BCSE301L_SOFTWARE ENGINEERING

MODULE 1

Overview of Software Engineering

Presented By,

Dr.Baiju B V

Assistant Professor

SCOPE, VIT, Vellore

COURSE OBJECTIVES

1. To introduce the essential Software Engineering concepts.

2. To impart concepts and skills for performing analysis, design ,develop, test and evolve efficient software systems of various disciplines and applications

3. To make familiar about engineering practices, standards and metrics for developing software components and products.

COURSE OUTCOME

1. Apply and assess the principles of various process models for the software development.

2. Demonstrate various software project management activities that include planning , Estimations, Risk assessment and Configuration Management

3. Perform Requirements modelling and apply appropriate design and testing heuristics to produce quality software systems.

4. Demonstrate the complete Software life cycle activities from requirements analysis to maintenance using the modern tools and techniques.

5. Escalate the use of various standards and metrics in evaluating the process and product

MODULE – I (CONTENTS)

- Nature of Software
- Software Engineering
- Software process, project, product
- Process Models
- Classical Evolutionary models
- Introduction to Agility
- Agile Process
- Extreme programming
- XP Process
- Principles of Agile Software Development framework
- Overview of System Engineering

Nature of Software

1. Software

Software takes on a dual role.

(i) Software is a product

- It delivers the computing potential embodied by computer hardware.
- It is an **information transformer** - ***producing***, ***managing***, ***acquiring***, ***modifying***, ***displaying***, or ***transmitting information*** that can be as simple as a single bit or as complex as a multimedia presentation derived from data acquired from dozens of independent sources.

The online learning platform itself is a software product. It embodies the computing potential of hardware and provides users with the capability to:

- ✓ ***Access course content (text, video, and multimedia).***
- ✓ ***Manage personal learning progress (e.g., tracking completed lessons or assignments).***
- ✓ ***Acquire knowledge through interactive quizzes and virtual labs.***
- ✓ ***Modify and customize the learning experience (e.g., adjusting video playback speed, changing language settings).***

(ii) Software is the vehicle for delivering a product.

- Supports or directly provide system functionality.
- Control other programs (Eg. Operating System)
- Effects Communication (Eg. Networking Software)
- Helps build other software (Eg. Software tools and environment)

System Functionality Support: Platform enables users to register for courses, track progress, and interact with instructors or peers.

Control Other Programs : Mobile app version of the platform interacts with the underlying operating system (iOS or Android) to provide push notifications, manage permissions (e.g., access to storage or camera), and optimize resource usage.

Effect Communication: Networking software embedded in the platform facilitates real-time interactions, such as live video sessions, chat support, or peer discussions in forums.

Help Build Other Software: The platform may provide an integrated development environment (IDE) or coding workspace within certain courses (e.g., programming tutorials). This helps learners write, test, and debug code directly in the browser or app.

Nature of Software

Software is:

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

(2) ***data structures*** that enable the programs to adequately manipulate information.

(3) ***documentation*** that describes the operation and use of the programs.

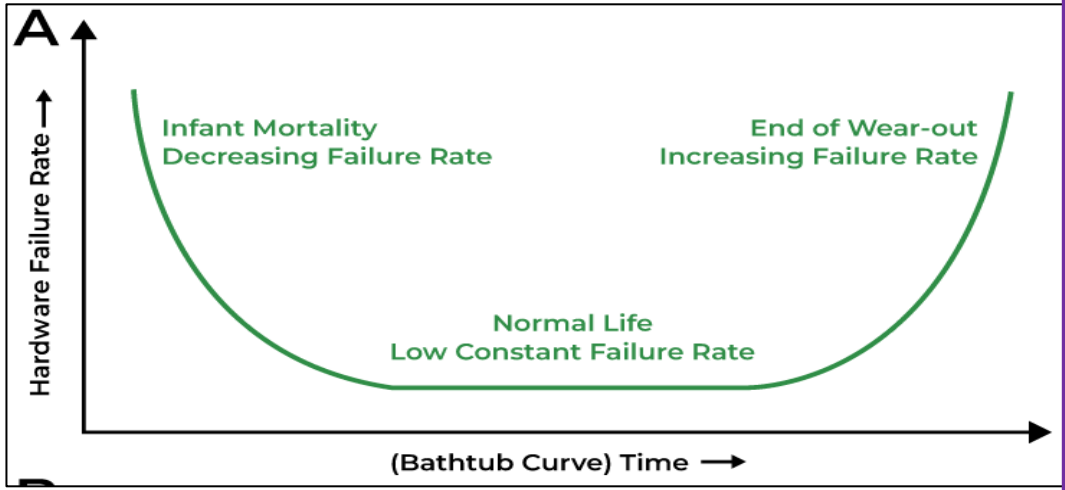
- Software is a ***logical rather than a physical system element***.
- Software has characteristics that are considerably different than those of hardware:

(i) Software is developed or engineered; it is not manufactured in the classical sense.

- In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
- Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different.
- Both activities require the construction of a "product," but the approaches are different.
- Software costs are concentrated in engineering.
- This means that software projects cannot be managed as if they were manufacturing projects.

(ii) Software doesn't "wear out."

- Figure depicts failure rate as a function of time for hardware.



- ***Bathtub Curve*** is generally graph that is used to ***graphically demonstrate run-to-failure maintenance strategy***. It indicates that
  - Hardware exhibits relatively high failure rates early in its life
  - Defects are corrected and the failure rate drops to a steady-state level (quite low) for some period of time
- The failure rate rises again as hardware components suffer from the increasing effects of ***dust***, ***vibration***, ***mishandling***, ***temperature extremes***, and many other environmental conditions.

Hardware begins to wear out.

Early Failures (High Failure Rate):

- When you first buy a new smartphone, some devices may have manufacturing defects (e.g., faulty screens, defective batteries) causing early failures.
- These issues are fixed through warranty repairs or replacements.

Steady State (Low Failure Rate):

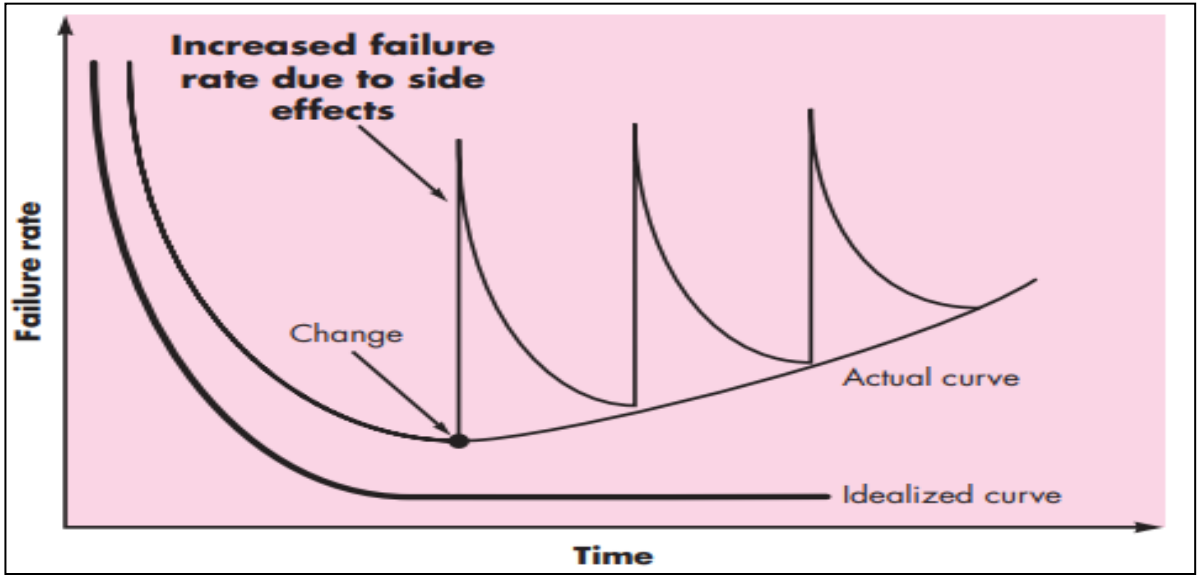
- After initial defects are resolved, the smartphone performs reliably for years with minimal failures.

Wear-Out Failures (Rising Failure Rate):

- Over time, hardware starts to degrade due to factors like battery aging, dust in the ports, or damage from drops.
- Failures become more frequent as the phone gets older.

- Software is not susceptible to the environmental conditions that cause hardware to wear out.
- The failure rate curve for software should take the form of the "***idealized curve***" .
- **Undiscovered defects** will cause high failure rates early in the life of a program.
- The idealized curve is a gross simplification of actual failure models for software

- During its life, software will undergo change.
- As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the ***"actual curve"***



Early Failures (High Failure Rate):

- When the smartphone's software (e.g., operating system) is newly released, bugs or undiscovered defects may cause apps to crash or features to malfunction.
- Updates (patches) are released to fix these bugs, improving reliability.

Idealized Steady State (Low Failure Rate):

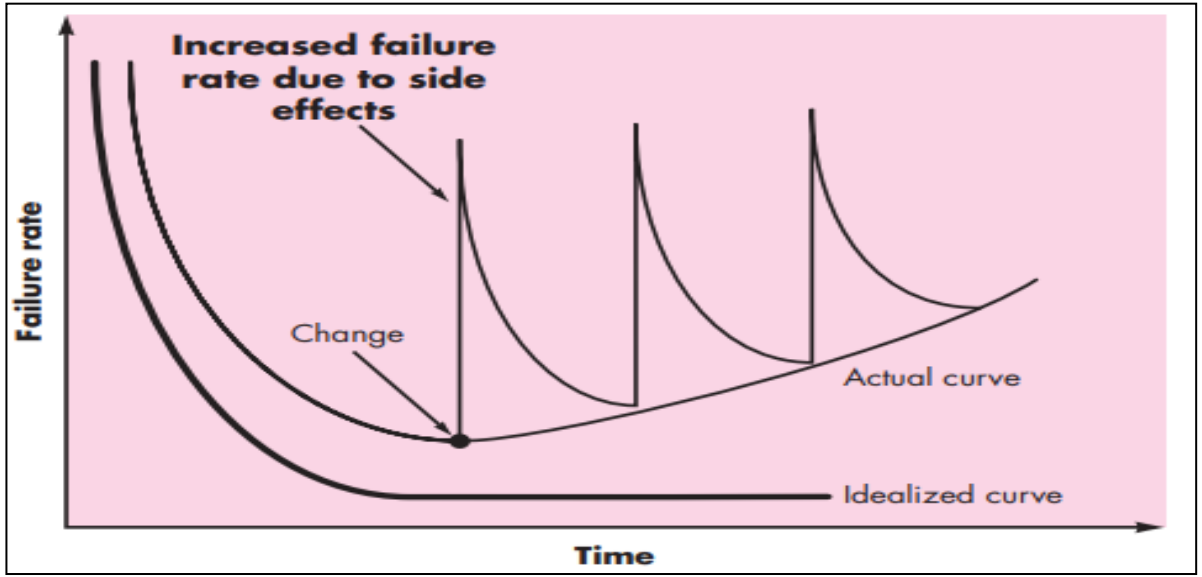
- Once most bugs are fixed, the software runs smoothly for a period, without significant failures.

Actual Failures with Changes (Spikes in Failure Rate):

- Over time, the operating system or apps receive updates (new features or bug fixes).
- These updates may accidentally introduce new bugs, causing temporary spikes in the failure rate.
- For example, after a major software update, users might experience slower performance or battery drain until a subsequent patch is released to fix the issues.

- Before the curve return to the original steady-state failure rate, another change is requested, causing the curve to spike again.

- Slowly, the minimum failure rate level begins to rise - the software is deteriorating due to change.



Software doesn't wear out. But it does deteriorate!

- When a hardware component experiences wear and tear, it can be substituted with a spare part.
- There are no equivalent spare parts for software.
- Every software failure indicates an error in design or in the process through which design was translated into machine executable code.

(iii) Although the industry is moving toward component-based construction, most software continues to be custom built.

- An engineering field develops and produces a set of standard design components.
- ***Standard screws*** and ***off-the-shelf integrated circuits*** are only two of thousands of standard components that are used by mechanical and electrical engineers as they design new systems.

- In the **hardware world**, component reuse is a natural part of the engineering process.
- In the **software world**, it is something that has only begun to be achieved on a broad scale.

- A software component should be designed and implemented so that it can be reused in many different programs.
- Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts.

- For example, interactive user interfaces are built with reusable components that enable
  - → the creation of graphics windows,
  - → pull-down menus,
  - → a wide variety of interaction mechanisms.

- The data structures and processing detail required to build the interface are contained within a library of reusable components for interface construction.

2. Software Application Domains

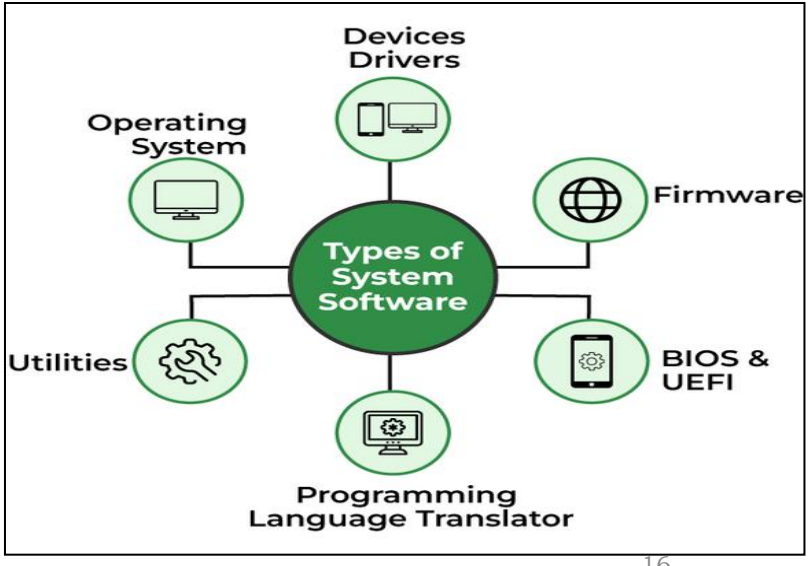
- There are currently seven main categories of computer software that software engineers must deal with.

