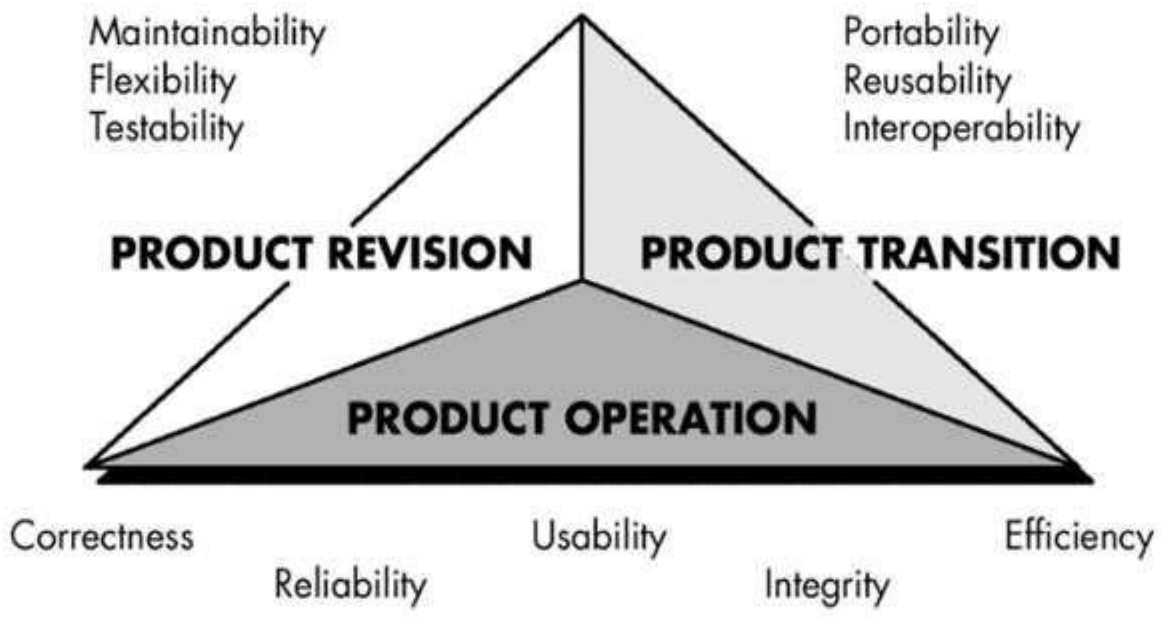
8.3 : SOFTWARE QUALITY FRAMEWORK



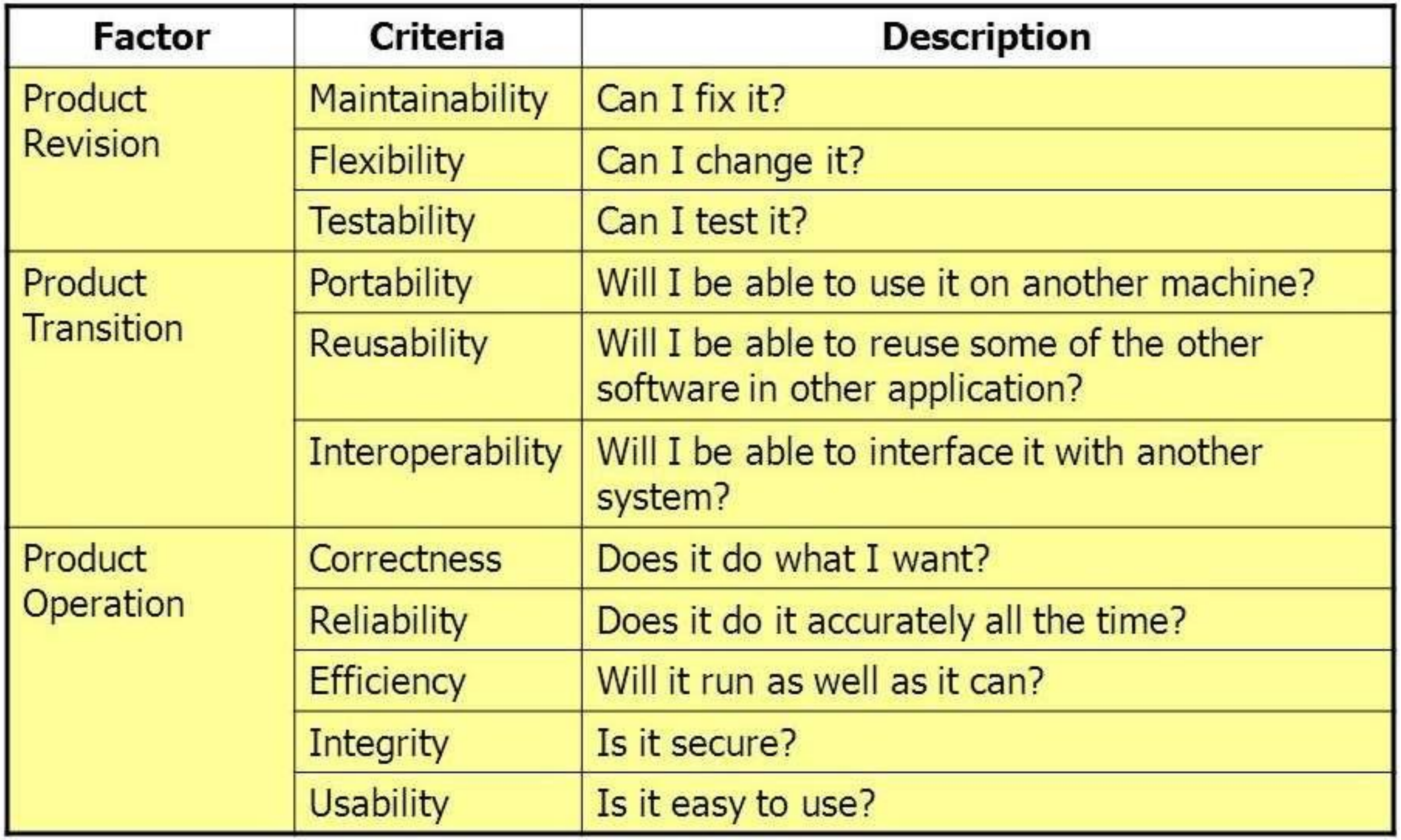
8.3.1 : McCall's Quality Factors

- There are 2 Classes of quality Factors that affects software Quality
  - Directly Measured.
  - Indirectly Measured.
- McCall , Richards , Walters have proposed Software Quality factors into 3 Categories .

