Session – 1

SOFTWARE

K L University

u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

INTRODUCTION

What is Software

Software is:

instructions (computer programs) that when executed provide desired features, function, and performance; .

Software is a set of instructions, data or programs used to operate computers and execute specific tasks

Characteristics of S/W

1. *Software is developed or engineered, it is not manufactured in the classical sense.*

- *Software is logical and hardware is physical entity. Both activities require the construction of a "product," but the approaches are different*

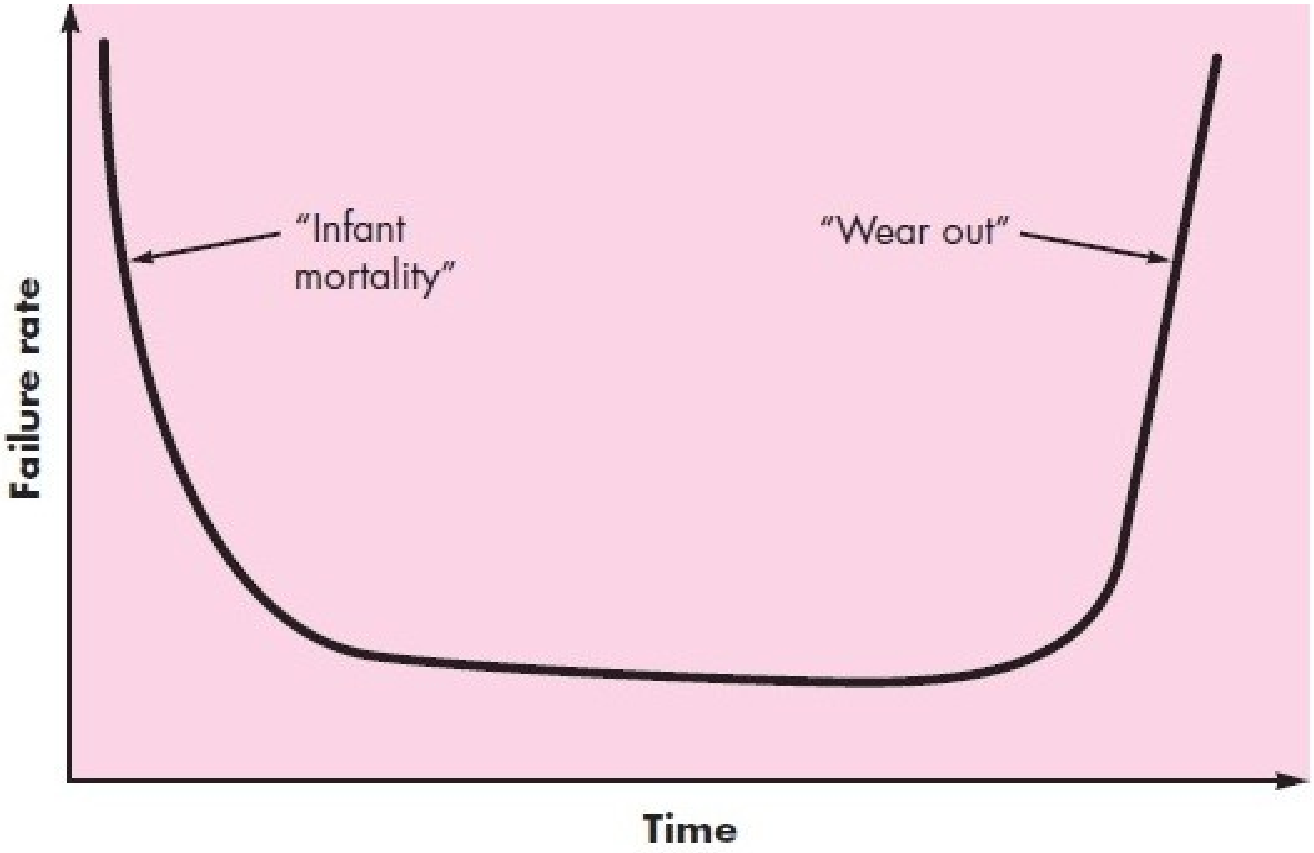
2.*Software doesn′t ″wear out.″* but it *deteriorates*

Hardware effected by heat ,dust etc., ,software does not effected by environmental conditions

3.*Although the industry is moving toward component-based construction, most software continues to be custom-built.*

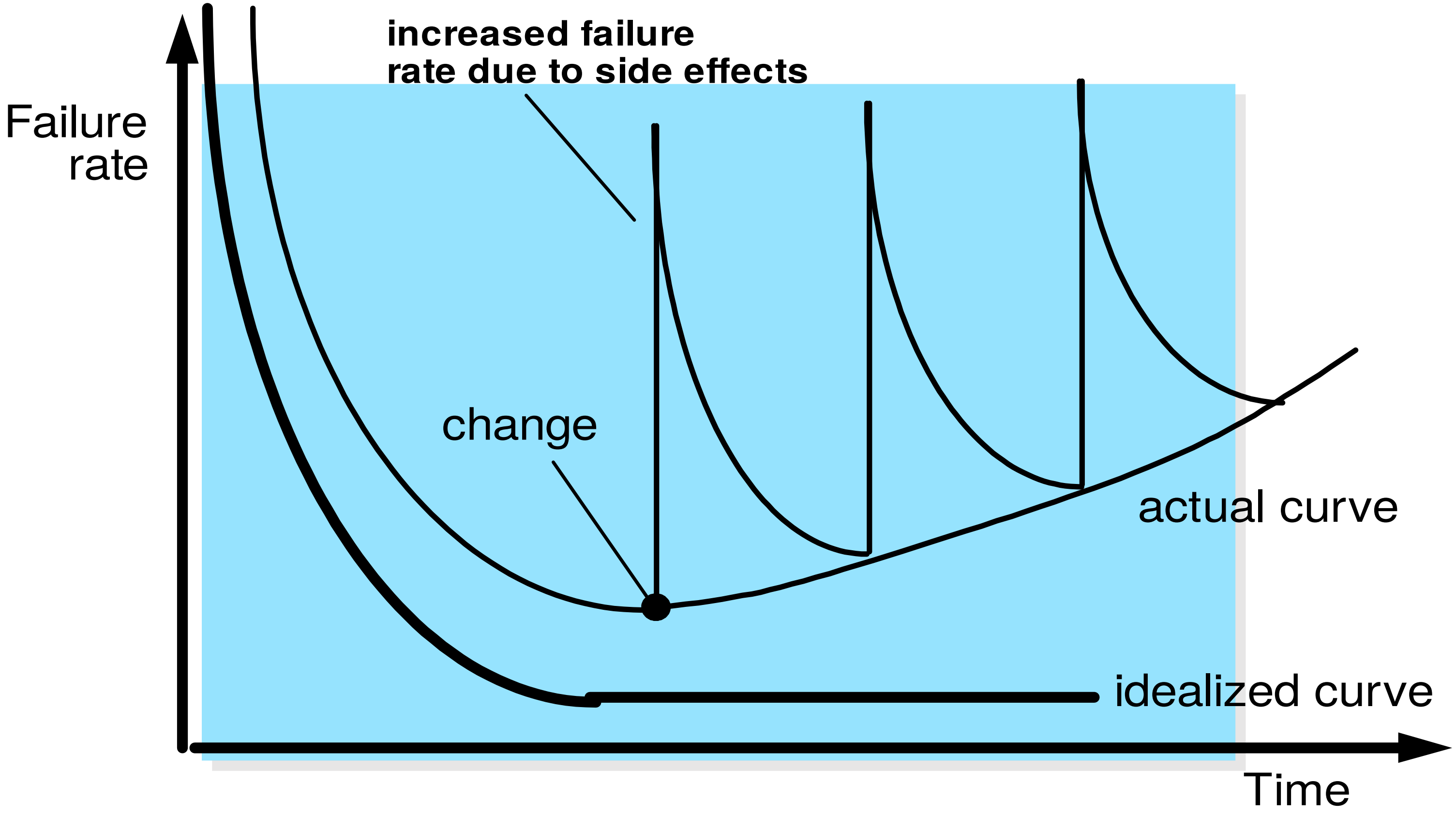
A software component should be designed and implemented so that it can be reused in many different programs

Failure curves for hardware



Failure curves for software

Wear vs. Deterioration



Nature of Software

Software is a **Product**

- *Transforms* information- produces, manages, acquires, modifies, displays or transmits information
- Delivers computing potential of hardware and networks

Software is a **vehicle** for delivering a product

- Controls other programs (operating system)
- Effects communications (networking software)
- Helps build other software (software tools & environments)

Software Application Domains

K L University

u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

Software Application Domains- 7

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software

- **System software—a collection of programs written to service other programs.**

Some system software (e.g., compilers)

- **Application software—stand-alone programs that solve a specific business** need.

(e.g., Ticket booking from irctc,Passport application ,real-time manufacturing process control).

- **Engineering/scientific software—has been characterized by “number** crunching” algorithms

(e.g., Computer-aided design, system simulation,

and other interactive applications)

- **Embedded software—resides within a product or system and is used to** implement and control features

e.g., key pad control for a microwave oven)

- **Product-line software—designed to provide a specific capability for use by** many different customers. (e.g., word processing, spreadsheets
- **Web applications—called “WebApps,” this network-centric software** (e.g., True caller.. etc)
- **Artificial intelligence software—makes use of nonnumerical algorithms to** solve complex problems (e.g., game playing etc)

Software - New Challenges

- Open world computing—pervasive, distributed computing
- Ubiquitous computing—wireless networks
- Net sourcing—the Web as a computing engine-Need target marketing
- Open source—”free” source code open to the computing community

Legacy Software

- older programs—often referred to as *legacy software*
- They were developed decades ago and have been continually modified to meet changes in business requirements
- one additional characteristic that is present in legacy software—*poor quality*

- a legacy system must be reengineered

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.

Unique Nature of WebApps

- **Network intensiveness.** A WebApp resides on a network and must serve the needs of a diverse community of clients.
- **Concurrency.** A large number of users may access the WebApp at one time.
- **Unpredictable load.** The number of users of the WebApp may vary by orders of magnitude from day to day.
- **Performance.** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.
- **Availability.** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a “24/7/365” basis.

Unique Nature of WebApps

- **Data driven.** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end-user.
- **Content sensitive.** The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.
- **Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously.
- **Immediacy.** Although *immediacy*—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks.
- **Security.** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application..