(i) System Software

- System software is a collection of ***programs written to service other programs***.
- Some system software processes complex, but determinate, information structures.
- Example
  - ✓ ***Compilers***
  - ✓ ***Editors***
  - ✓ ***File Management Utilities***

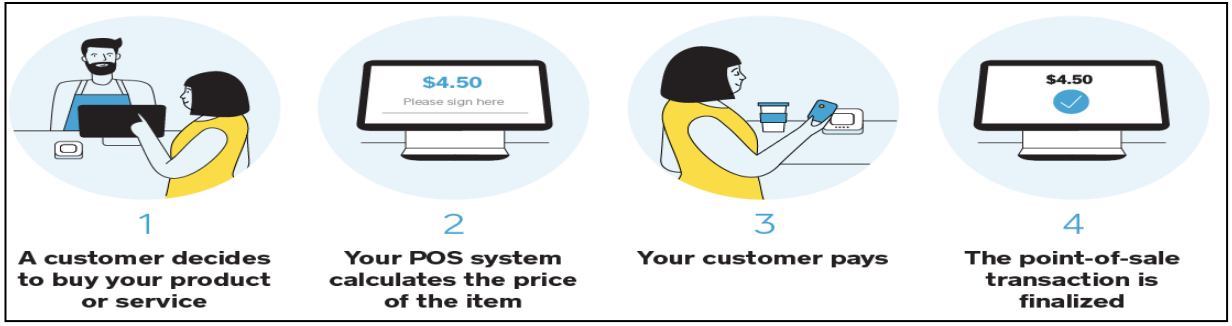
- Systems applications process largely unknown data.
- Example
  - ✓ **Operating system components**
  - ✓ **Drivers**
  - ✓ **Networking software**
  - ✓ **Telecommunications processors**

- The systems software area is characterized by
  - ✓ **heavy interaction with computer hardware**
  - ✓ **heavy usage by multiple users**
  - ✓ **concurrent operation** that requires
    - ***Scheduling***
    - ***Resource sharing***
    - ***Sophisticated process management***
    - ***Complex data structures***
    - ***Multiple external interfaces***



(ii) Application Software

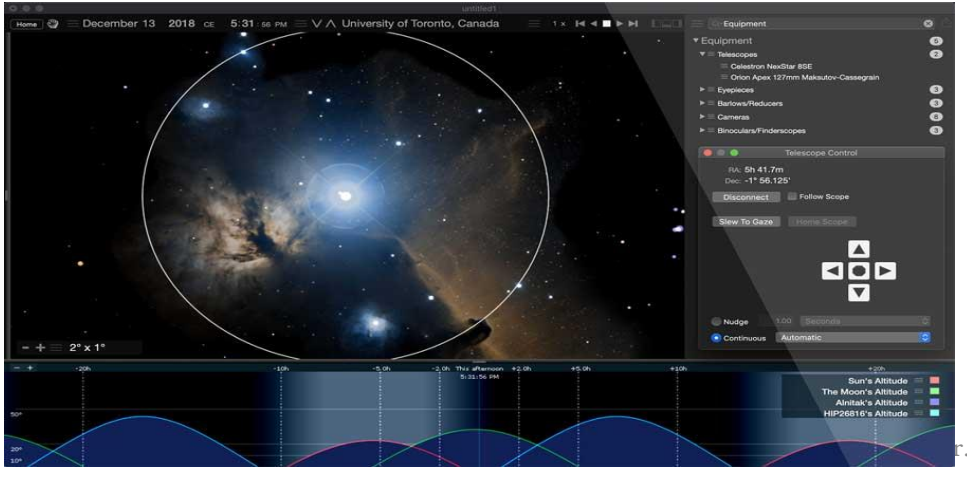
- Application software is a stand-alone programs that solve a specific business need.
- Application software is used to control business functions in real time.
- Example
  - ✓ ***Point-of-sale transaction processing***

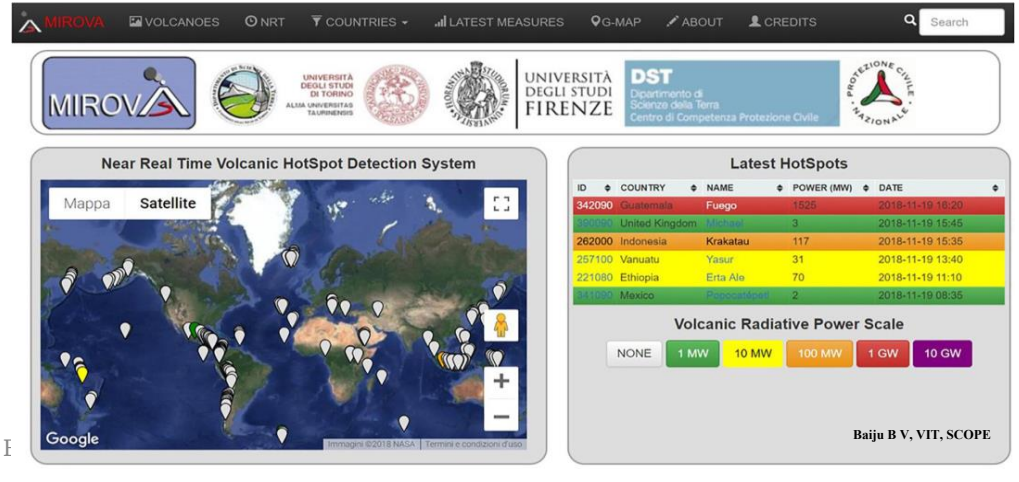


  - ✓ ***Real-time manufacturing process control***

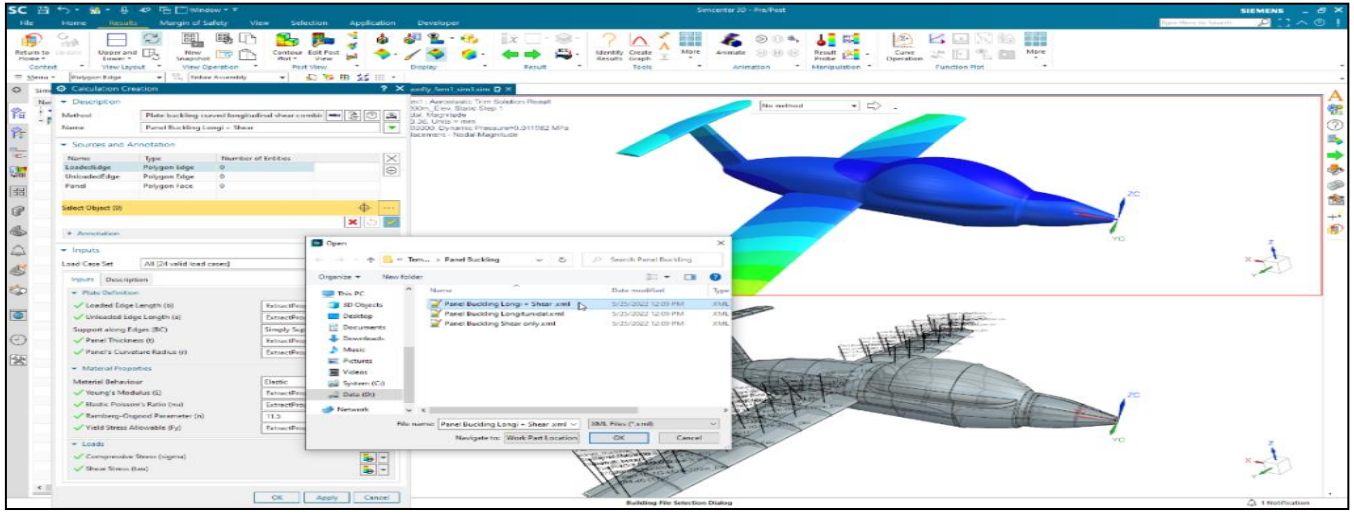
(iii) Engineering /Scientific Software

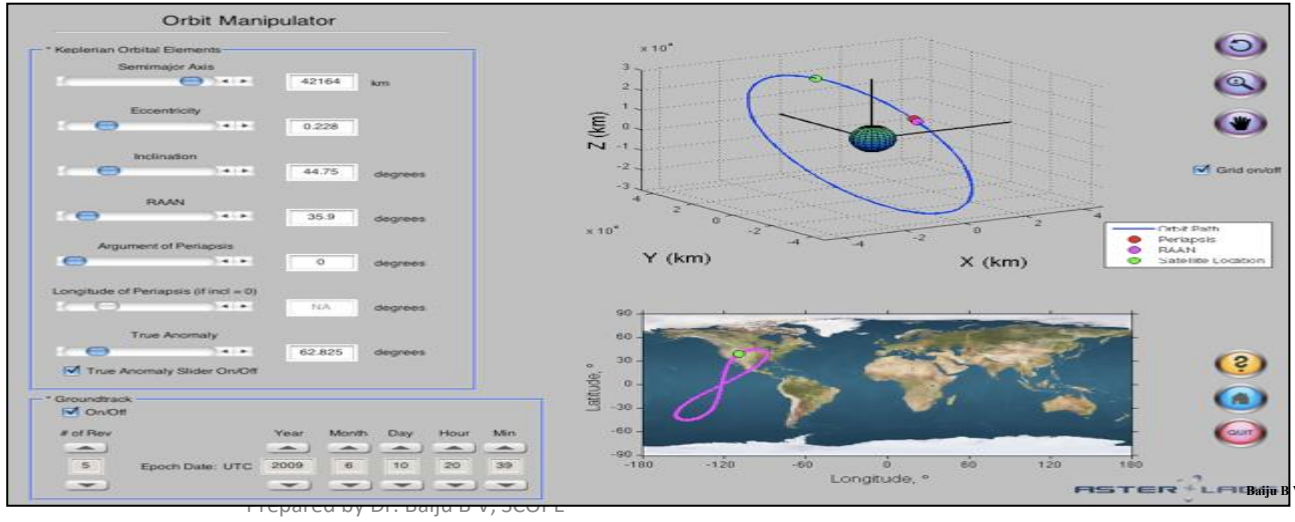
- This has been characterized by "***number crunching***" algorithms (the act of processing numerical data).
- Applications range from
  - ***Astronomy to Volcanology***



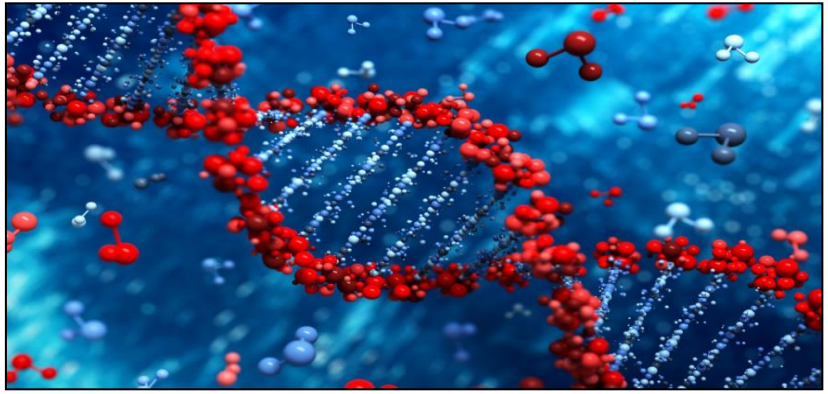


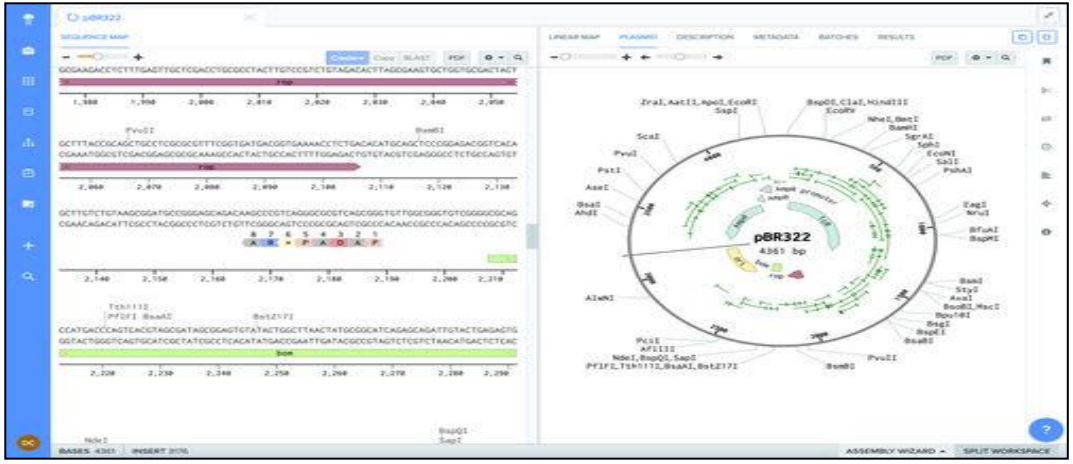
- Automotive stress analysis to Space shuttle orbital dynamics





- Molecular biology to Automated manufacturing





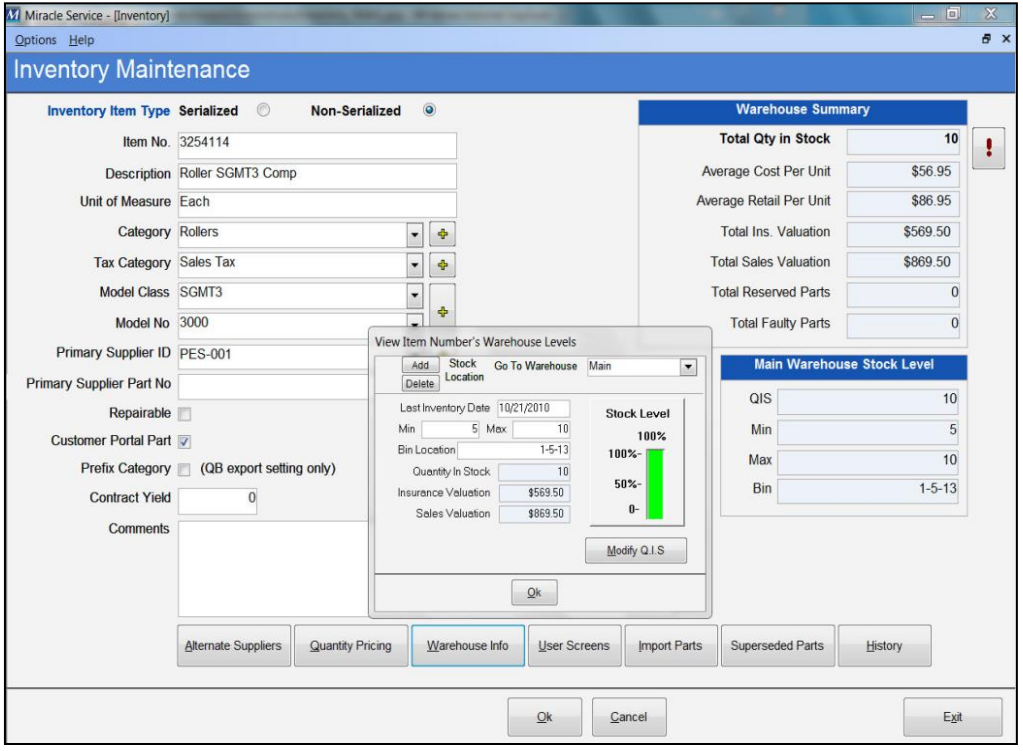
(iv) Embedded Software

- Embedded software resides within a product or system and is used to implement and control features and functions for the end user and for the system itself.
- Embedded software can
  - ***Perform limited and esoteric functions*** (e.g., key pad control for a microwave oven)

- ***Provide significant function and control capability*** (e.g., digital functions in an automobile such as fuel control, dashboard displays, and braking systems).