McCall’s Triangular Model



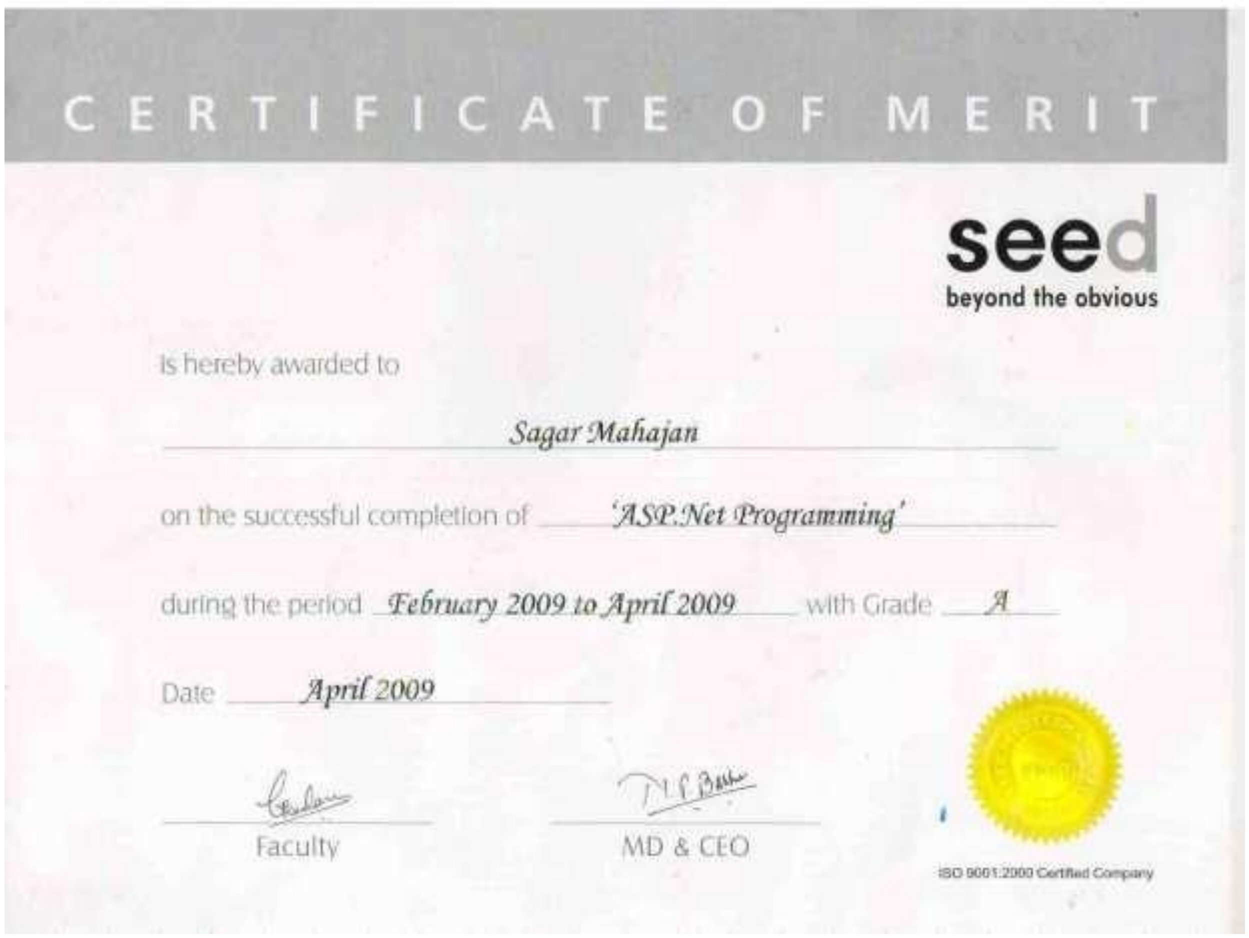
8.3.1 : McCall's Quality Factors (1)



Factor	Criteria	Description
Product Revision	Maintainability	Can I fix it?
	Flexibility	Can I change it?
	Testability	Can I test it?
Product Transition	Portability	Will I be able to use it on another machine?
	Reusability	Will I be able to reuse some of the other software in other application?
	Interoperability	Will I be able to interface it with another system?
Product Operation	Correctness	Does it do what I want?
	Reliability	Does it do it accurately all the time?
	Efficiency	Will it run as well as it can?
	Integrity	Is it secure?
	Usability	Is it easy to use?