End of session - 1

Questions

1. Define Software.
2. What is Bathtub curve.
3. What are Characteristics of S/W.
4. What are different Software Application Domains.
5. Advantages of Software

Session – 2
Software Engineering

K L University

u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

Software Engineering

- Some realities:
  - *a concerted effort should be made to understand the problem before a software solution is developed*
  - *design becomes a pivotal activity*
  - *software should exhibit high quality*
  - *software should be maintainable*
- The seminal definition:
  - *[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.*

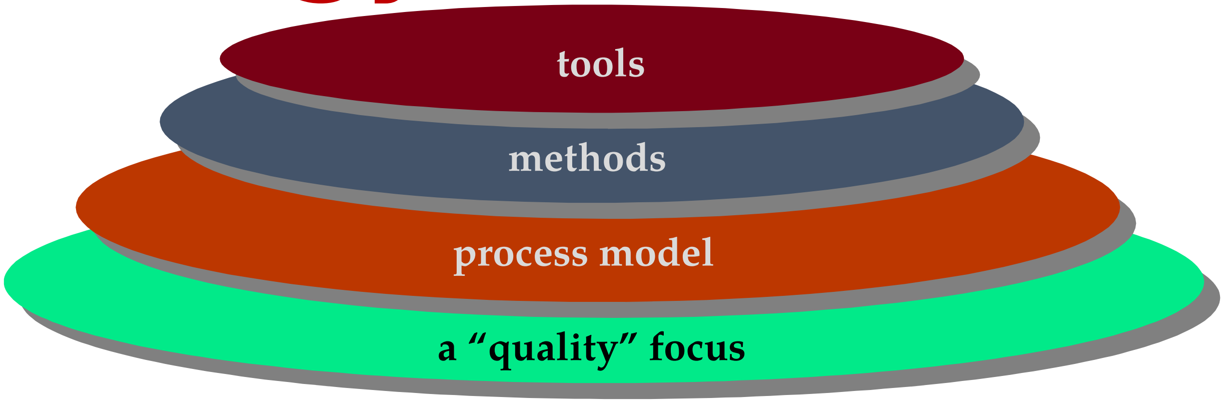
Software Engineering

- The IEEE definition:
  - *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.

A Layered Technology



Software Engineering

A Layered Technology

Quality: Organization is commitment to quality

Process: Manages the control of software projects
Ensures quality, establishes milestones, manages changes

Methods: Provide technical ways for building software i.e;
Communication, requirement analysis, design modeling, program construction , testing and support

Tools: Provide automated or semi automated support for the process & methods

The Software Process

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

- ***Activity-***Strives to achieve a broad objective

  (eg: communication with stakeholders)
- ***Actions-*** encompasses a set of tasks that produce a major work

  product ***(eg:*** architectural design***)***
- ***Task-***focusses on small, but well-defined objective that

  produces a tangible outcome ***(eg:*** conduct a unit test***)***

Process framework Activities

1. **Communication**
2. **Planning**
3. **Modeling**
   - Analysis of requirements
   - Design
4. **Construction**
   - Code generation
   - Testing
5. **Deployment**
   - Delivery
   - Feedback

Process framework Activities(cont.)

- **Communication:** communicate with customer to understand objectives and gather requirements
- **Planning**:
- **Modeling:**
- **Construction**:
- **Deployment:**

Process framework Activities(cont.)

- **Communication:**
- **Planning**: creates a “map” that defines the work by describing tasks, risks and resources, work products and work schedule.
- **Modeling:**
- **Construction**:
- **Deployment:**

Process framework Activities(cont.)

- **Communication:**
- **Planning**:
- **Modeling:** Create a “sketch”, what it looks like architecturally, how the essential parts fit together and other characteristics.
- **Construction**:
- **Deployment:**

Process framework Activities(cont.)

- **Communication:**
- **Planning**:
- **Modeling:**
- **Construction:** code generation and the testing.
- **Deployment:**

Process framework Activities(cont.)

- **Communication:**
- **Planning**:
- **Modeling:**
- **Construction**:
- **Deployment:** Delivered to the customer who evaluates the products & provides feedback based on the evaluation.

Process framework Activities(cont.)

- These five framework activities can be used to all software development, regardless of the application domain, size of the project, complexity of the efforts etc.
  - though the details will be different in each case.

- For many software projects, these framework activities are applied **iteratively** as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

Umbrella Activities

- Complete the five process framework activities and help team manage and control progress, quality, change, and risk.

Software project tracking and control:

- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**: assess progress against the plan and take actions to maintain the schedule.
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**: assesses risks that may affect the outcome and quality.
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**: defines and conduct activities to ensure quality.
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**: assesses work products to uncover and remove errors before going to the next activity.
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**: define and collects process, project, and product measures to ensure stakeholder's needs are met.
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**: manage the effects of change throughout the software process.
- **Reusability management**:
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**: defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- **Work product preparation and production:**

Umbrella Activities

- **Software project tracking & control**:
- **Risk management**:
- **Software quality assurance**:
- **Technical reviews**:
- **Measurement**:
- **Software configuration management**:
- **Reusability management**:
- **Work product preparation and production:** create work products such as models, documents, logs, forms and lists.

Software Engineering Practice

Software Engineering Practice

The Essence of Practice

- How does the practice of software engineering fit in the process activities mentioned above? Namely,
  - **communication,**
  - **planning,**
  - **modeling,**
  - **construction**
  - **deployment**.
- The essence of problem solving is outlined in 4 points:
  1. *Understand the problem* (communication and analysis).
  2. *Plan a solution* (modeling and software design).
  3. *Carry out the plan* (code generation).
  4. *Examine the result for accuracy* (testing and quality assurance).

Understand the Problem

- ***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?

Plan the Solution

- ***Have you seen similar problems before?*** Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- ***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?

Carry Out the Plan

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

Examine the Result

- ***Is it possible to test each component part of the solution?*** Has a reasonable testing strategy been implemented?
- ***Does the solution produce results that conform to the data, functions, and features that are required?*** Has the software been validated against all stakeholder requirements?

Hooker’s General Principles

1: *The Reason It All Exists*

2: *KISS (Keep It Simple, Stupid!)*

3: *Maintain the Vision*

4: *What You Produce, Others Will Consume*

5: *Be Open to the Future*

6: *Plan Ahead for Reuse*

7*: Think!*

Software Myths

1.Management Myths

- "We already have a book of standards and procedures for building software. It does provide my people with everything they need to know …"
- "If my project is behind the schedule, I always can add more programmers to it and catch up …"

  (a.k.a. "**The Mongolian Horde concept**")
- "If I decide to outsource the software project to a third party, I can just relax: Let them build it, and I will just pocket my profits …"

2. Customer Myths

- “A general statement of objectives is sufficient to begin writing programs - we can fill in the details later …”
- “Project requirements continually change but this change can easily be accommodated because software is flexible …”

3. Practitioner's Myths

- "Let's start coding ASAP, because once we write the program and get it to work, our job is done …"
- "Until I get the program running, I have no way of assessing its quality …"
- "The only deliverable work product for a successful project is the working program …"
- "Software engineering is baloney(nonsense.). It makes us create tons of paperwork, only to slow us down …"

End of Session - 2

Questions:

1. Define Software Engineering.

2. What is the need of Umbrella Activities.

3. What are different Layers of Software Engineering.

4. What are Process framework Activities.

5. What are different activities comes under Modeling face.

6. Need of Software quality assurance in software development process.

7. Define Myth ?

8. How many types of Myths available.

9. What are myths comes under Management Myths.

10. What are myths comes under Customer Myths.

11. What are myths comes under Practitioner’s Myths.

Session -3

A Generic Process Model

K L University

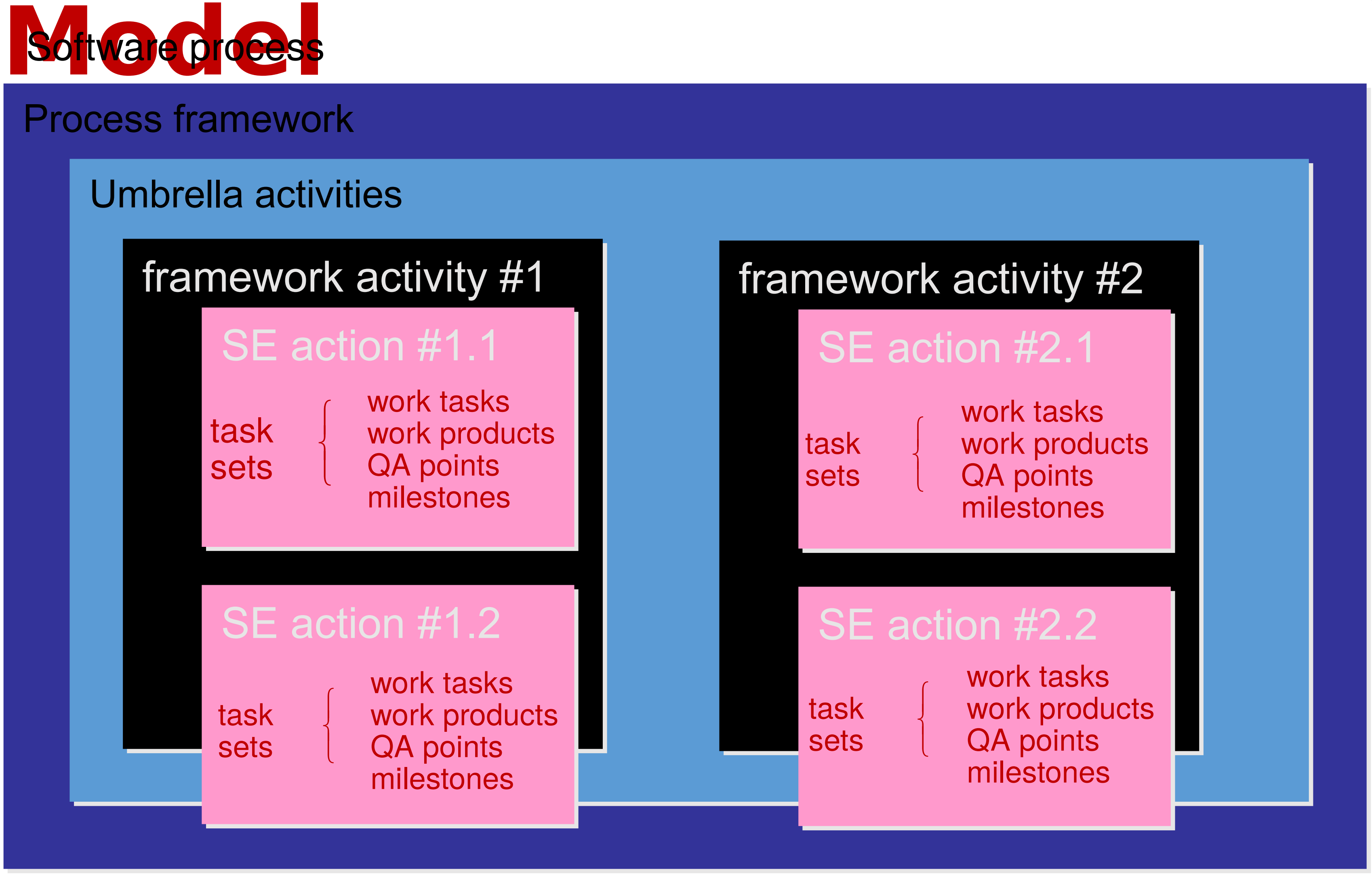
u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

- a process was defined as a collection of work activities, actions, and tasks that are performed when some work product is to be created

- each framework activity is populated by a set of software engineering actions.