(v) Product-line Software

- Product-line software is designed to provide a specific capability for use by many different customers.
- Product-line software can focus on
  - A limited and esoteric marketplace (e.g., inventory control products)

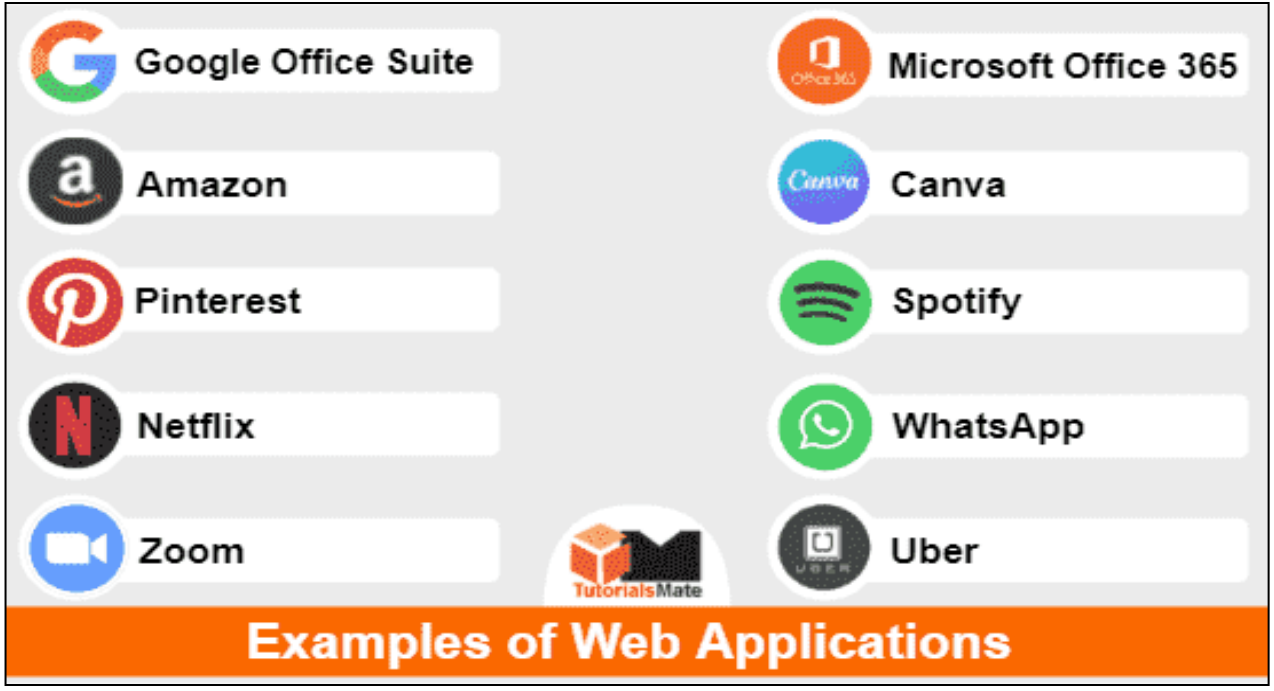


- Address mass consumer markets. Example
  - Word processing
  - Spreadsheets
  - Computer graphics
  - Multimedia
  - Entertainment
  - Database Management,
  - Personal and Business Financial Applications

(vi) Web Applications

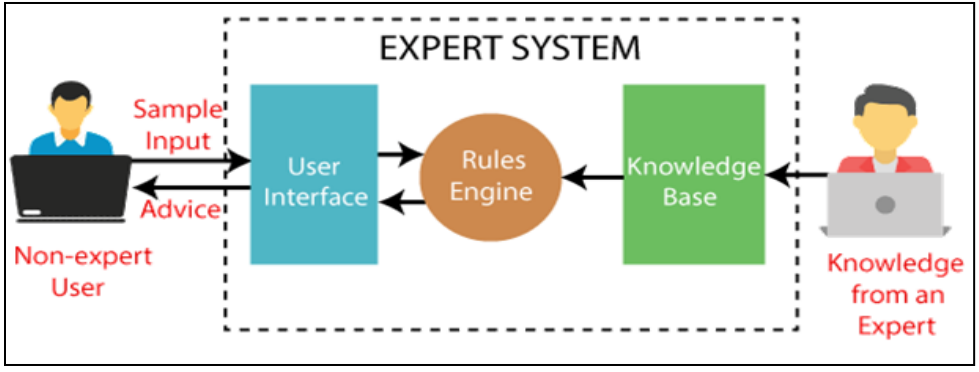
- Commonly referred to as "WebApps," represent a diverse category of network-centric software encompassing a broad range of applications.

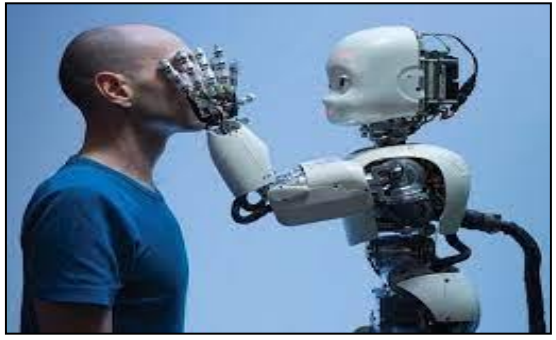
- WebApps are becoming advanced computing environments that not only offer independent features, computing capabilities, and content to users but are also seamlessly integrated with corporate databases and business applications.



(vii) Artificial Intelligence Software

- Artificial intelligence software employs nonnumerical algorithms to address complex problems that cannot be easily computed or directly analyzed.
- Applications within this area include
  - Robotics
  - Expert systems
  - Artificial neural networks
  - Theorem proving
  - Game playing
  - Pattern recognition (Image and Voice)





3. Legacy Software

- Legacy software are older programs that are developed decades ago.
- The quality of legacy software is poor because it has inflexible design, complex code, poor and nonexistent documentation, test cases and results that are not achieved.
- As time passes legacy systems change due to following reasons:
  - The software must be ***adapted*** to meet the needs of new computing environment or technology.
  - The software must be ***enhanced*** to implement new business requirements.
  - The software must be ***extended*** to make it interoperable with more modern systems or database.
  - The software must be ***rearchitected*** to make it viable within a network environment.

Software Engineering

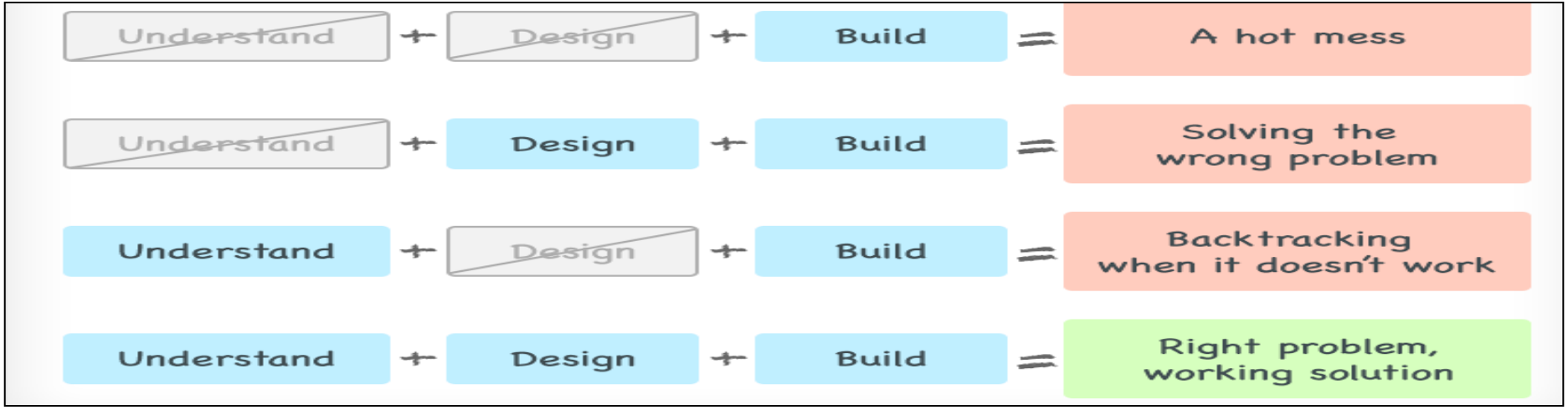
- **Software :**
  - The software is a ***collection of integrated programs***.
  - Computer programs and related documentation such as requirements, design models and user manuals.
- **Engineering:**
  - Engineering is the ***application of scientific and practical knowledge*** to invent, design, build, maintain, and improve frameworks, processes, etc.
- **Software Engineering**
  - Software Engineering is an ***engineering branch*** related to the ***evolution of software product*** using ***well defined scientific principles***, ***techniques***, and ***procedures***.
  - The result of software engineering is an ***effective and reliable software product***.

The **IEEE** has developed a more comprehensive definition when it states:

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

Building software ready for 21st century demands requires accepting certain fundamental truths. To construct software that meets modern complexities, key realities must be recognized:

- ***Understand the problem before you build a solution***
  - Software is now essential in everything we do.
  - People care about what apps or systems can do.
  - When creating something new, many people share different ideas on what it should do.
  - This can be confusing, so it's important to fully understand the problem before building a solution.



- ***Design is a key software engineering activity***
  - Over time, our needs from technology have become more complex.
  - Now, large teams work together to create software.
  - Software that used to run only on computers is now in things like phones, medical devices, and even weapons.
  - Since these systems are so complicated, we need to ***carefully design them to ensure all the parts work smoothly together***.

- ***Both quality and maintainability are an outgrowth of good design***
  - **People**, **businesses**, and **governments** use software a lot for important decisions and everyday tasks.
  - If the software doesn't work well, it can cause small problems or even really big disasters.
  - So, it's important for the ***software to be really good and reliable***.

- Software engineering is a ***layered technology.***
- Any engineering approach including software engineering must rest on an organizational commitment to quality.
- The foundation for software engineering is the process layer.

- The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software.

- **Process:**
  - Defines a framework that must be established for ***effective delivery of software engineering technology***.
  - The software process helps manage software projects effectively.
  - It guides how technical methods are used, ensures work products like
    - ***models and reports are created***
    - ***sets milestones***
    - ***maintains quality***
    - ***handles changes efficiently.***

- Software engineering **methods** are like ***guides that show us how to build software.***
- Methods encompass a broad array of tasks that include
  - ***Communication***
  - ***Requirements Analysis***
  - ***Design Modeling***
  - ***Program Construction***
  - ***Testing***
  - ***Support***

- Software engineering uses basic principles to guide how software is designed and built.
- These principles include creating models and descriptions to understand and plan the system before developing it.
- Software engineering **tools** provide ***automated or semiautomated support for the process and the methods***.
- When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called **computer-aided software engineering**, is established.

SOFTWARE PROCESS MODEL

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

Activities

- An activity focuses on ***achieving a general goal***, like communicating with stakeholders.
- It applies to all projects, no matter the domain, size, complexity, or how strictly software engineering principles are followed.