PART 2 : SOFTWARE QUALITY ASSURANCE

8.4 : ISO-9000 MODELS



C E R T I F I C A T E O F M E R I T

seed
beyond the obvious

is hereby awarded to

Sagar Mahajan

on the successful completion of *'ASP.Net Programming'*

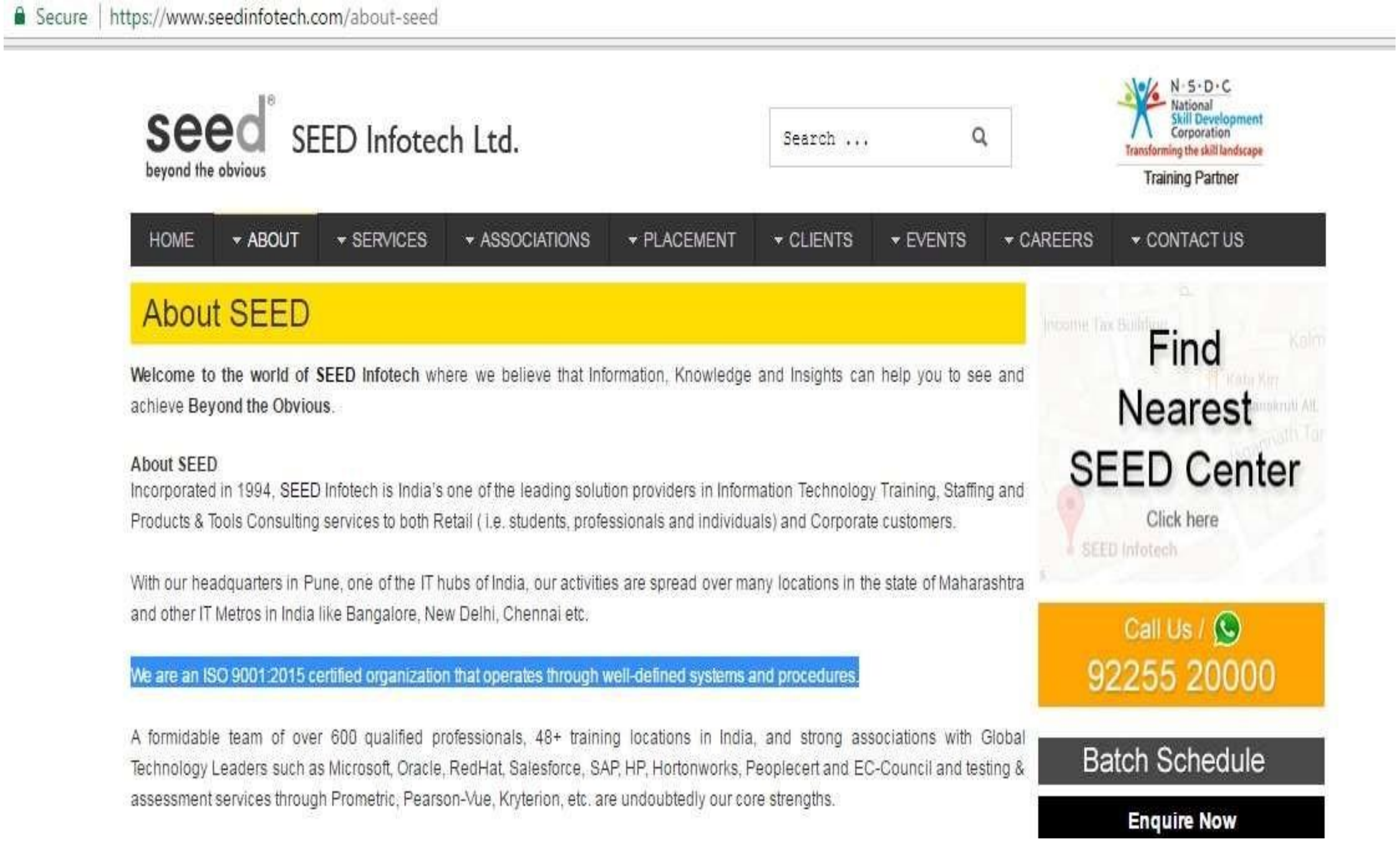
during the period *February 2009 to April 2009* with Grade *A*

Date *April 2009*

[illegible] [illegible]

Faculty MD & CEO

ISO 9001:2000 Certified Company



About SEED

Welcome to the world of SEED Infotech where we believe that Information, Knowledge and Insights can help you to see and achieve **Beyond the Obvious**.

About SEED
Incorporated in 1994, SEED Infotech is India's one of the leading solution providers in Information Technology Training, Staffing and Products & Tools Consulting services to both Retail (i.e. students, professionals and individuals) and Corporate customers.

With our headquarters in Pune, one of the IT hubs of India, our activities are spread over many locations in the state of Maharashtra and other IT Metros in India like Bangalore, New Delhi, Chennai etc.

We are an ISO 9001:2015 certified organization that operates through well-defined systems and procedures.