- Each software engineering action is defined by a *task set that*
- *identifies the work* tasks that are to be completed,
- the work products that will be produced,
- the quality assurance points that will be required,
- and the milestones that will be used to indicate progress.

A Generic Process Model



Software process

Process framework

Umbrella activities

framework activity #1

SE action #1.1

task sets: work tasks, work products, QA points, milestones

SE action #1.2

task sets: work tasks, work products, QA points, milestones

framework activity #2

SE action #2.1

task sets: work tasks, work products, QA points, milestones

SE action #2.2

task sets: work tasks, work products, QA points, milestones

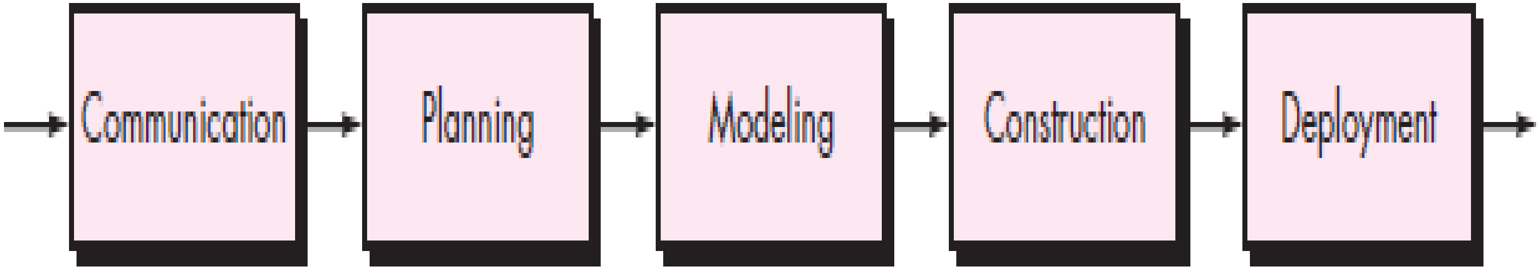
Process Flow: How the framework activities, actions, tasks occur within each framework activity

A *linear process flow executes each of the five framework activities in sequence*, beginning with communication and culminating with deployment

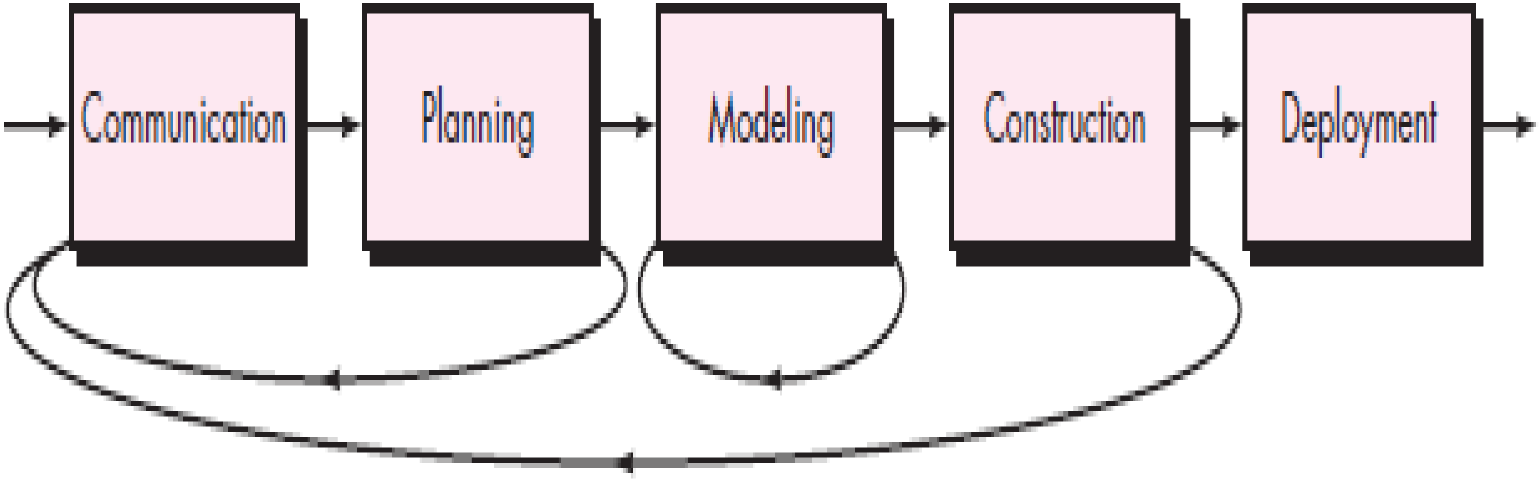
An *iterative process flow repeats one or more of the activities before proceeding to the* next.

An *evolutionary process flow executes the activities in a “circular”* manner. Each circuit through the five activities leads to a more complete version of the software

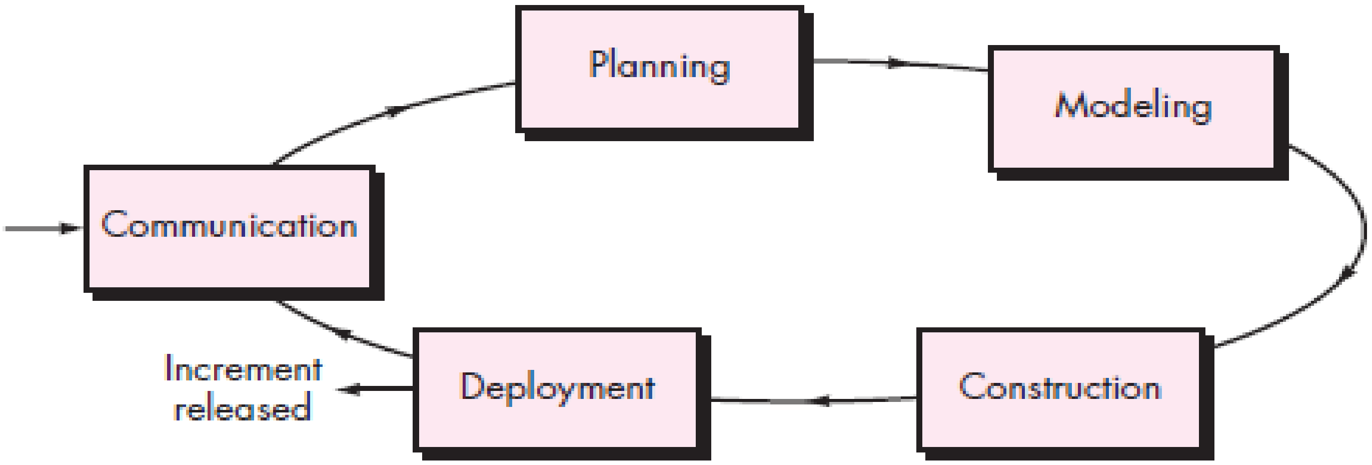
A *parallel process flow executes one or* more activities in parallel with other activities (e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software).



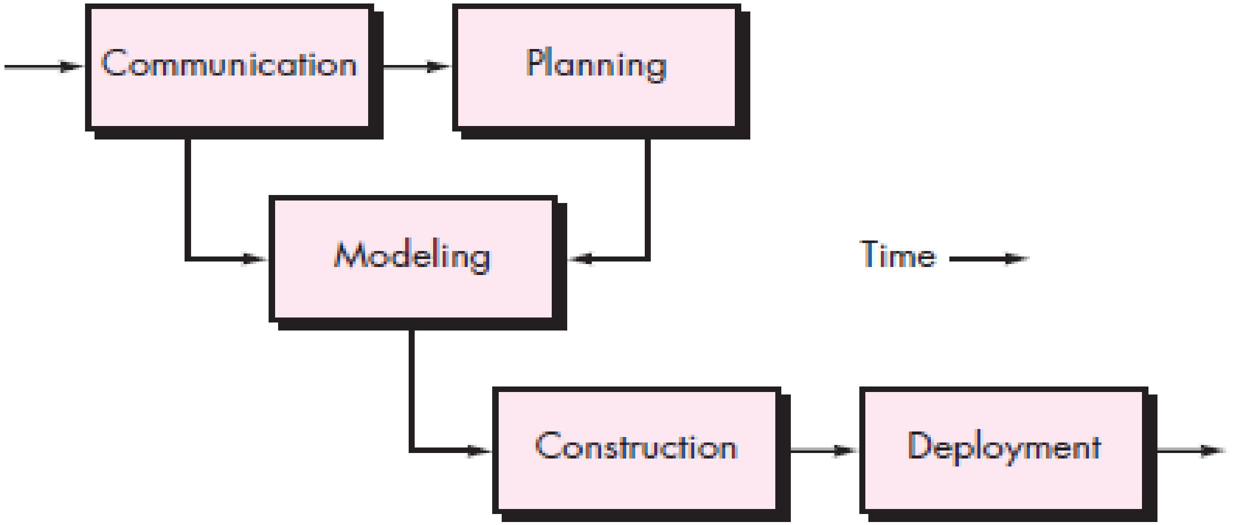
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow



(d) Parallel process flow

Prescriptive Process models

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Prescriptive Process models

- Waterfall Model.
- Incremental Process Model.
- Evolutionary Process Model.
- Concurrent model