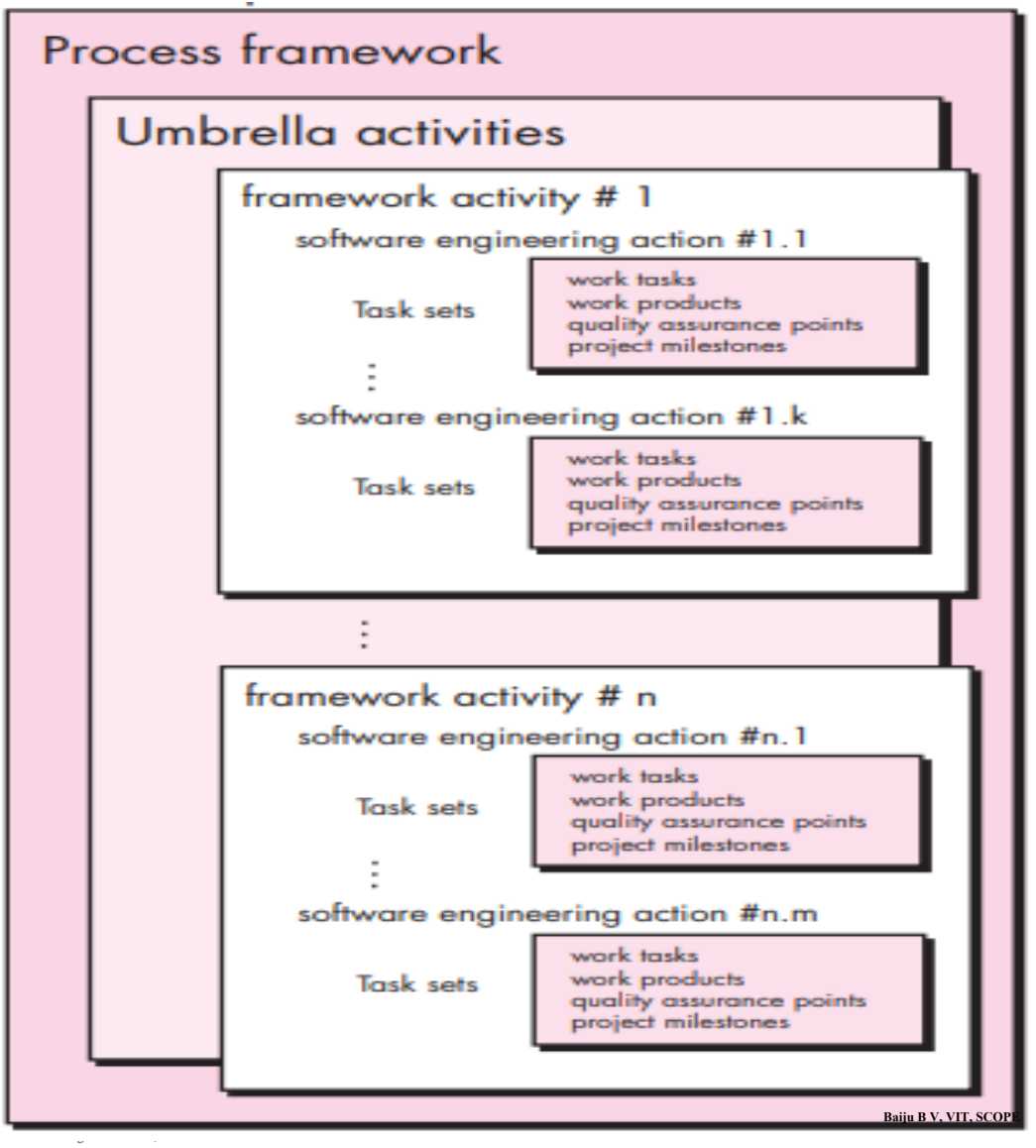
Action

- An action (e.g., architectural design) encompasses a set of tasks that ***produce a major work product*** (e.g., an architectural design model).

Task

- A task focuses on a ***small, but well-defined objective*** (e.g., conducting a unit test) that produces a tangible outcome.

- A **software process** means to ***develop quality products*** by using technical and management rules.
- A **generic process model** is an ***description of the software development process***.
- It is used in most software models since it provides a base for them.
- A **process framework** sets the groundwork for an entire software engineering process by defining a few core activities that are relevant to all software projects, regardless of their size or complexity.



- A **generic process framework** for software engineering encompasses five activities:

Communication

- Software development starts with the communication between ***customer and developer***.
- This helps ***understand their project goals and collect requirements*** to define software features and functions.

Planning

- Planning is like ***creating a map*** that guides the team throughout the complex journey.
- This "map," known as a **software project plan**, outlines technical tasks, potential risks, needed resources, expected work products, and the schedule.

Modeling

- Develop a ***practical model*** to get better understanding of the project.
- Consists of complete requirement analysis and design the project like algorithm, flowchart.
- If changes are required, we implement them in this step.

Construction

- This involves ***creating code***, either manually or using automation, and then testing to find and fix any errors in the code.

Deployment

- The customer ***receives the software***, whether it's complete or in parts, assesses it, and gives feedback based on the evaluation.

Developing a Food Delivery App

Communication:

- The customer (restaurant owner) talks to the developer to explain the goal: an app for customers to order food online and track delivery.

Requirement Analysis:

- The developer gathers details like:
  - ✓ Customers need to browse the menu, place orders, and make payments.
  - ✓ The restaurant needs to manage orders and update delivery status.
- These requirements are documented clearly.

Planning:

- The developer creates a "map" or plan:
  Tasks: Build the menu page, order system, and payment gateway.
  Risks: Payment gateway errors or slow app performance.
  Resources: Designers, developers, and testing tools.
  Timeline: 3 months for development.

Design:

- The developer creates a **flowchart** for the app:
  Start: Customer logs in → Browse menu → Add items to cart → Checkout → Payment → Order confirmation.

Implementation:

- The developer writes the code for app features like login, menu display, and order tracking.
- Automated tools are used for testing, and errors (like incorrect pricing or crashes) are fixed.

Testing and Feedback:

- The customer receives the app for testing.
- Feedback: “Add a feature to schedule delivery times.”
- The developer makes the change.

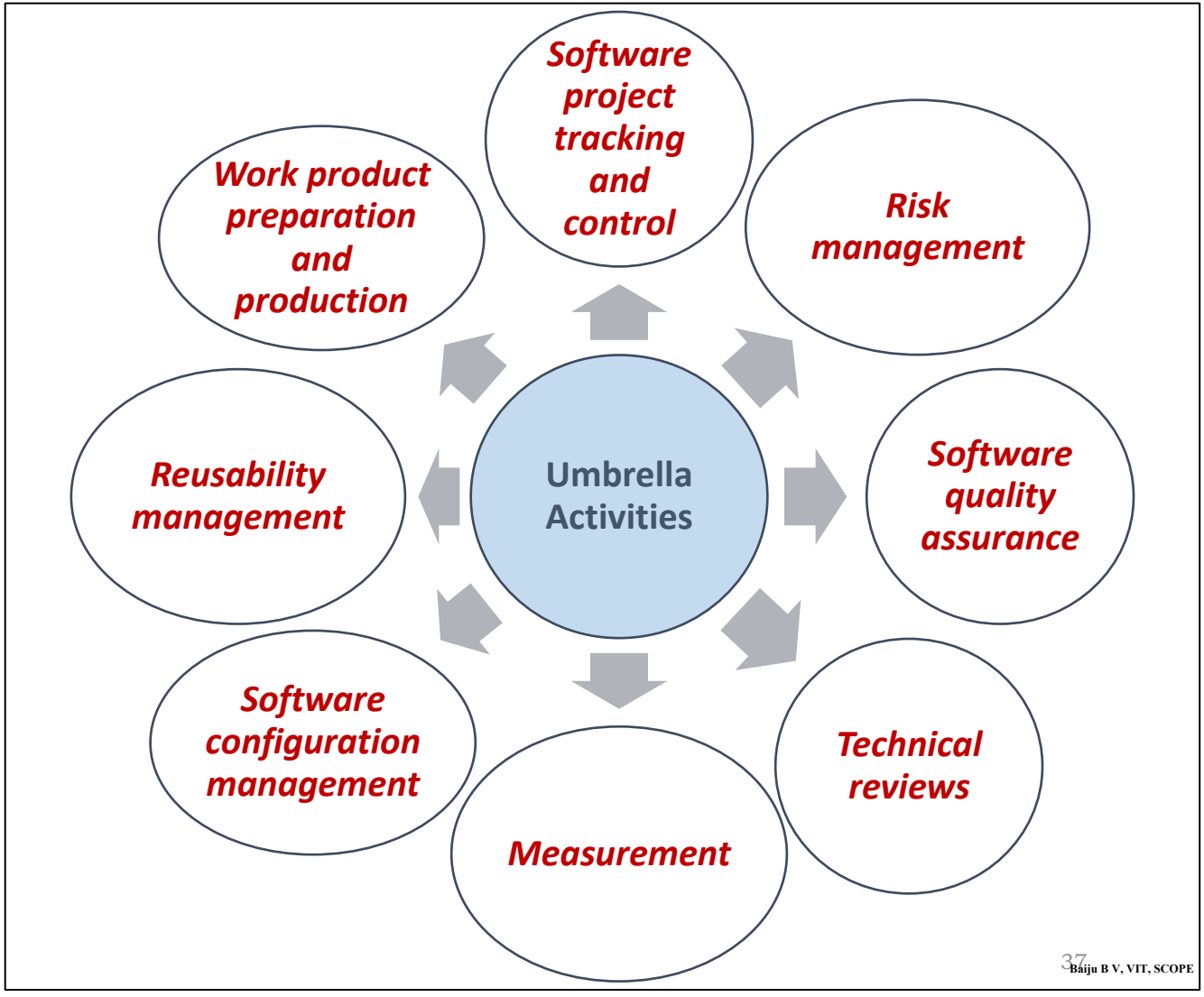
Deployment:

- The final app is delivered to the restaurant, ready for customers to use.

Umbrella Activities

- Software engineering process framework activities are complemented by a number of umbrella activities.

Typical umbrella activities



a. Software Project Tracking and Control : Allows the software team to assess progress against the project plan and take any necessary action to maintain the schedule.

b. Risk Management : Assesses risks that may affect the outcome of the project or the quality of the product.

c. Software Quality Assurance : Defines and conducts the activities required to ensure software quality.

d. Technical Reviews : Assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next activity.

e. Measurement : Specifies and gathers data on the process, project, and product to help the team deliver software that meets stakeholders' needs.

f. Software Configuration Management : Manages the effects of change throughout the software process.

g. Reusability Management : Defines criteria for work product reuse and establishes mechanisms to achieve reusable components.

h. Work Product Preparation and Production : Encompasses the activities required to create work products such as models, documents, logs, forms, and lists.

Framework Activities

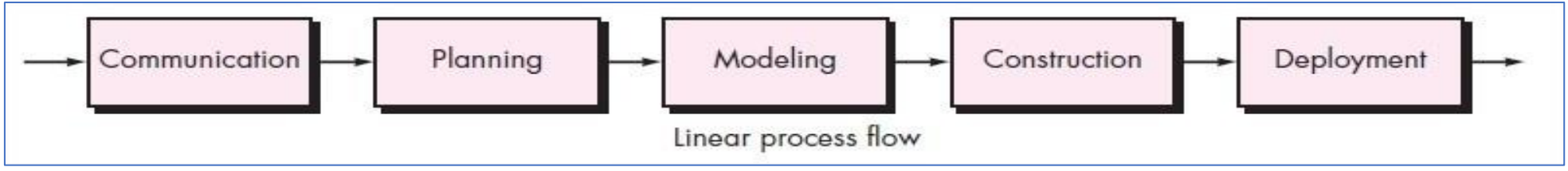
- ***Consider communication activity for small project***
  - ➢Making a phone call with stakeholder
  - ➢Discuss requirements and note them down
  - ➢Organize requirements
  - ➢Mail stakeholder for review and approval
- ***Consider communication activity for large project***
  - ➢Arrange live meeting
  - ➢Complete feasibility study
  - ➢Requirement analysis
  - ➢Specification documents

Task Set

- Task set is the actual work done to achieve an objective of each engineering module.
- **Consider communication activity task set**
  - ➢ Prepare a list of stakeholder of the project
  - ➢ Organize a meeting for stakeholder
  - ➢ Discuss requirements
  - ➢ Finalize requirement list
  - ➢ Make a list of issues raised

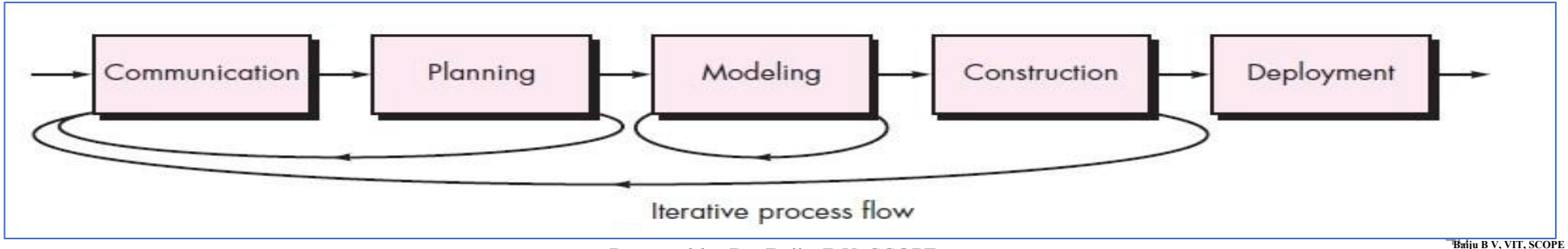
- **Process flow** describes how the framework activities and the actions and tasks that occur within each framework activity are organized with respect to sequence and time.