A formidable team of over 600 qualified professionals, 48+ training locations in India, and strong associations with Global Technology Leaders such as Microsoft, Oracle, RedHat, Salesforce, SAP, HP, Hortonworks, Peoplecert and EC-Council and testing & assessment services through Prometric, Pearson-Vue, Kryterion, etc. are undoubtedly our core strengths.

8.4 : ISO-9000 MODELS

➢The **ISO 9000** family of quality management systems standards is designed to help organizations ensure that they meet the needs of customers and other stakeholders while meeting statutory and regulatory requirements related to a product or program.

➢The ISO 9000 series are based on seven Quality Management Principles (QMP).

QMP 1 – **Customer focus**
QMP 2 **--** **Leadership**
QMP 3 – **Engagement of people**
QMP 4 – **Process approach**
QMP 5 – **Improvement**
QMP 6 – **Evidence-based decision making**
QMP 7 – **Relationship management**

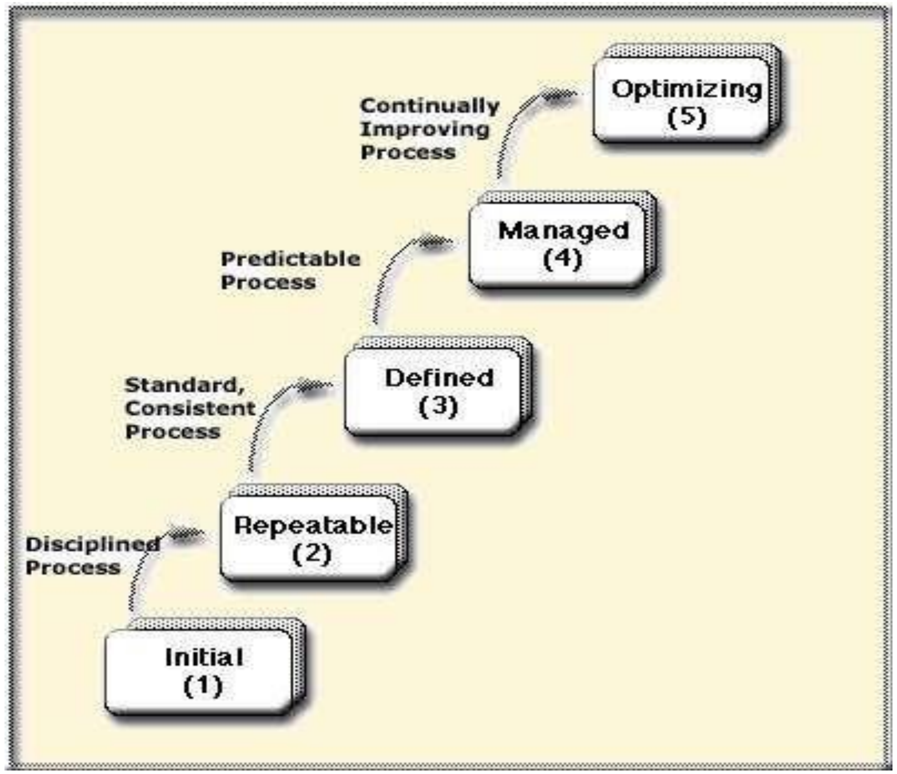
8.5 : CMM MODELS

➢Used For Improving Software Project.

➢Used In Assessing how well an Organization's processes allow to complete & Manage new Software Projects

➢Process Maturity Levels :