Prescriptive Process models

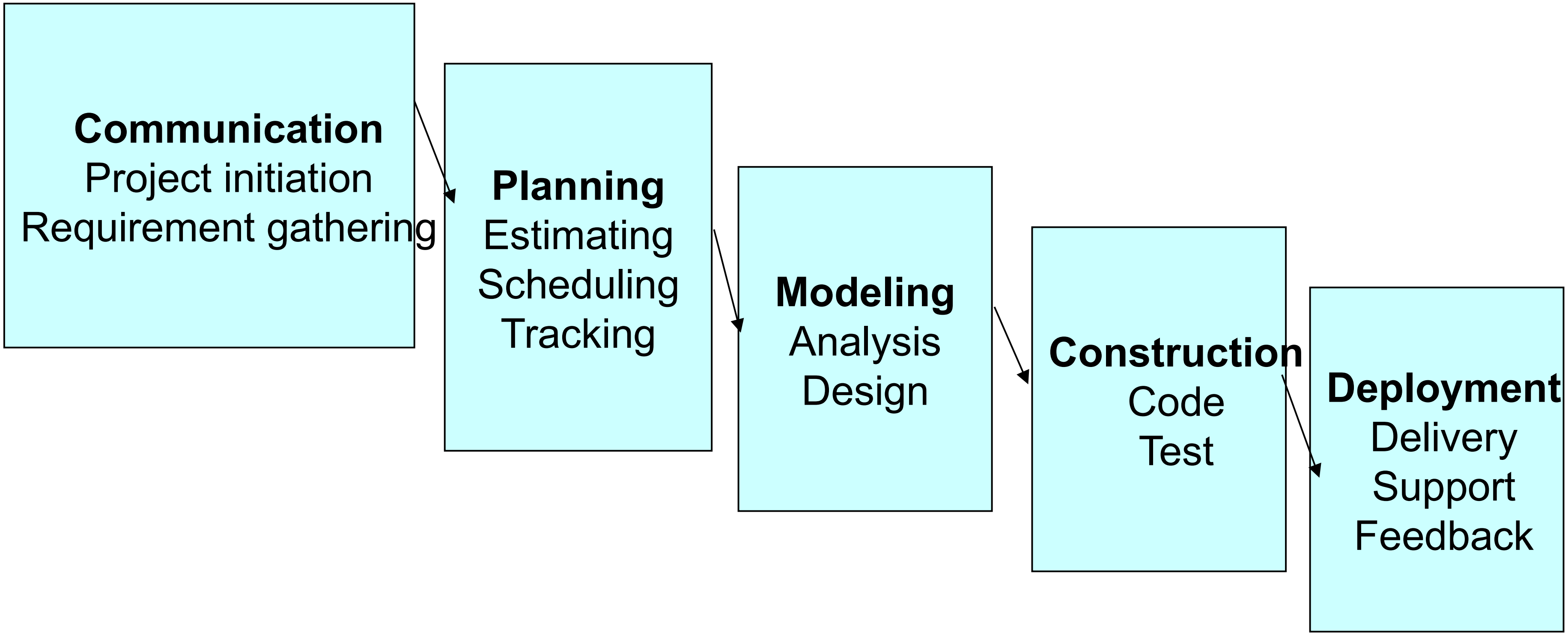
Waterfall Model

- Oldest software lifecycle model & best understood by upper management

- Used when requirements are well understood and risk is low

- Work flow is in a linear fashion (i.e., sequential)

- Used often with well-defined adaptations or enhancements to current software



Communication
Project initiation
Requirement gathering

Planning
Estimating
Scheduling
Tracking

Modeling
Analysis
Design

Construction
Code
Test

Deployment
Delivery
Support
Feedback

Waterfall Model

Useful

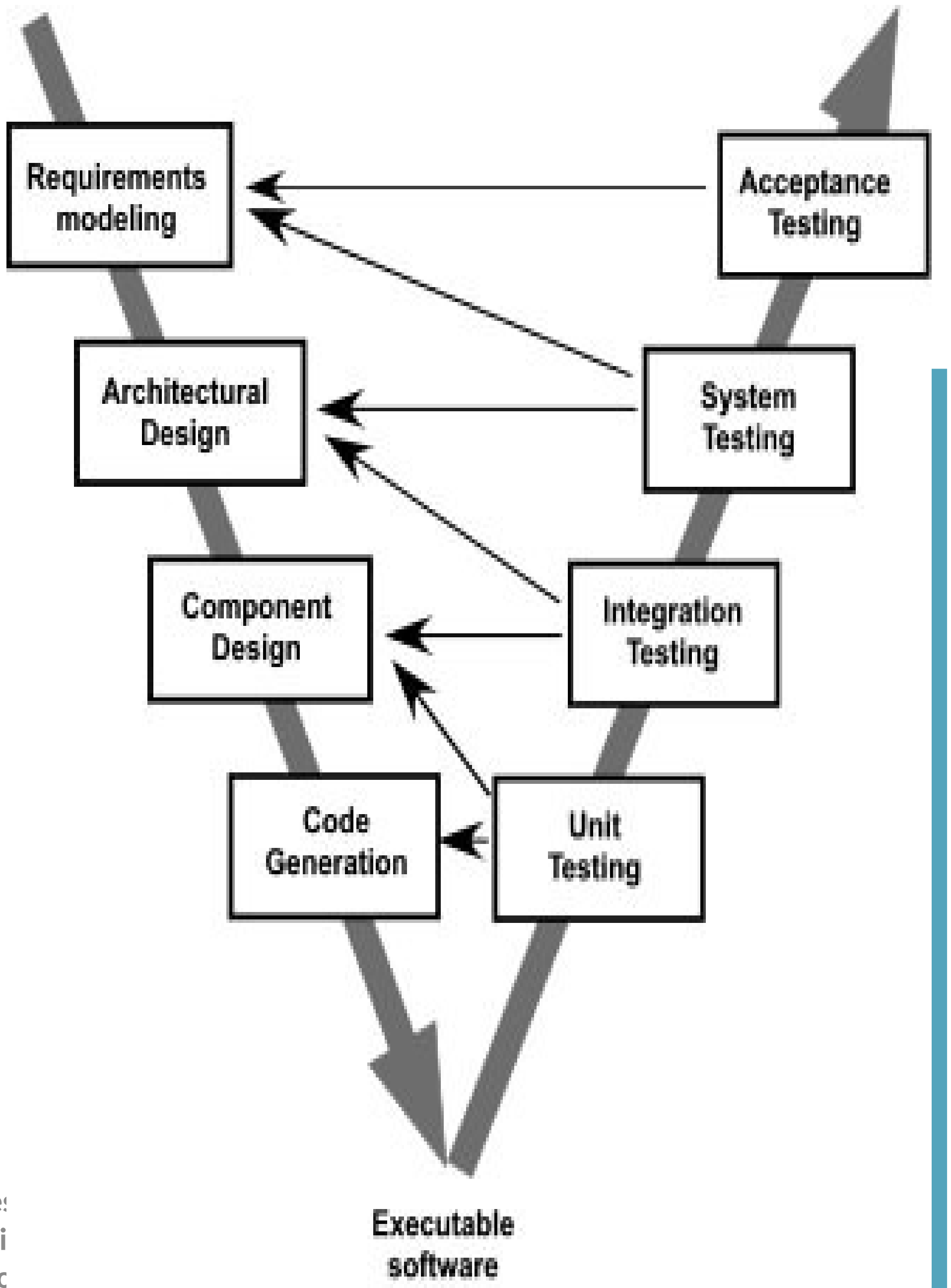
- Where requirements are fixed and is to proceed to completion in a linear manner.

Cons in Water fall model

- Real projects rarely follow the sequential flow that the model proposes

- It is often difficult for the customer to state all requirements explicitly

- The customer must have patience. A working version of the program(s) will not be available until late in the project time span



The:
desi
accc
*Engineering: A
Practitioner's
Approach, 7/e*

- The V-model provides a way of visualizing how verification and validation actions are applied to earlier engineering work

- V- model means Verification and Validation model. Just like the **waterfall model**, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins

- **Advantages of V-model:**

- 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.

- Defects are found at early stage.

- Works well for small projects where requirements are easily understood.

- **Disadvantages of V-model:**
- 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.

- **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 ample technical resources are available with needed technical expertise.

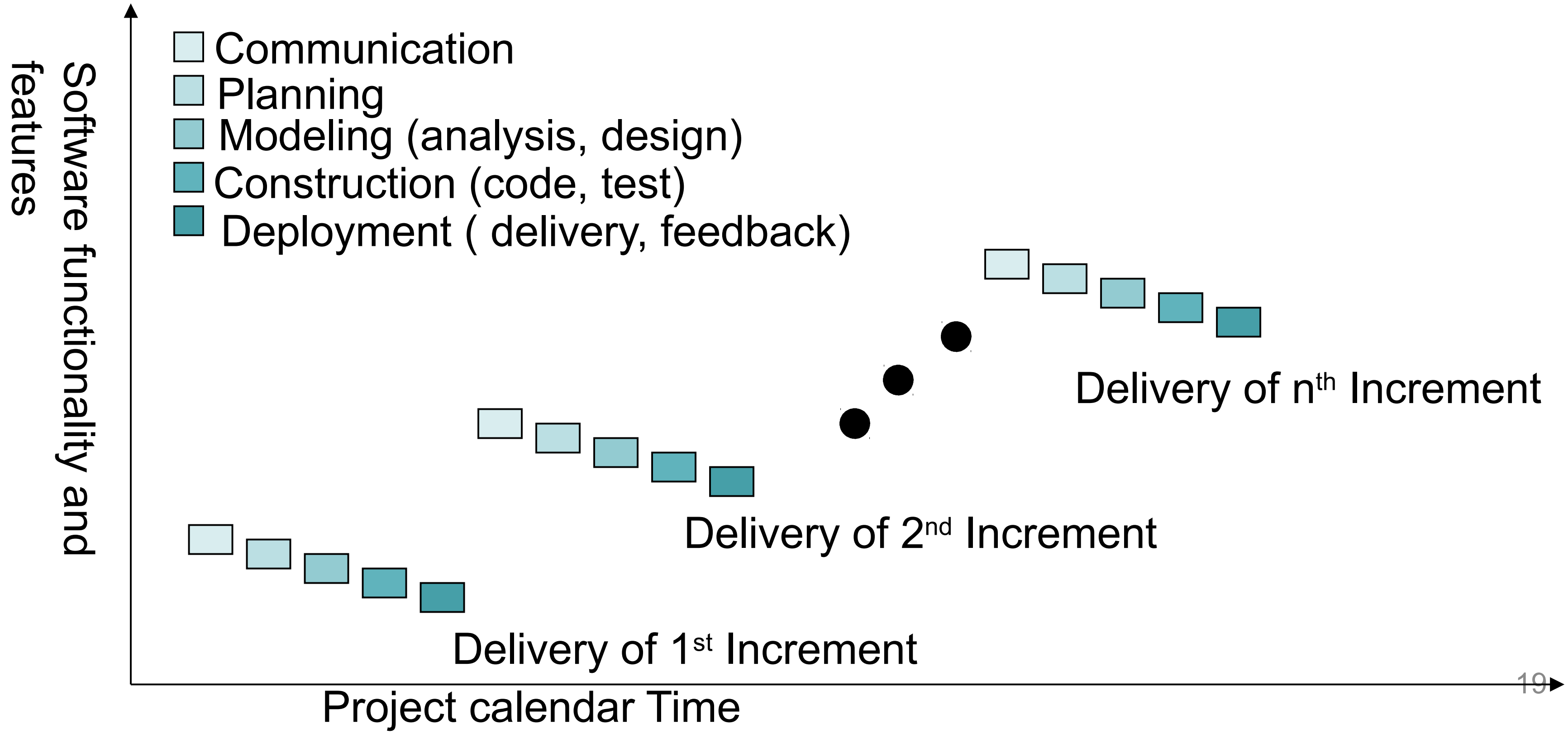
Prescriptive Process models

Incremental Process Model

- Combines elements of linear and parallel process flows.
- It delivers a series of releases, called increments that provide progressively more functionality for the customer as each is delivered
- The first increment is often a core product.
- The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional feature and functionality.
- It focuses on the delivery of an operational product with each increment.