(i) Linear Process Flow : A *linear process flow* ***executes each of the five framework activities in sequence***, beginning with communication and ending with deployment



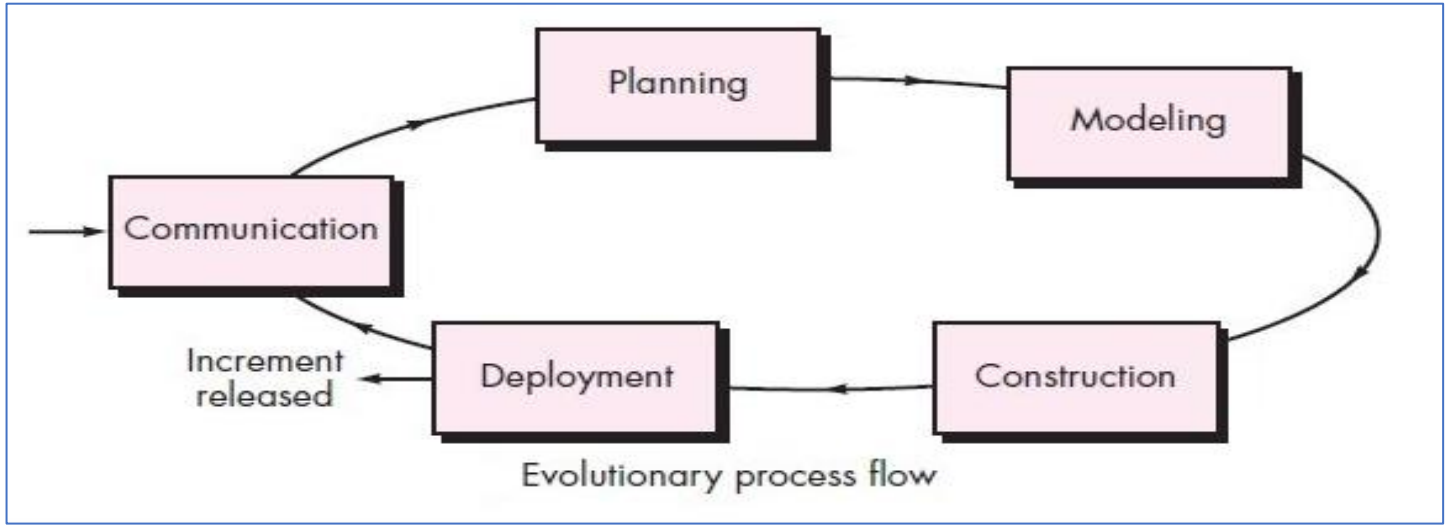
Linear process flow

(ii) Iterative Process Flow : An *iterative process flow* ***repeats one or more of the activities*** before proceeding to the next

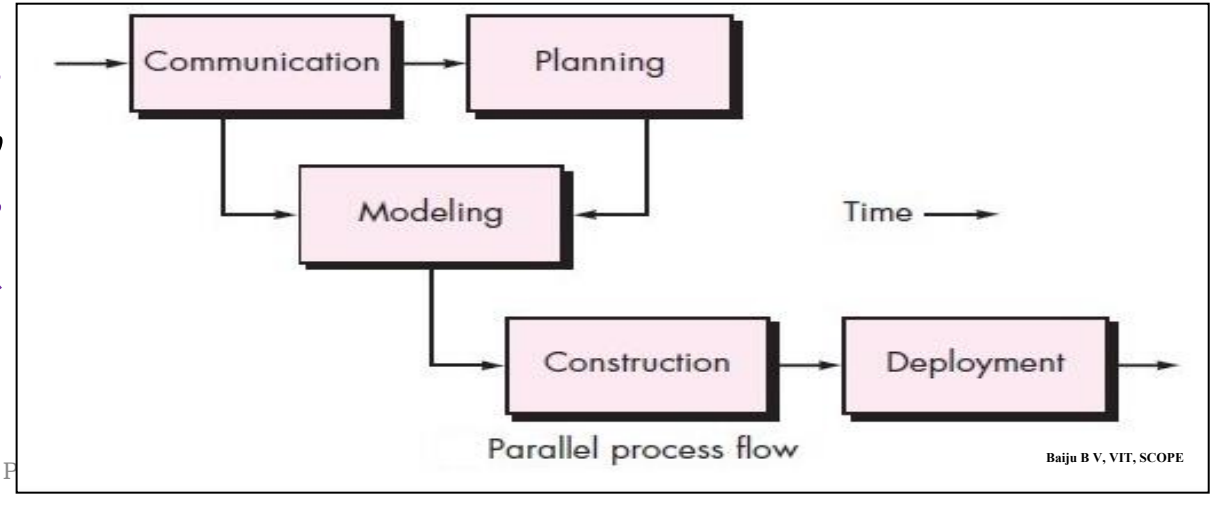


Iterative process flow

(iii) Evolutionary Process Flow : An *evolutionary process flow* ***executes the activities in a "circular" manner.***



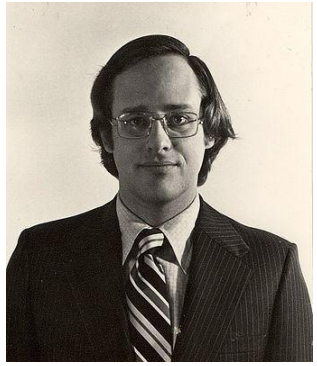
(iv) Parallel Process Flow : A *parallel process flow* ***executes one or more activities in parallel with other activities.***

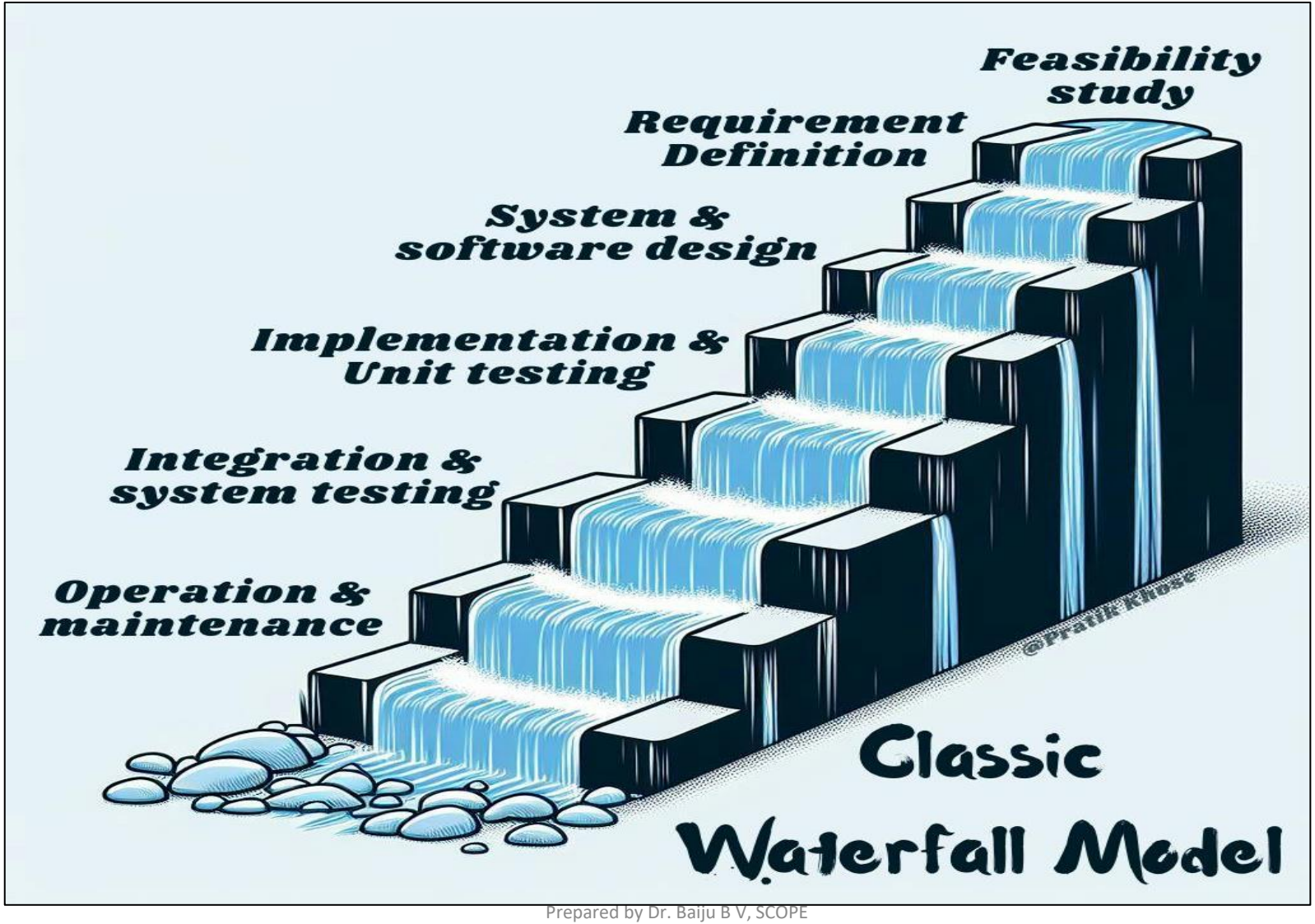


A **Software Process Model** (sometimes called a Software Development Life Cycle or SDLC model) is a simplified representation of a software process.

1. Waterfall Model

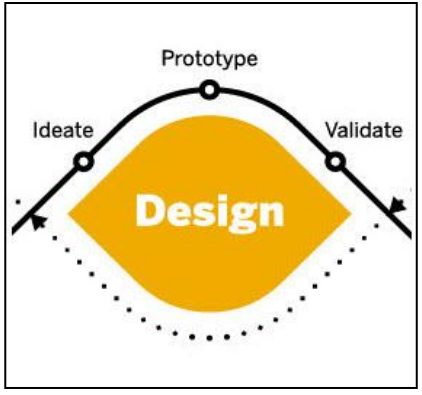
- The first published model of the software development process was derived from engineering process models used in ***large military systems*** engineering
- The waterfall model, sometimes called the ***linear sequential model***, suggests a systematic, sequential approach to software development.
- ***Winston Royce*** introduced the Waterfall Model in 1970.
- You plan and schedule all of the process activities before starting software development.



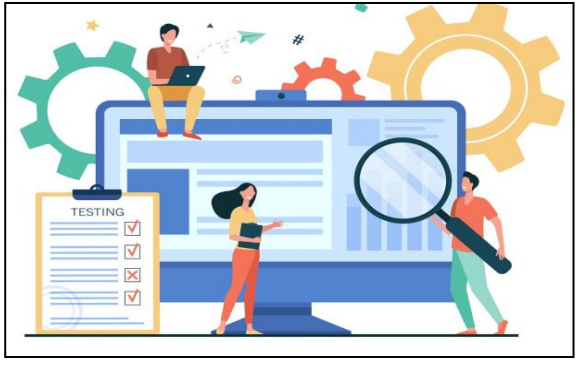

Feasibility study
Requirement Definition
System & software design
Implementation & Unit testing
Integration & system testing
Operation & maintenance
@Pratik Khose
Classic
Waterfall Model

- The stages of the waterfall model directly reflect the fundamental software development activities:

Stages	Activities performed in each stage
Requirements Analysis and Definition	• The system's services, constraints, and goals are established by consultation with system users. • They are then defined in detail and serve as a system specification. • ***Software Requirement Specification*** (SRS) document is created. • SRS serve as a ***contract*** between the development team and customers
System and Software Design	• The systems design process allocates the requirements to either hardware or software systems. • Software design involves ***identifying and describing*** the fundamental software system abstractions and their relationships. • ***High level design*** includes Algorithms, Flowcharts, Decision tree etc. • ***Low level design*** includes rough paper design, user interface components



Stages	Activities performed in each stage
Implementation and Unit testing	• Software design is realized as a set of programs or program units. • ***Unit testing*** involves verifying that each unit meets its specification.
Integration and System testing	• The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. • After testing, the software system is ***delivered to the customer***.
Operation and Maintenance	• The system is installed and put into practical use. • Maintenance involves ***correcting errors*** that were not discovered in earlier stages of the life cycle, improving the implementation of system units, and ***enhancing the system's services*** as new requirements are discovered.



Waterfall Methodology can be used when:

- ***Requirements are not changing frequently***
- ***Application is not complicated and big***
- ***Project is short***
- ***Requirement is clear***
- ***Environment is stable***
- ***Technology and tools used are not dynamic and is stable***
- ***Resources are available and trained***

A software development company has been tasked with creating a hospital management system for a client. The client has provided a detailed document of requirements and insists on no changes once the development begins. The project involves integrating various modules like patient registration, billing, inventory management, and report generation, each depending on the previous one to ensure the system works cohesively.

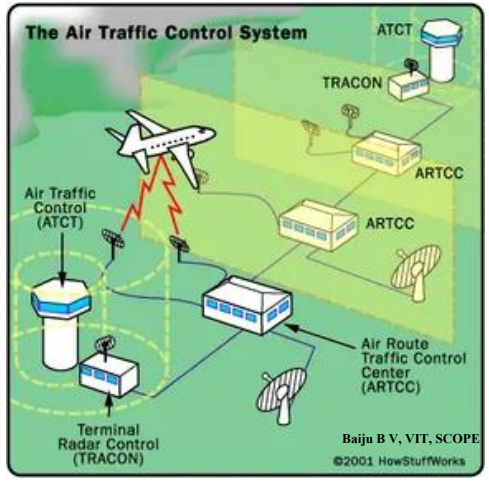
1. Identify the reasons why the Waterfall Model is appropriate for this project.
2. List the challenges the team might face if the client requests a change in the requirements after the design phase is complete.
3. Describe how the team should approach testing to ensure all modules are correctly integrated and function as expected.

1
Well-defined Requirements
Linear Dependencies
No Expected Changes:
Complex System Integration:

2.
High Cost of Changes
Disruption in Schedule
Impact on Dependencies
Testing Overheads

3.
Unit Testing
Integration Testing
System Testing
Regression Testing
Acceptance Testing

- The Waterfall software engineering model can be used for the following **scenarios**:
  - ***Embedded systems*** where the software has to interface with hardware systems.
  - Projects are large, complex, and require a well-structured, sequential, linear approach. This helps projects to be developed on time and within budget.
  - ***Safety-critical systems*** such as aerospace and medical systems.
  - ***Government projects***, defense – security project.
  - Projects with clear and stable requirements documentation.
  - ***Critical systems*** where there is a need for extensive safety and security analysis of the software specification and design.



ArtOfTesting

ADVANTAGES OF THE WATERFALL MODEL

Remarkable benefits of this classical model

All processes, actions, and results are well documented.

It is easy to understand and implement.

There are specific deliverables in each phase of the life cycle.

All the activities to be performed in each phase are clearly defined.