1. Initial
2. Repeatable
3. Defined
4. Managed
5. Optimizing

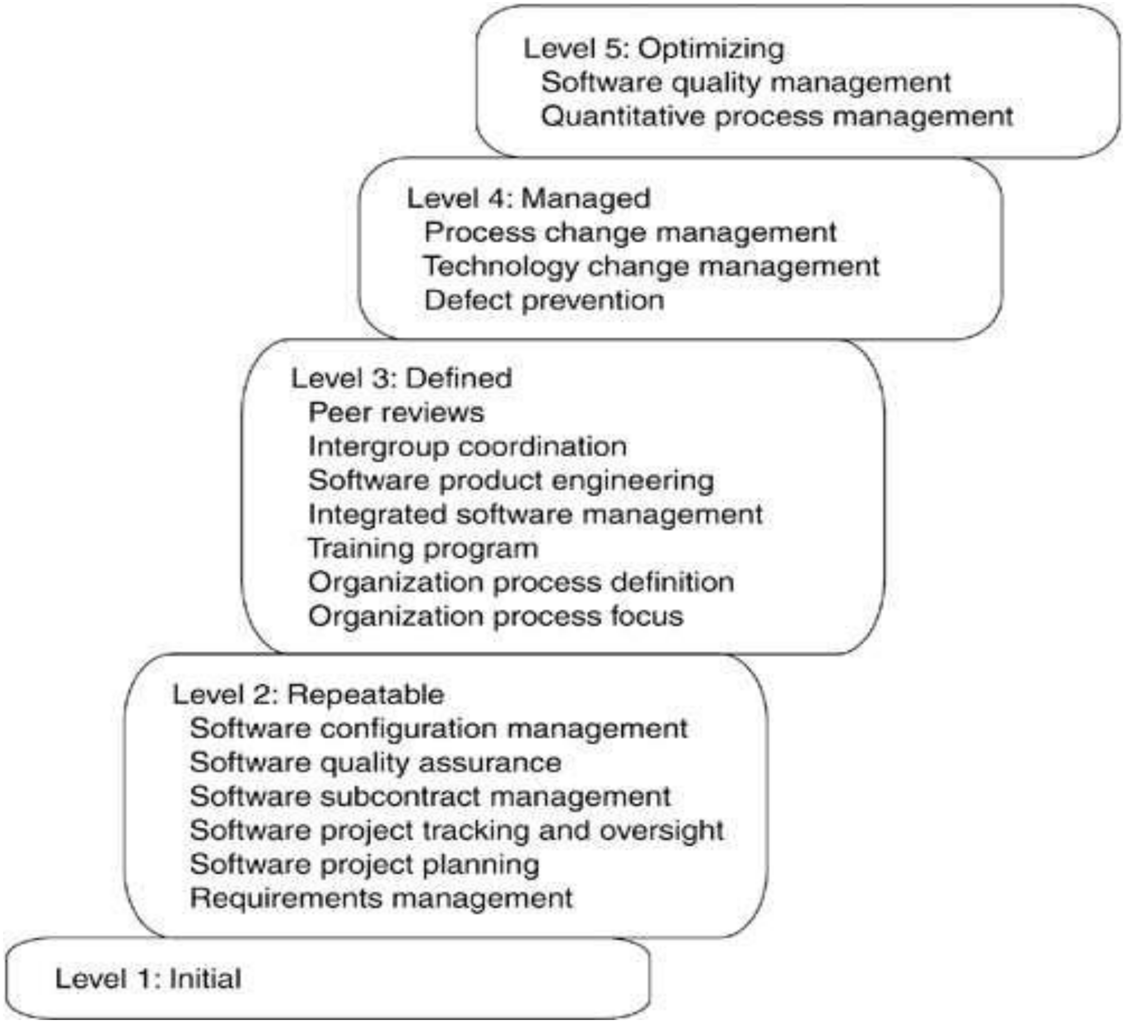


PART 2 : SOFTWARE QUALITY ASSURANCE

8.5 : CMM MODELS

Maturity Levels	Description
1- (Initial)	Few Processes are defined. Individual Efforts Taken.
2 – (Repeatable)	Depending upon Earlier Success of projects with similar applications , necessary Project Discipline can be Repeated.
3 – (Defined)	The Process is standardized , documented & followed.
4 – (Managed)	Both Software Process & Product are quantitatively understood & controlled.
5 – (Optimizing)	Establish Mechanisms to Plan and Implement Change. Innovative Ideas & Technologies can be tested.

8.5 : CMM MODELS


Level 5: Optimizing
Software quality management
Quantitative process management
Level 4: Managed
Process change management
Technology change management
Defect prevention
Level 3: Defined
Peer reviews
Intergroup coordination
Software product engineering
Integrated software management
Training program
Organization process definition
Organization process focus
Level 2: Repeatable
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management
Level 1: Initial