Incremental Process Model

- It is useful when staffing is unavailable for a complete implementation.
- Increments can be planned to manage technical risks.



Software functionality and features

Project calendar Time

□ Communication

□ Planning

□ Modeling (analysis, design)

□ Construction (code, test)

□ Deployment (delivery, feedback)

Delivery of 1st Increment

Delivery of 2nd Increment

Delivery of nth Increment

For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment

- **Advantages of Incremental model:**
- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it’s iteration.

- **Disadvantages of Incremental model:**
- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than **waterfall**.

- **When to use the Incremental model:**
- This model can be used when the requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used

Evolutionary Process Model

- Software, like Complex systems evolve over a period of time

  -Business and product requirements often change as development proceeds.

  - tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure

  - a set of core product or system requirements is well understood, but the details of product or system extensions have yet to be defined.

- Evolutionary models are iterative.
- Evolutionary Process Model produce an increasingly more complete version of the software with each iteration.
- Specification, development and validation are interleaved.

Prescriptive Process models

Evolutionary Process Model

- Prototyping
- Spiral Model

Prototype Model is a software development life cycle model which is used when the customer is not known completely about how the end product should be and its requirements. So in this model, a prototype of the end product is first developed by the developers and then tested and changes were made as per customer feedback until the customer is satisfied with the prototype.

Evolutionary Process Model

1. Prototyping

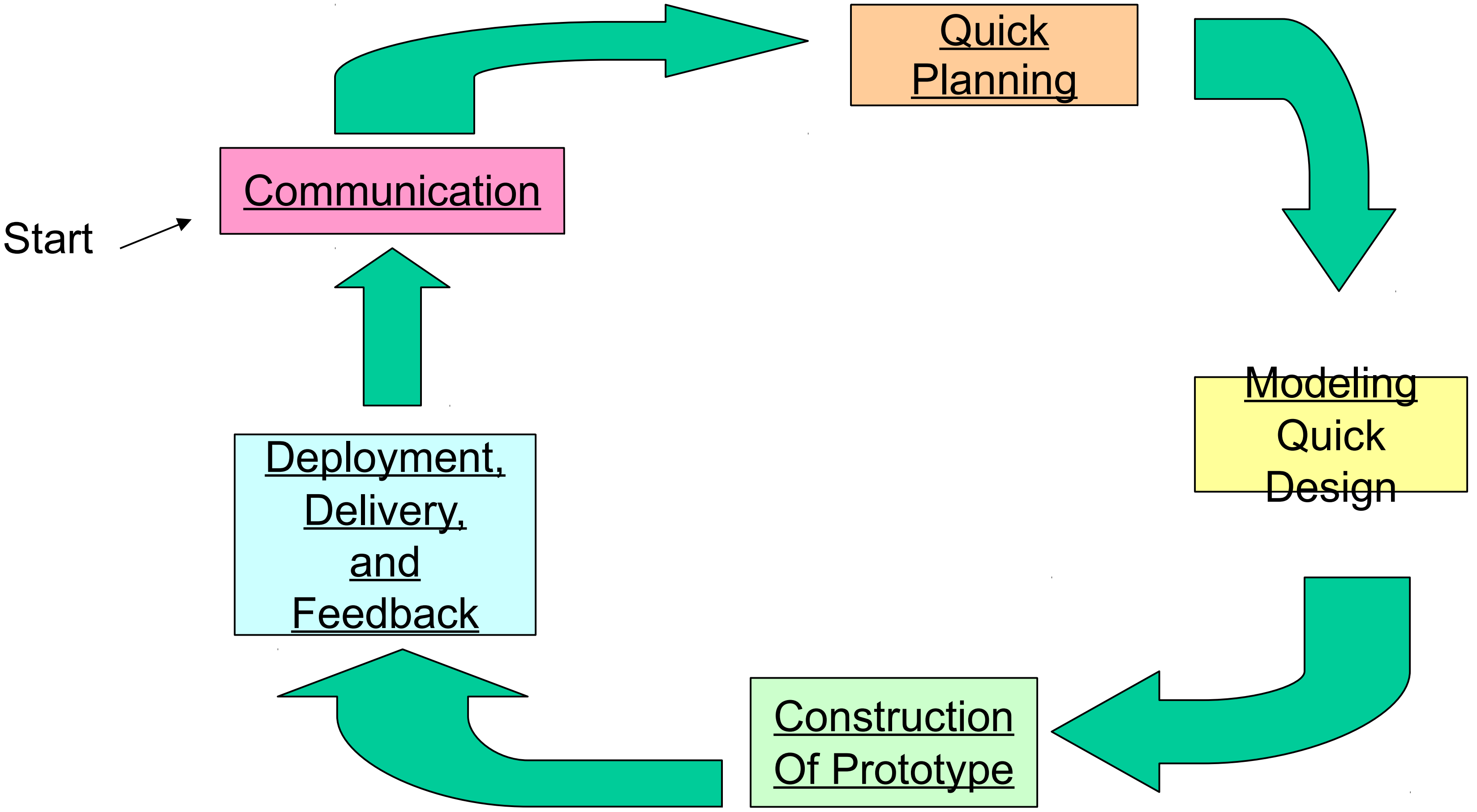
- It assists you and stakeholders to better understand what is to built when requirements are fuzzy.
- Prototyping paradigm

  -begins with communication.

  - planned quickly and modelling occurs

  - quick design focuses on a representation of those aspects of the software that will be visible to end users.
- It serves as a mechanism for identifying software requirements.
- a first or preliminary version of a device or vehicle from which other forms are developed

Prototyping Model(Diagram)



Start

Communication

Quick Planning

Modeling Quick Design

Construction Of Prototype

Deployment, Delivery, and Feedback

- **Advantages of Prototype model:**
- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified

- **Disadvantages of Prototype model:**
- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed
- Incomplete or inadequate problem analysis.
- **When to use Prototype model:**
- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.

Evolutionary Process Model
2. Spiral development

- Originally proposed by Barry Boehm.
- It couples the iterative nature of prototyping with the controlled and systematic aspects of water fall model.
- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

The process of development is fast.

There is control towards all phases of development.

Customer feedback is taken into consideration and changes implemented as soon as possible.

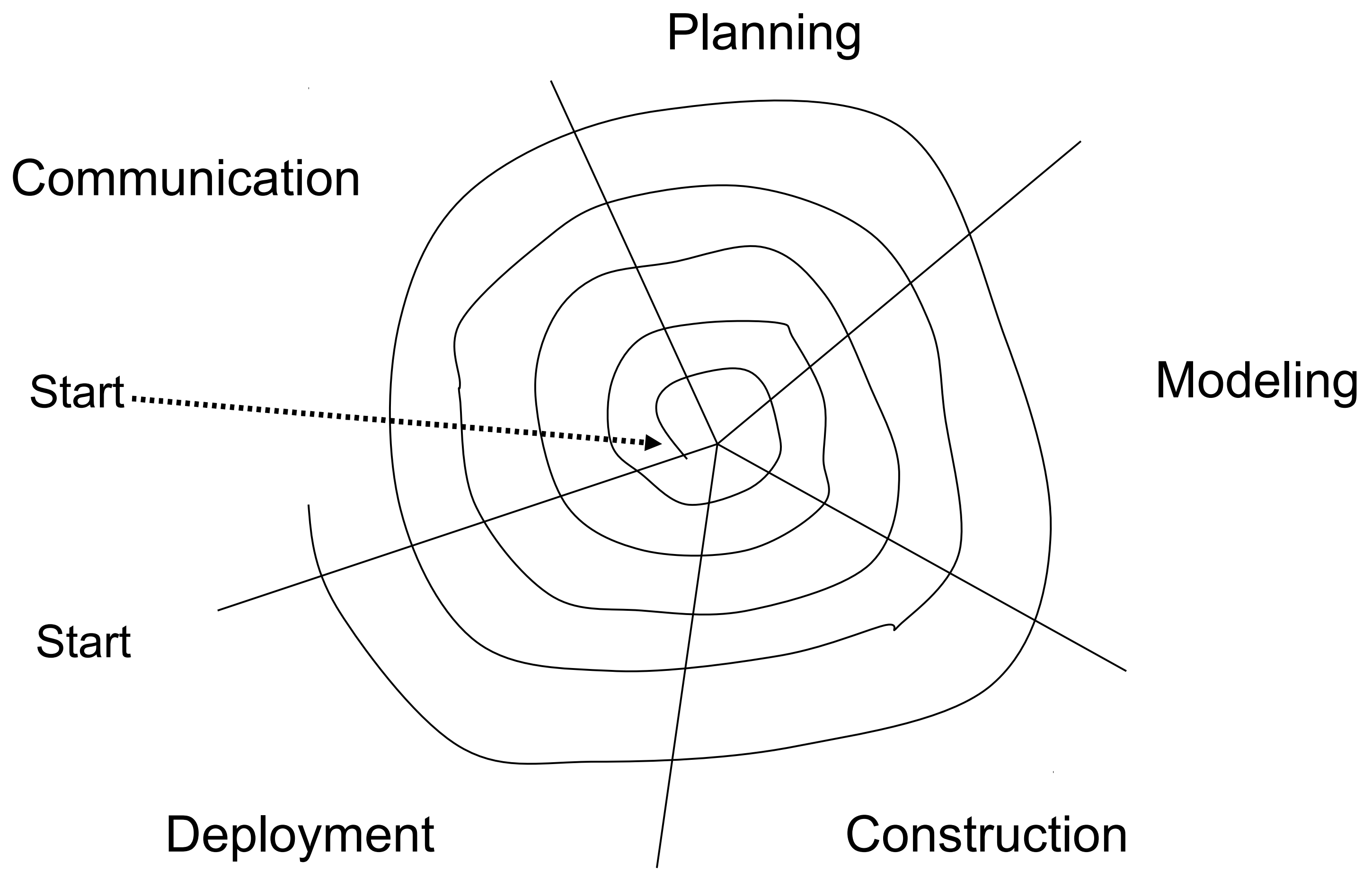
Many and more features are added systematically.

Most ideal for large and risky projects.

Spiral model

- It is a realistic approach to the development of large-scale systems and software.
- The software evolves as the process progresses, the developer and customer better understand and react at each evolutionary level.
- It uses prototyping as a risk reduction mechanism.
- It demands considerable risk assessment expertise and realise on this expertise for success.

Spiral Model (Diagram)



- **Advantages of Spiral model:**
- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the **software life cycle** .

- **Disadvantages of Spiral model:**
- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project’s success is highly dependent on the risk analysis phase.
- Doesn’t work well for smaller projects.
-

- **When to use Spiral model:**
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex

Evolutionary development

- Problems
  - Do not establish the max. speed of the evolution.
  - Systems are often poorly structured;
  - Special skills (e.g. in languages for rapid prototyping) may be required.
  - Project management and estimation technique do not fit completely.