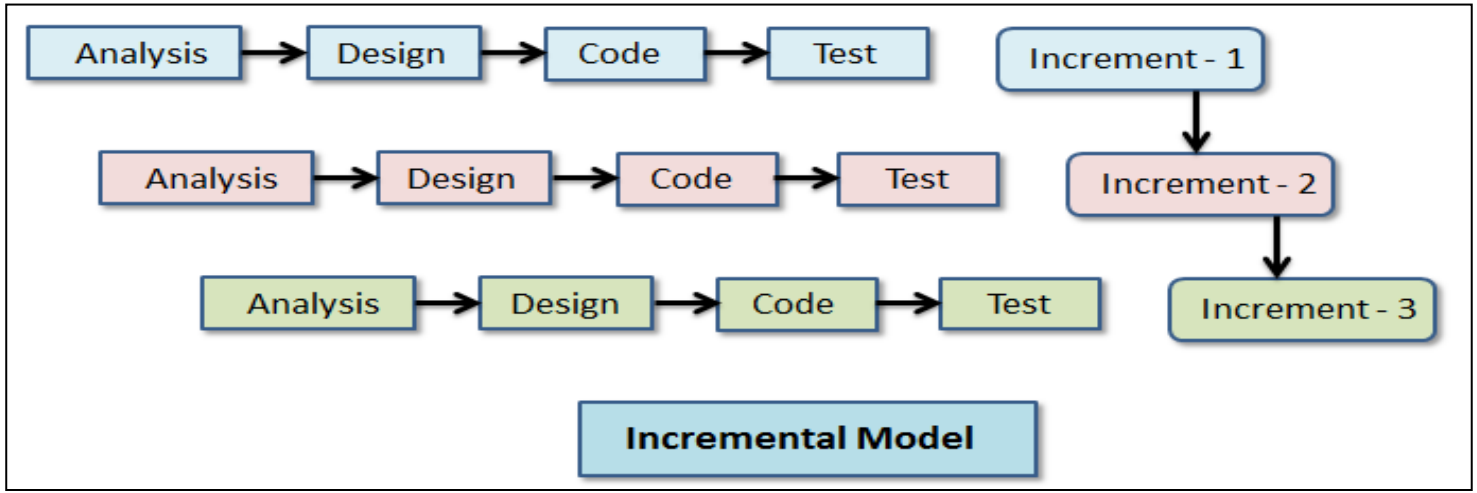
It is perfectly suitable for projects where all the requirements are predefined and understood clearly.

The release date and the final cost are already estimated before the development begins.

It is easier to assign tasks to different team members.

Incremental Model

- Incremental Model is a process of software development where ***requirements are broken down into multiple standalone modules*** of software development cycle.
- In this model, each module goes through the requirements, design, implementation and testing phases.
- Every subsequent release of the module adds function to the previous release.
- The process continues until the complete system achieved.



Development of an Online Banking System

Project Requirements:

The online banking system will have the following features:

- ***User Login and Account Management***
- ***Fund Transfer Between Accounts***
- ***Bill Payment System***
- ***Account Statement Generation***

Increment 1: User Login and Account Management

Features Developed:

- User authentication (login/logout)
- View account details (balance, account type)
- Create and manage user profiles

Output:

- A functional system where users can log in and manage their accounts.

Increment 2: Fund Transfer Between Accounts

Features Developed:

- Transfer funds between the user's accounts (e.g., savings to checking)
- Transfer funds to other users' accounts
- View transaction status

Output:

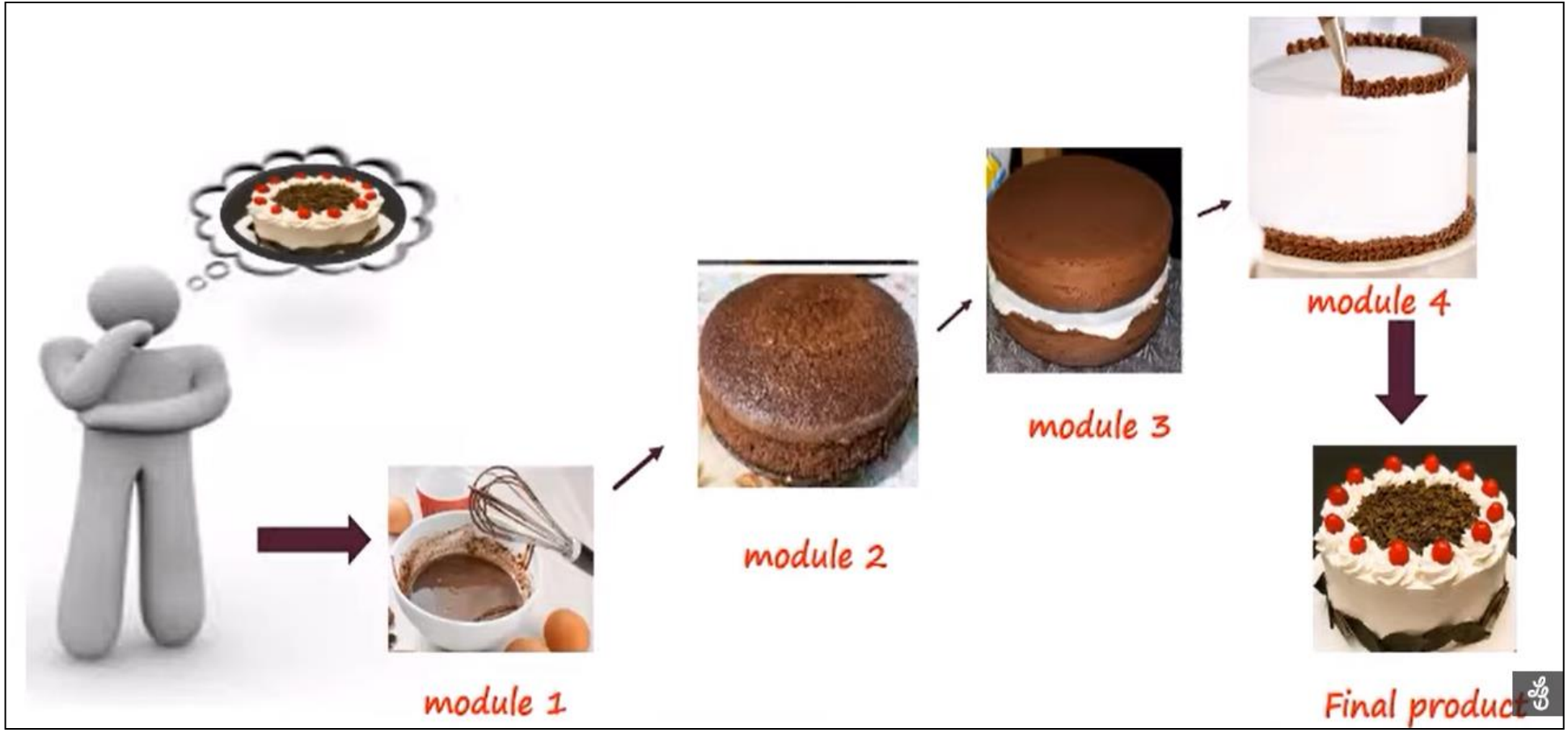
- Users can now log in and transfer funds.
- The previous features (Increment 1) remain operational.

Creating an Educational Learning Platform

Question: ***Your company is creating an online educational platform for schools. How would you use the Incremental Model to handle diverse requirements and ensure the platform's scalability?***

Start with basic functionalities like user registration and course creation. Deliver this increment to gather initial feedback. Next, add features for quizzes and assessments, followed by interactive elements like discussion forums and live classes. Each increment allows for adjustments based on user feedback, ensuring the platform meets the evolving needs of educators and students.

First Increment:	User Registration and Course Creation
Second Increment:	Quizzes and Assessments
Third Increment:	Discussion Forums
Fourth Increment:	Live Classes


module 1
module 2
module 3
module 4
Final product

Characteristics of an Incremental module includes

Incremental Phases	Description
Requirement Analysis	• Complete analysis is performed on the requirement (functional and non-functional). • These requirements serve as a foundation for the subsequent phases.
Design	• Once the requirement for this particular increment is understood and clear then design will be drafted on how to implement and archive this requirement. • The design is often ***divided into smaller segments*** to ensure a focused and organized development process.
Code	• Now the coding is performed in accordance to achieve the purpose of the requirements. • All the coding standards will be followed without any defaults and unnecessary hard codes.
Test	• This is the last in the incremental phase where aggressive testing is performed on the developed code and defects are reported and resolved.

When do we Use the Incremental Model?

- The Incremental Model is used in software development when

Changing Requirements	• Requirements are likely to change during development, and it's better to address these changes incrementally.
Mitigating Risks	• High-risk projects benefit from incremental development as it allows for early identification and management of potential issues.
Early Feedback	• Frequent iterations enable stakeholders to provide feedback early in the development process, improving the final product's alignment with expectations.
Complex Systems	• Building large and complex systems can be overwhelming; incremental development breaks it into manageable parts.
Short Time-to-Market	• Incremental releases allow for faster delivery of functional components, providing quick value to users.
Parallel Development	• Multiple teams can work concurrently on different increments, enhancing productivity and collaboration.

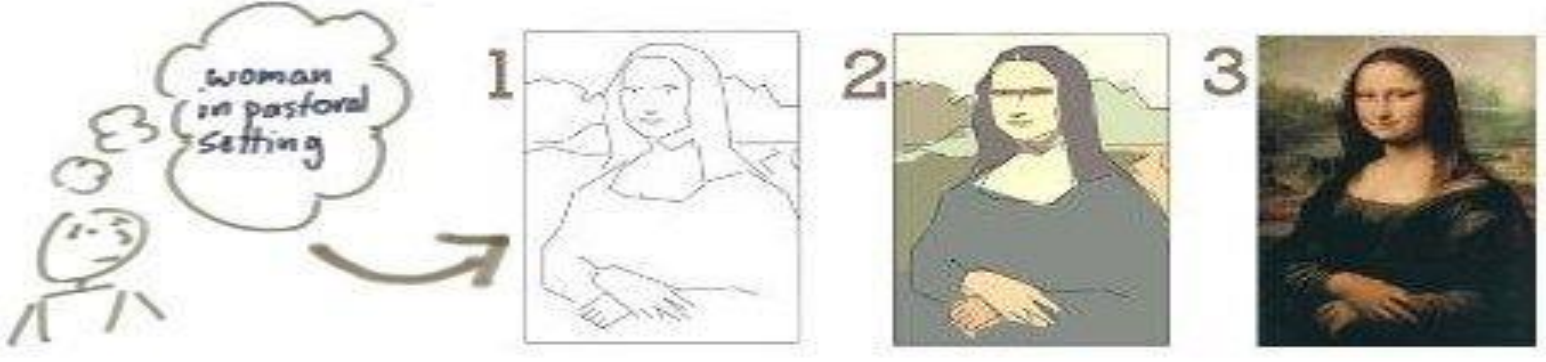
Budget Constraints	• Incremental development allows for distributing costs across iterations, making it more manageable for limited budgets.
Evolving Solutions	• When the exact solution isn't clear from the outset, an incremental approach allows for refining the solution as the project progresses.
Customer Collaboration	• Close interaction with customers is required to ensure the product meets their needs; incremental models facilitate this.
Flexibility	• Projects where requirements are uncertain or evolve benefit from the model's adaptability to changes.

Examples of the incremental model

- ❖ Linux kernel
- ❖ Web development
- ❖ E-commerce website

Incremental

Iterative



ADVANTAGE

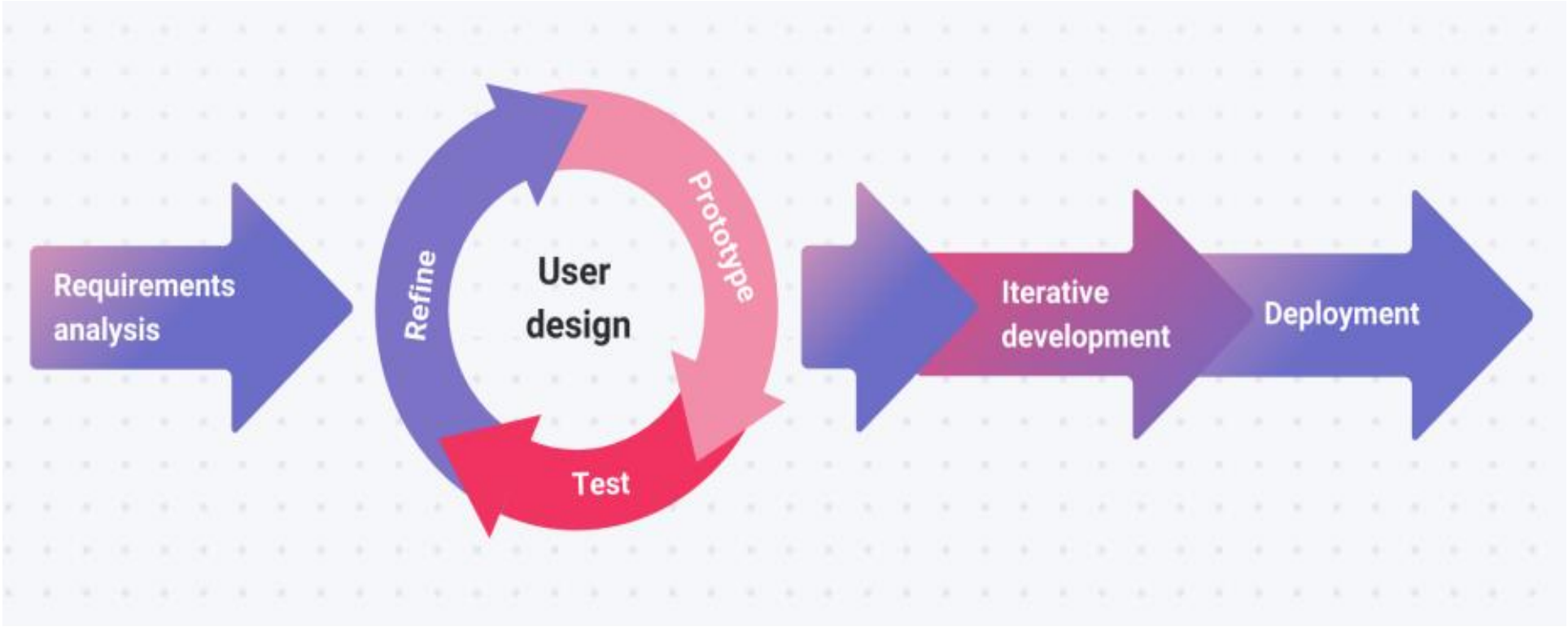
- Generates working software quickly and early during the software life cycle.
- More flexible - less costly to change scope and requirements
- Easier to test and debug during a small iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration
- Each iteration is an easily managed milestone.