- Applicability
  - For small or medium-size interactive systems;
  - For parts of large systems (e.g. the user interface);
  - For short-lifetime systems.

Concurrent Model

- The concurrent development model sometimes called Concurrent Engineering.
- It allows team to represent iterative and concurrent elements of any of the process models.
- All software engineering activities exist concurrently but reside in different states.
- Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the activities.
- Concurrent modeling is applicable to all types of software development and provide an accurate picture of the current state of a project.

Concurrent models are those **models** within which the various activities of software development happen at the same time, for faster development and a better outcome. The **concurrent model** is also referred to as a parallel working **model**

Advantages of the concurrent development 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.

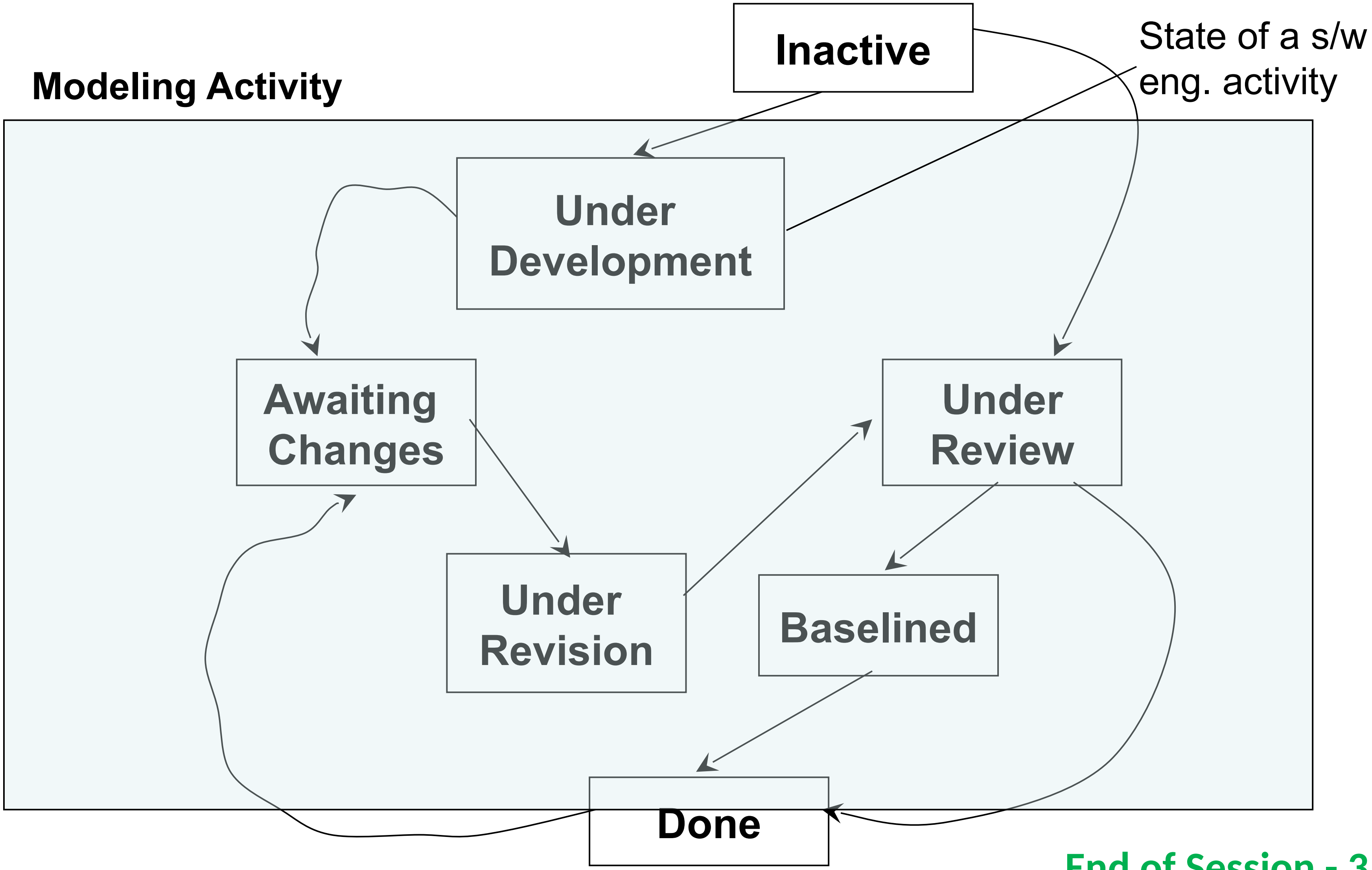
It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development 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.

One element of the concurrent process model



End of Session - 3

Questions:

1. What Classic life cycle model.
2. Disadvantages of Incremental Process Model.
3. Main drawback of Prototyping Model.
4. Speciality in Spiral Model.
5. Need of Process models in Software development.
6. Advantages of Spiral model.
7. Disadvantages of Spiral model.
8. Advantages of Evolutionary model.
9. Disadvantages of Evolutionary model.
10. Specialty of Concurrent model.
11. Difference between Waterfall model and Spiral model.

Session – 4

Specialized Process Models

K L University
u/s 3 of UGC Act. 1956
Koneru Lakshmaiah Education Foundation

Component-based Development Model

- The process to apply when reuse is a development objective
- It Consists of the following process steps
  - Available component-based products are researched and evaluated for the application domain in question
  - Component integration issues are considered
  - A software architecture is designed to accommodate the components
  - Components are integrated into the architecture
  - Comprehensive testing is conducted to ensure proper functionality

- Capitalizes on software reuse, which leads to documented savings in project cost and time

Formal Methods Model (Description)

- Encompasses a set of activities that leads to formal mathematical specification of computer software
- Enables a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation
- Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily through mathematical analysis
- Offers the promise of defect-free software
- Used often when building safety-critical systems

Formal Methods Model (Challenges)

- Development of formal methods is currently quite 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

Aspect-Oriented Software Development

It provides a process and methodological approach for defining, specifying, designing, and constructing aspects

As modern computer based systems become more sophisticated and complex there are certain ***concerns by the customer***

- required properties or areas of technical interest.
- Span the entire architecture
- High-level properties of a system (e.g; security, fault tolerance)
- Other concern affect functions (e.g; the application of business rules)
- While others are systemic (e,g; task synchronization or memory management)

The Unified Process

K L University

u/s 3 of UGC Act. 1956

Koneru Lakshmaiah Education Foundation

Background

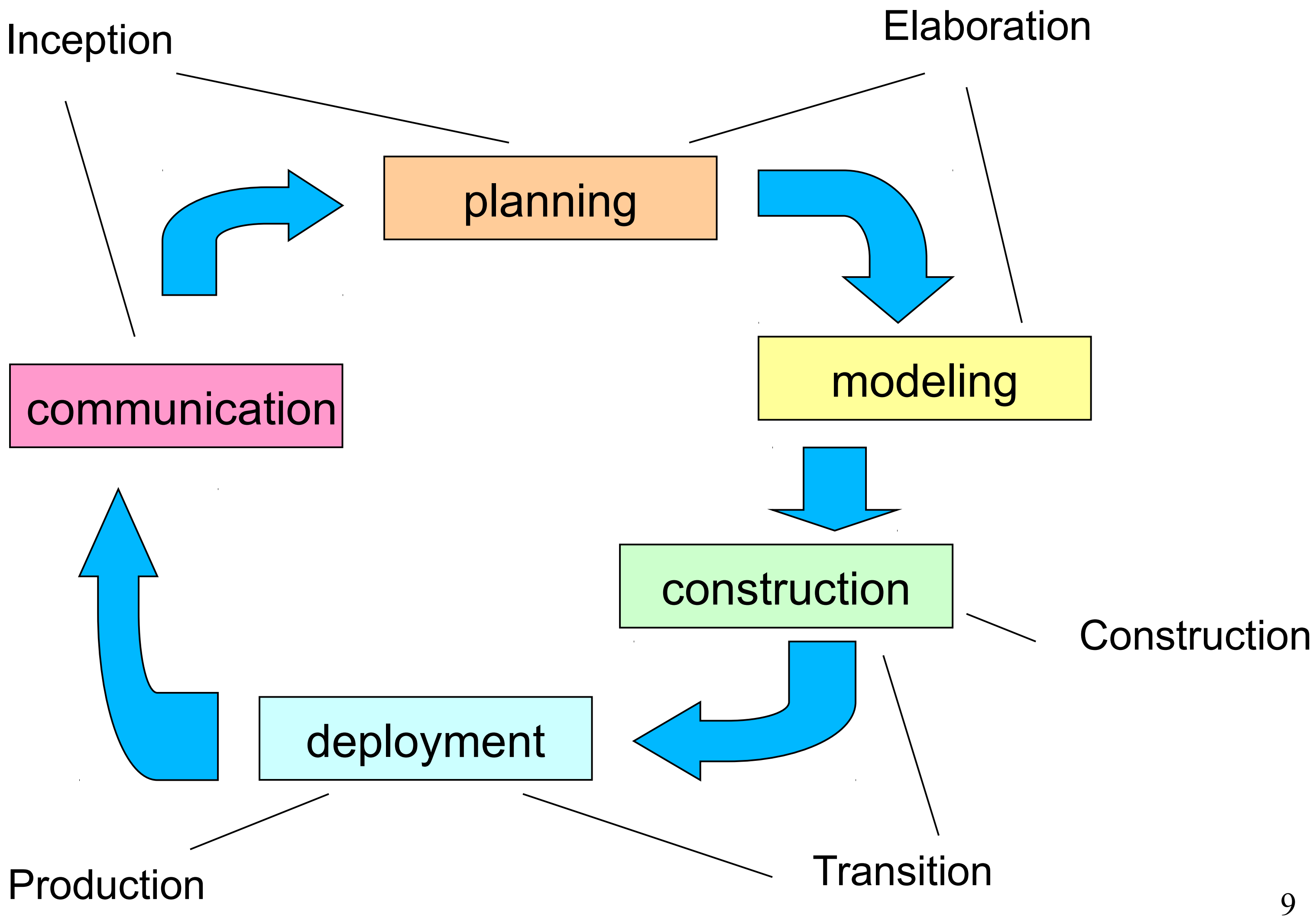
- During early 1990s JmaesRumbaugh, Grady Booch and Ivar Jacobson eventually worked together on a unified method, called the Unified Modelling Language (UML)
  - UML is a robust notation for the modelling and development of object-oriented systems
  - UML became an industry standard in 1997
  - However, UML does not provide the process framework, only the necessary technology for object-oriented development

- Unified process developed which is a framework for object-oriented software engineering using UML
  - Draws on the best features and characteristics of conventional software process models
  - Emphasizes the important role of software architecture
  - Consists of a process flow that is iterative and incremental, thereby providing an evolutionary feel

Background (continued)

- Consists of 5 phases:
  1. Inception
  2. Elaboration
  3. Construction
  4. Transition
  5. Production

Phases of the Unified Process



(1) - Inception Phase

- Encompasses both customer communication and planning activities of the generic process

- Business requirements for the software are identified

- A rough architecture for the system is proposed

- A plan is created for an incremental, iterative development

- Fundamental business requirements are described through preliminary use cases
  - A use case describes a sequence of actions that are performed by a user

(2) - Elaboration Phase

- Encompasses both the planning and modelling activities of the generic process
- Refines and expands the preliminary use cases
- Expands the architectural representation to include five views
  - Use-case model
  - Analysis model
  - Design model
  - Implementation model
  - Deployment model
- Often results in an executable architectural baseline that represents a first cut executable system
- The baseline demonstrates the viability of the architecture but does not provide all features and functions required to use the system

(3) - Construction Phase

- Encompasses the construction activity of the generic process
- Uses the architectural model from the elaboration phase as input
- Develops or acquires the software components that make each use-case operational
- Analysis and design models from the previous phase are completed to reflect the final version of the increment
- Use cases are used to derive a set of acceptance tests that are executed prior to the next phase

(4) - Transition Phase

- Encompasses the last part of the construction activity and the first part of the deployment activity of the generic process
- Software is given to end users for beta testing and user feedback reports on defects and necessary changes
- The software teams create necessary support documentation (user manuals, trouble-shooting guides, installation procedures)
- At the conclusion of this phase, the software increment becomes a usable software release

(5) - Production Phase

- Encompasses the last part of the deployment activity of the generic process
- On-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated

Unified Process Work Products

- Work products are produced in each of the first four phases of the unified process

- In this course, we will concentrate on the analysis model and the design model work products

- Analysis model includes
    - Scenario-based model, class-based model, and behavioural model

- Design model includes
    - Component-level design, interface design, architectural design, and data/class design

End of session - 4

Questions:

1. Main importance of Unified Process model.
2. Various Challenges in Formal Methods Model.
3. Difference between Inception and Elaboration.
4. Importance of Construction Phase in Unified process model.
5. Need and importance of UML in software development.

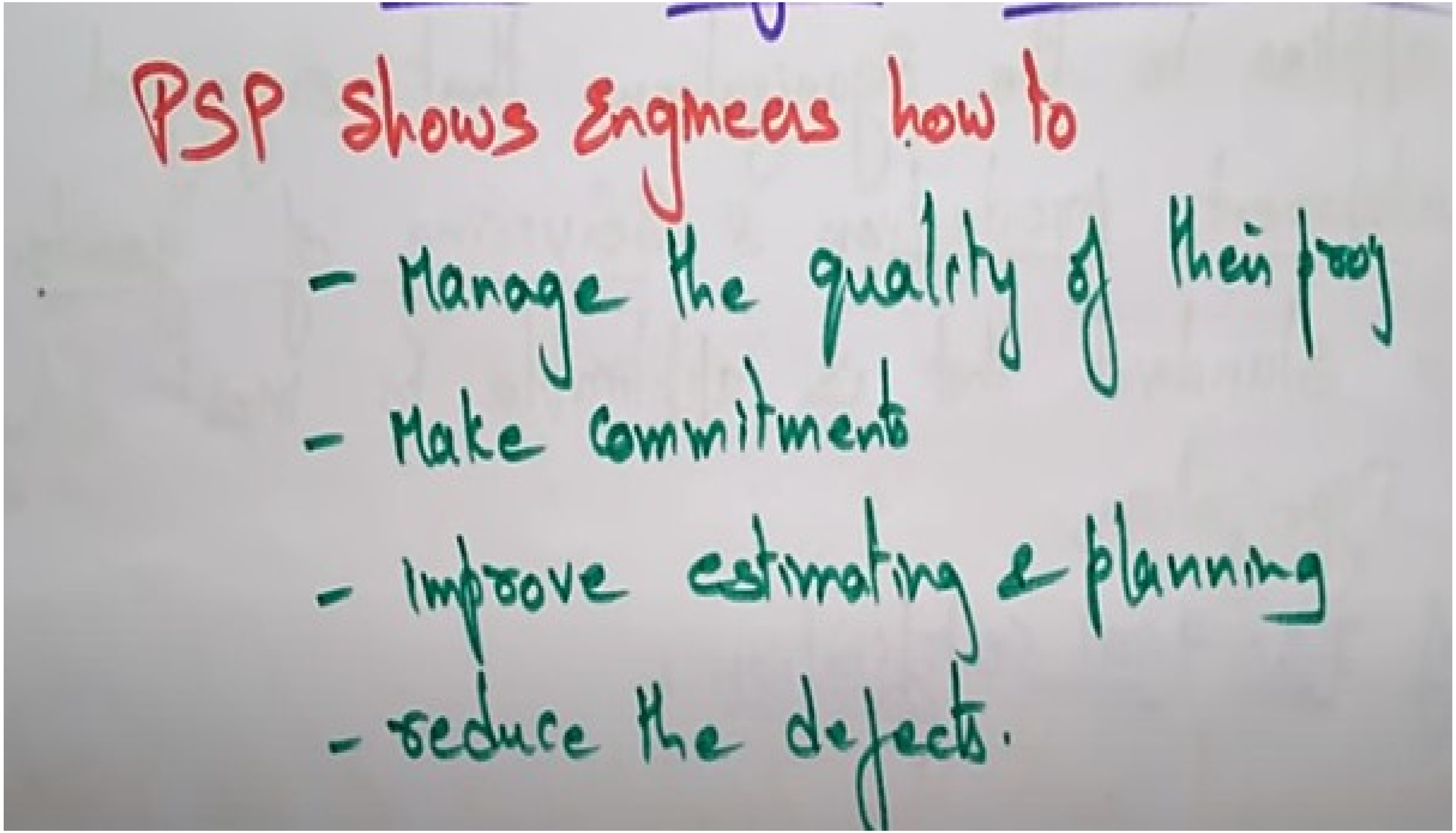
Session -5

Personal and Team Process models

Personal and Team Process models

- The software process is effective only if it is amenable to significant adaption to meet the needs of the project team.
- **Personal process model:** Individual create a process that best fits your needs, and at the same time, meets the broader needs of the team and the organization.
- **Team process model:** The Team itself create its own process, and at the same time meet the narrower needs of individuals and the broader needs of the organisation.

"Watts Humphrey" *argues that both models are achievable but require hard work, training and coordination.*



PSP shows Engineers how to

- Manage the quality of their proj
- Make commitments
- improve estimating & planning
- reduce the defects.

Personal Software Process (PSP)

Watts Humphrey suggests that in order to change an ineffective personal process, an individual must move through four phases, each requiring training and careful instrumentation.

The PSP

- emphasizes personal measurement of the work product that is produced .
- emphasizes the resultant quality of the work product.
- Makes practitioner responsible for project planning (e.g; estimating and scheduling)
- Empowers the practitioner to control the quality of all software work products that are developed.

Personal Software Process (PSP) defines 5 framework activities

- **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.** An external specification is created for each component and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.

Personal Software Process (PSP)

- **High-level design review**. Formal verification methods 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 measures and metrics collected the effectiveness of the process is determined. (Final O/P If this is a large amount of data it should be analyzed statistically.) Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

Personal Software Process (PSP)

- PSP Stresses the need for each software engineer to identify errors early and as important, to understand the types of errors through a rigorous assessment activity performed on all work products you produce.

- However, PSP has not been widely adopted throughout the industry.
- PSP is intellectually challenging and demands a level of commitment (by practitioners and managers) that is not always possible to obtain.
- Training is relatively lengthy and training costs are high.
- The required level of measurement is culturally difficult for many software people.

Team Software Process (TSP)

The goal of TSP is to build a “self-directed” project team that organizes itself to produce high quality software. **Humphrey** defines the following objectives for 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. **End of Session -5**

Additional topics for self learning:

- Product and Process

Questions:

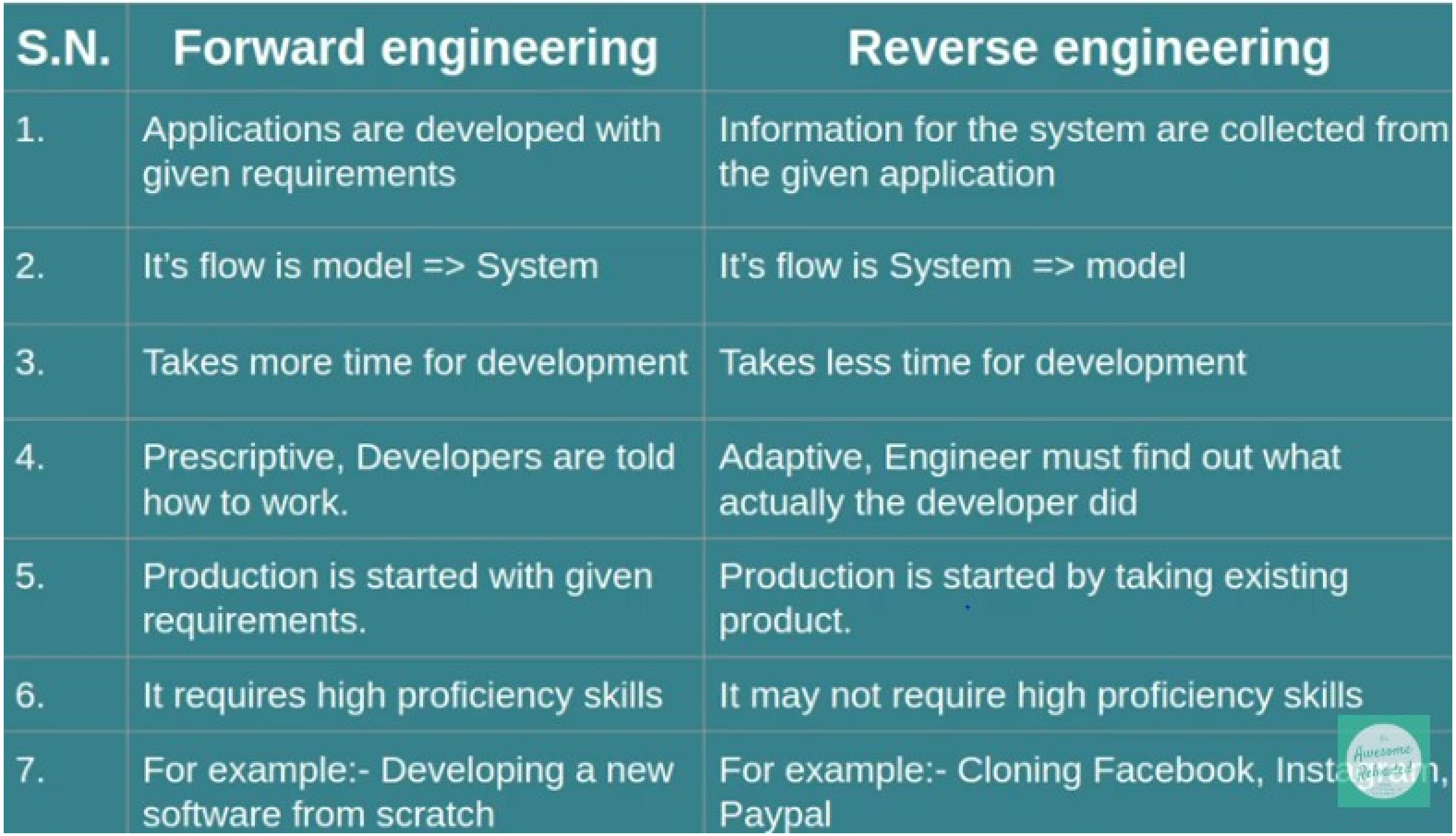
1. What is PSP.
2. What is TSP.
3. Differences between PSP and TSP.
4. How PSP used for Quality improvement.
5. How TSP used for Quality improvement.