DISADVANTAGE

- Each phase of an iteration is rigid and does not overlap with each other.
- Problem may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle
- It requires good planning for designing
- Rectifying a problem in one unit requires correction in all the units and consumes a lot of time

RAD Model

- **RAD Model** or ***Rapid Application Development model*** is a software development process ***based on prototyping without any specific planning***.
- In RAD model, there is ***less attention paid to the planning*** and ***more priority is given to the development tasks***.
- Development speed is achieved through close collaboration between developers and users.
- It targets at developing software in a short span of time.
- The key objectives of the RAD model are ***high quality***, ***high speed***, and ***low cost.***
- Rapid application development aims
  - **to gather customer needs through workshops or focus teams**
  - **initial testing of iteration**
  - **using iterative model**
  - **reusing existing designs**
  - **continuous integration**
  - **faster delivery.**

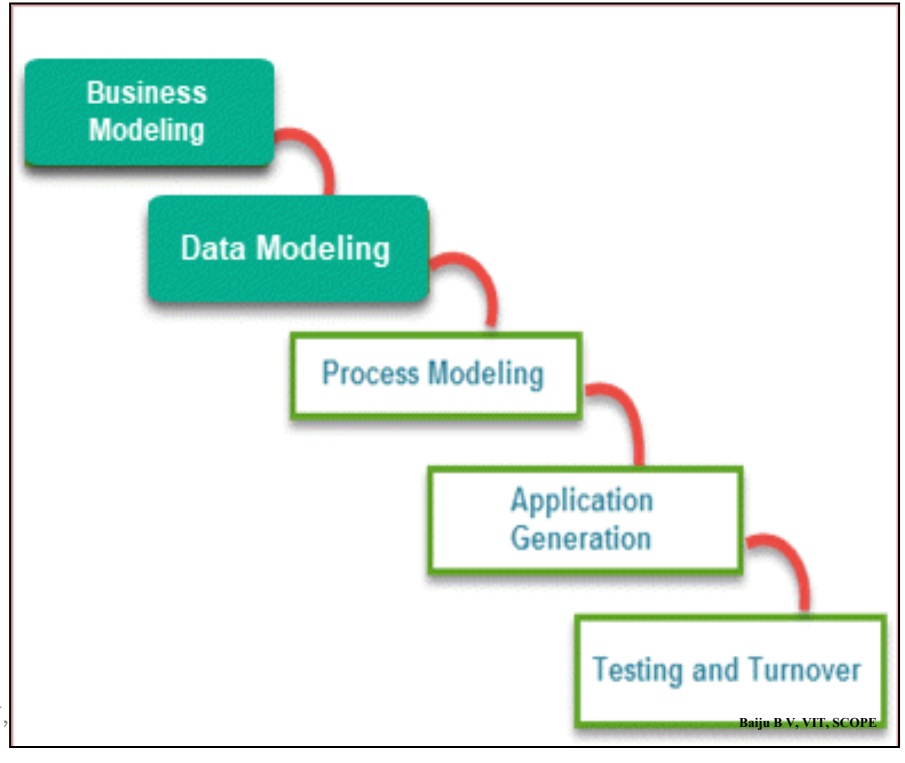

Requirements analysis
Refine
User design
Prototype
Test
Iterative development
Deployment

Uber

- The various phases of RAD are as follows:

(i) Business Modeling

- Understand the ***business requirements*** and ***define the scope*** of the project.
- The information flow among business functions is defined by answering questions like
  - ***What data drives the business process***
  - ***What data is generated***
  - ***Who generates it***
  - ***Where does the information go***
  - ***Who process it***



Business Context:

A hospital needs to streamline its operations and improve efficiency. The system is being developed to support various functions like patient registration, billing, inventory management, and reporting, all of which need to be aligned with the hospital's goals of improving patient care, optimizing resources, and maintaining compliance with regulations.

What data drives the business process?

- **Patient Information** (e.g., name, age, contact details)
- **Medical Records** (e.g., diagnosis, treatments, prescriptions)

What data is generated?

- **Patient Registration Data** (e.g., new patients' details, historical medical data)
- **Treatment and Procedure Data** (e.g., prescribed treatments, patient vitals, test results)

Who generates it?

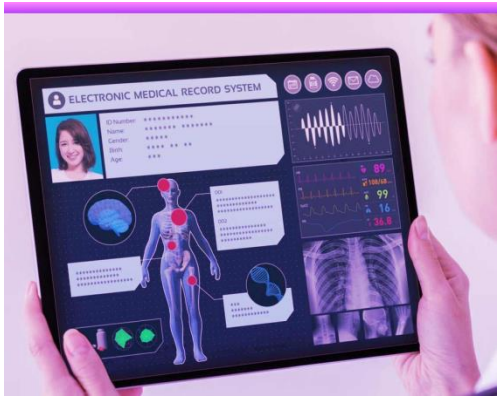
- **Patient Registration:** Front desk staff or receptionists input patient data into the system.
- **Medical Records:** Doctors and nurses input patient health information, diagnoses, and prescribed treatments.

Where does the information go?

- **Patient Information:** Stored in a central patient database, accessible by healthcare providers for treatment and medical history.
- **Medical Records:** Sent to the Electronic Health Record (EHR) system, accessible by doctors, nurses, and other medical staff.

Who processes it?

- **Patient Information:** Processed by doctors, nurses, and medical staff for diagnosis and treatment.
- **Medical Records:** Processed by healthcare providers to monitor patient progress and adjust treatments.



(ii) Data Modelling:

- In this phase, developers ***analyze the data needed for the project and organize it into data models.***
- This includes defining ***data objects***, their ***relationships***, and ***data management rules***.
- The data modelling phase is essential for ensuring that the application can efficiently manage and process the required data.

Patient Object:

- Includes attributes like name, age, address, phone number, medical history, and insurance details.

Data Object: Patient

- ***Attributes***: name, age, address, contact details, medical history, insurance number
- ***Relationships***: One-to-many with Medical Record, one-to-one with Billing.

Data Integrity Rules:

- Ensuring data is consistent and valid (e.g., a patient's age must be a non-negative integer).

Medical Record Object:

- Contains attributes like diagnosis, prescribed treatments, test results, treatment date, and attending doctor.

Medical Records:

Data Object: MedicalRecord

- ***Attributes***: diagnosis, prescribed treatments, test results, doctor_id
- ***Relationships***: Many-to-one with Patient, one-to-many with Treatment

Data Access Rules:

- Defining who can access or modify specific data (e.g., only doctors can input diagnoses, while receptionists can update contact details).

(iii) Process Modelling:

- During the process modelling phase, developers ***design the software processes to manipulate and manage data***.
- This involves creating ***flowcharts***, ***process diagrams***, and ***pseudocode*** representing the various algorithms and functions required to achieve the desired functionality.
- Process modelling ensures that the application's ***logic is well-structured and efficient***.

Patient Registration Flowchart:

- Start
- Receptionist inputs patient details (name, age, contact)
- System checks if the patient is new or returning
- If new, the system creates a new patient ID and stores the data
- If returning, system retrieves existing patient details
- Store data in the central database
- End

Patient Registration Process Diagram:

Input: Patient details (from reception)

Process: Check if patient exists, create new entry or fetch existing data.

Output: Central patient database (patient ID, medical history)

Interaction: Data stored in the database, accessible by healthcare providers for treatment decisions

Medical Records Flowchart:

- Start
- Doctor inputs patient diagnosis and treatment details
- System checks if treatment is prescribed
- If prescribed, treatment data is stored in the EHR system
- Update patient record with any new test results
- Medical staff monitors patient progress and updates records
- End

Medical Records Process Diagram:

Input: Doctor's diagnosis, treatment details, test results

Process: Validate and update records in the EHR system

Output: Updated patient record (treatment history, test results)

Interaction: Doctors, nurses, and medical staff interact with the EHR system for real-time updates

(iv) Application Generation:

- In this stage, the software's ***actual coding*** and ***development*** occur.
- Developers use various ***tools***, ***languages***, and ***frameworks*** to build applications based on the data and process models created in the previous phases.
- Rapid construction techniques, such as code reuse and integrated development environments (IDEs), are employed to accelerate the development process.

Patient Registration Module:

- Developers code the form where front desk staff enter new patient details.
- Data entered by staff is validated (e.g., ensuring valid phone numbers), and upon successful registration, the data is stored in a secure database, accessible by doctors.
- **Programming Languages**: Java
- **Integrated Development Environments (IDEs)**: Visual Studio
- **Database Management Systems (DBMS)**: MySQL
- **Frameworks**: Spring (Java)

Medical Records Module:

- Developers integrate the backend functionality that stores treatment histories, diagnoses, prescriptions, and test results. They also code the interface where doctors and nurses can easily retrieve and update patient medical records.

(v) Testing & Turnover:

- The final phase involves rigorous application ***testing*** to identify and fix any issues or bugs.
- This includes ***unit testing***, ***integration testing***, and ***user acceptance testing*** (UAT).
- Once the application has passed all tests and received user approval, it is deployed to the production environment, and **user training** is provided as needed.

Unit Testing:

- ***Patient Registration Module*:** Test that the system correctly captures patient details (e.g., name, age, contact information) and stores them in the central database.

Integration Testing:

- ***Patient Registration and Medical Records Integration*:** Ensure that data entered during patient registration (name, contact info, etc.) is correctly linked to the medical records, and that healthcare providers can access this data seamlessly during treatment.

User Acceptance Testing (UAT):

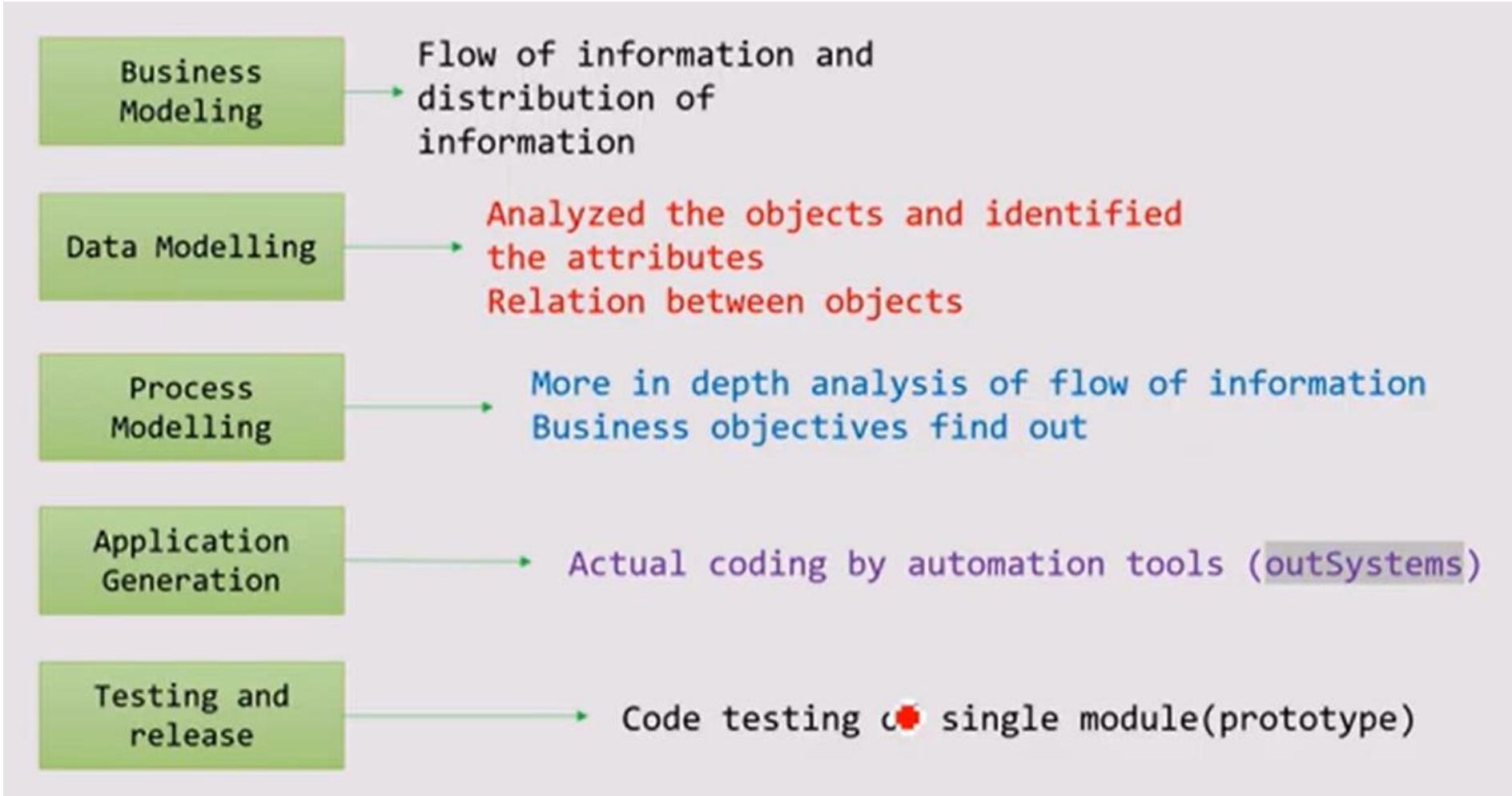
- ***Healthcare Providers (Doctors, Nurses):*** Test that they can easily access patient information and medical records to monitor and adjust treatments.