Session - 6

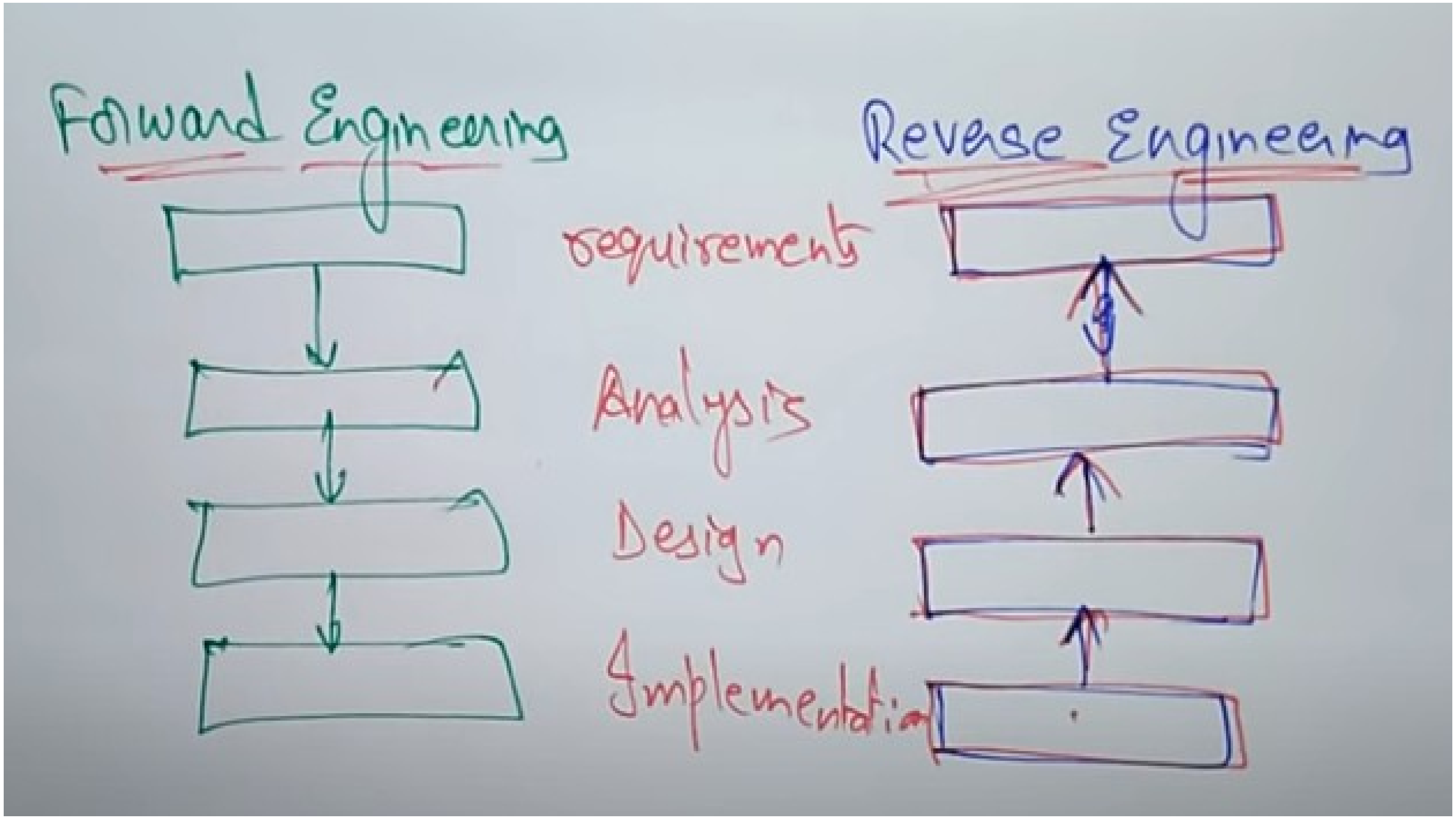
Reverse Engineering

- Forward Engineering
- In software development life cycle , we start from requirement phase ,analysis , design and implementation

- Reverse Engineering
- It is the process of creating of design document from source code and the requirements from design



S.N.	Forward engineering	Reverse engineering
1.	Applications are developed with given requirements	Information for the system are collected from the given application
2.	It’s flow is model => System	It’s flow is System => model
3.	Takes more time for development	Takes less time for development
4.	Prescriptive, Developers are told how to work.	Adaptive, Engineer must find out what actually the developer did
5.	Production is started with given requirements.	Production is started by taking existing product.
6.	It requires high proficiency skills	It may not require high proficiency skills
7.	For example:- Developing a new software from scratch	For example:- Cloning Facebook, Instagram, Paypal


Forward Engineering
Revease Engineering
requirements
Analysis
Design
Implementation

It is the process followed as to find unknown and difficult information about a software

Why it is important

It is important in case when software lacks proper documentation , and it is highly unstructured

It is degraded through maintenance works

Purpose

Recovering information from existing code and intermediate documents

Program understanding at any level it is included in RE

- Reverse Engineering Goals

1)Recover Lost information

2)Cope with complexity

3)Detect side Effects

4)Synthesize higher abstraction

5)Facilitate reusability

Uses of reverse engineering

Program understanding

Re-documentation or document generation

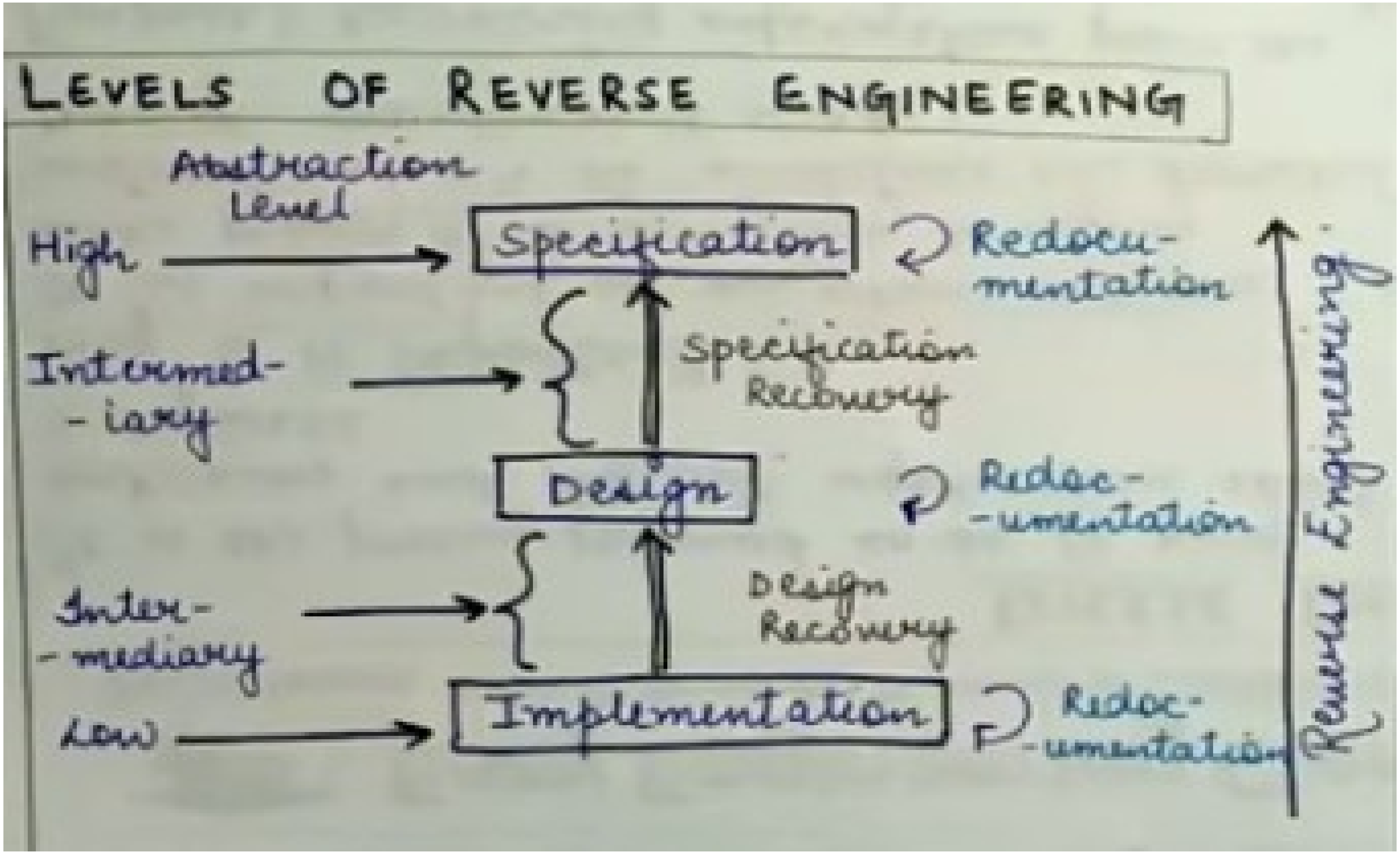
Recovery of design details

Business rules implied on software

Understanding high level system description

Requirement Engineering activities

- Understanding the process
- Understanding the data
- Understanding user interfaces


LEVELS OF REVERSE ENGINEERING
Abstraction level
High
Intermed-
-iary
Inter-
-mediary
Low
Specification
Specification Recovery
Design
Design Recovery
Implementation
Redocu-
mentation
Redoc-
-umentation
Redoc-
-umentation
Reverse Engineering

Questions:

1. Need of Reverse engineering.
2. How Forward Engineering work.
3. Different levels of reverse engineering