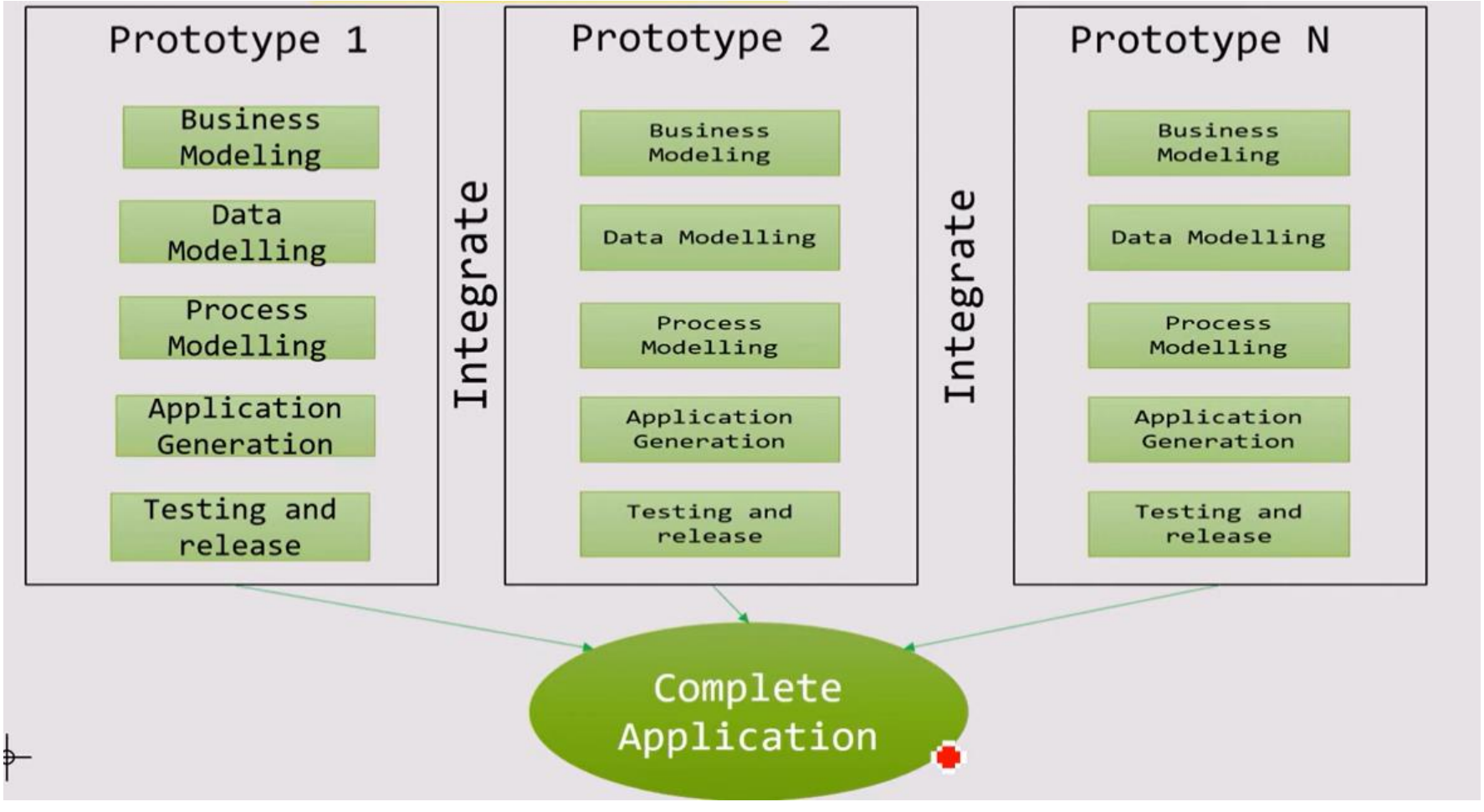
Unit Testing:

- ***Medical Records Module:*** Test that diagnosis, prescribed treatments, and patient vitals are correctly recorded and stored in the EHR system.
- **Integration Testing:**
- ***Patient Registration and Medical Records Integration*:** Ensure that data entered during patient registration (name, contact info, etc.) is correctly linked to the medical records, and that healthcare providers can access this data seamlessly during treatment.

User Acceptance Testing (UAT):

- ***Healthcare Providers (Doctors, Nurses):*** Test that they can easily access patient information and medical records to monitor and adjust treatments.


Business Modeling
Flow of information and distribution of information
Data Modelling
Analyzed the objects and identified the attributes
Relation between objects
Process Modelling
More in depth analysis of flow of information
Business objectives find out
Application Generation
Actual coding by automation tools (outSystems)
Testing and release
Code testing of single module(prototype)


Prototype 1
Business Modeling
Data Modelling
Process Modelling
Application Generation
Testing and release
Integrate
Prototype 2
Business Modeling
Data Modelling
Process Modelling
Application Generation
Testing and release
Integrate
Prototype N
Business Modeling
Data Modelling
Process Modelling
Application Generation
Testing and release
Complete Application

When to use the RAD Model:

- ***Tight Project Timelines:***
- ***Well-Defined Requirements:***
- ***Active User Involvement:***
- ***Small to Medium-Scale Projects:***
- ***Availability of Skilled Team Members:***
- ***Budget Flexibility:***
- ***Project with Modular Approach:***
- ***When Prototypes Can Be Developed Quickly:***

Real-life examples of RAD

- Examples of scenarios where the RAD model in software engineering could be used:
  - ✓**Mobile App Development**
  - ✓**E-Commerce Platform**
  - ✓**Event Management System**
  - ✓**Health Care Information System**
  - ✓**Online Learning Platform**

Advantages

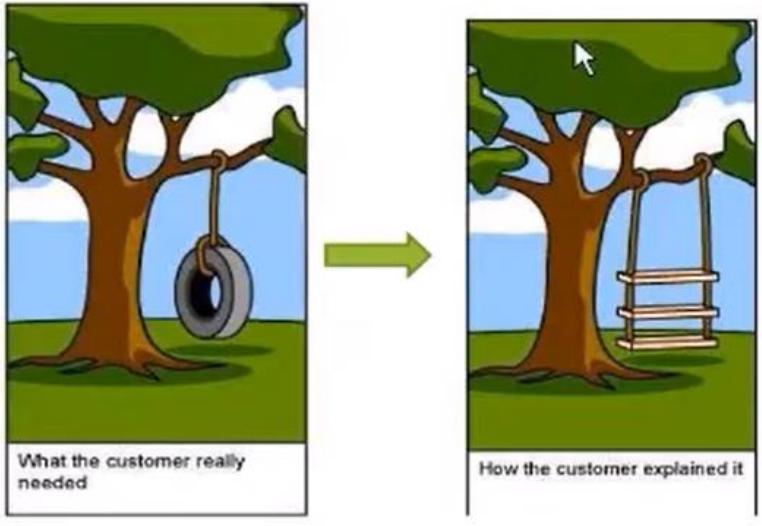
- Increases the reusability of components.
- Reduces the time of development.
- Progress of the project can be measured.
- Encourages the feedback of customers
- Productivity with a small team in a short time.

Disadvantages

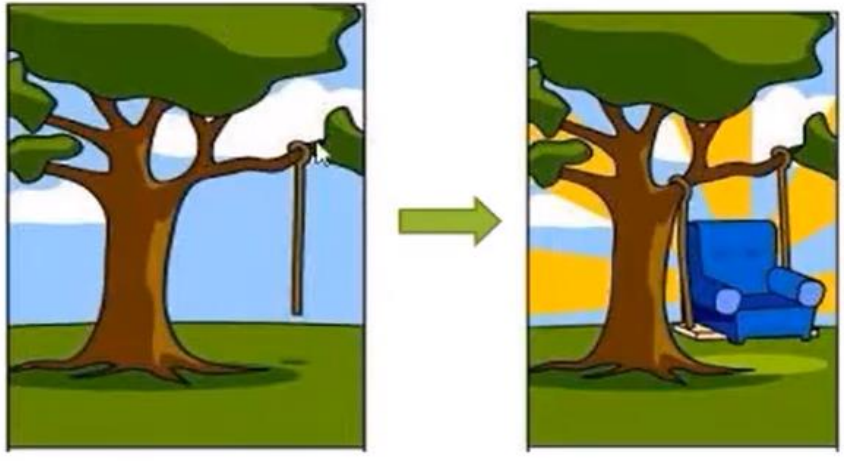
- Depends on a strong development team for recognizing the business requirements.
- Main requirement is a highly skilled development team.
- It is highly dependent on modeling skills.
- The management is more complex.
- Customer involvement is required throughout SDLC.
- This is mostly for projects which require a short time.

Evolutionary Models

Customer side

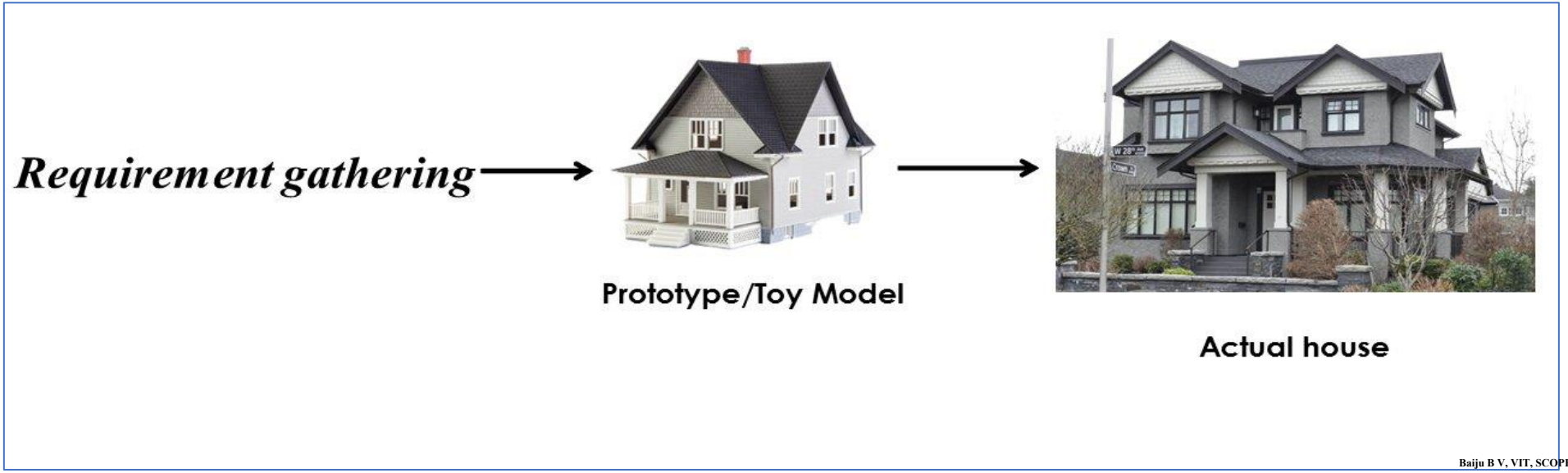


Developer side



PROTOTYPE MODEL

- A **prototype** means a ***preliminary model*** of anything which gives us a rough idea about the basic functionalities that the real model would have.
- It is not a complete product it is just function replica or toy implementation of any idea, software or system.
- It is applied when ***customers do not know the exact project requirements***.



- The prototype may include only the essential design elements and features that are necessary for visualizing and demonstrating the fundamental capabilities of the product.
- Prototypes are used by the development team in collaboration with customers and stakeholders to test the product's design, architecture, and functionality.
- A software prototype can be used in a software development process to help anticipate changes that may be required:

1. Requirements Engineering Process	• A prototype can help with the elicitation and validation of system requirements.
2. System Design Process	• Prototype can be used to explore software solutions and in the development of a user interface for the system.

Types of Prototypes

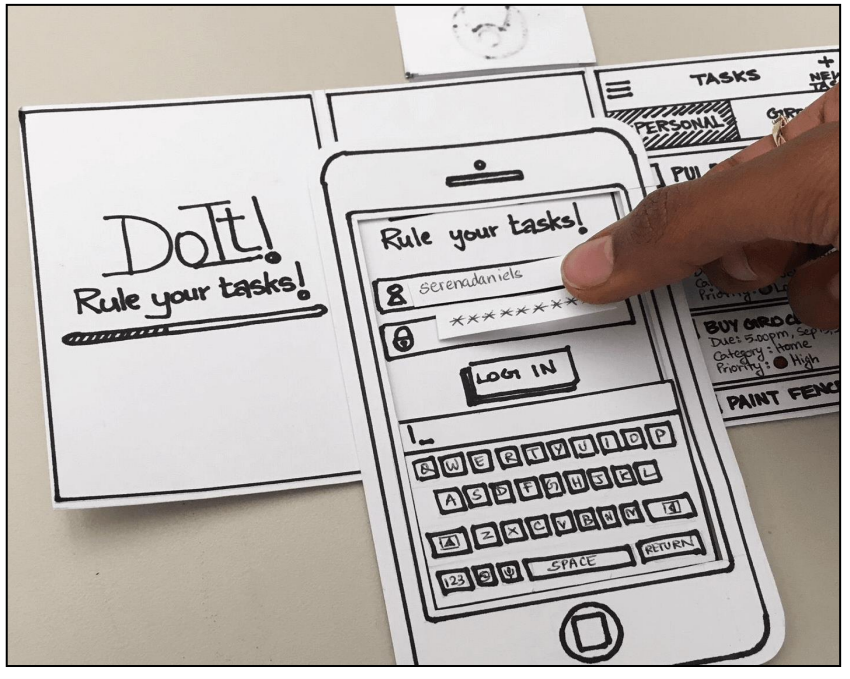
- Prototypes can be classified by the method of creation:

(i) Paper Prototype:

- The simplest and fastest to create, requiring only pen and paper.
- It's great for initial versions of prototypes.
- Paper prototypes show the ***structure and design*** of the product, as well as its ***basic functional elements***.

(ii) Digital Prototype:

- Created in specialized software and services for prototyping.
- Digital prototypes accurately show product functionality and may contain additional elements, such as ***animations*** and ***interactive components***.

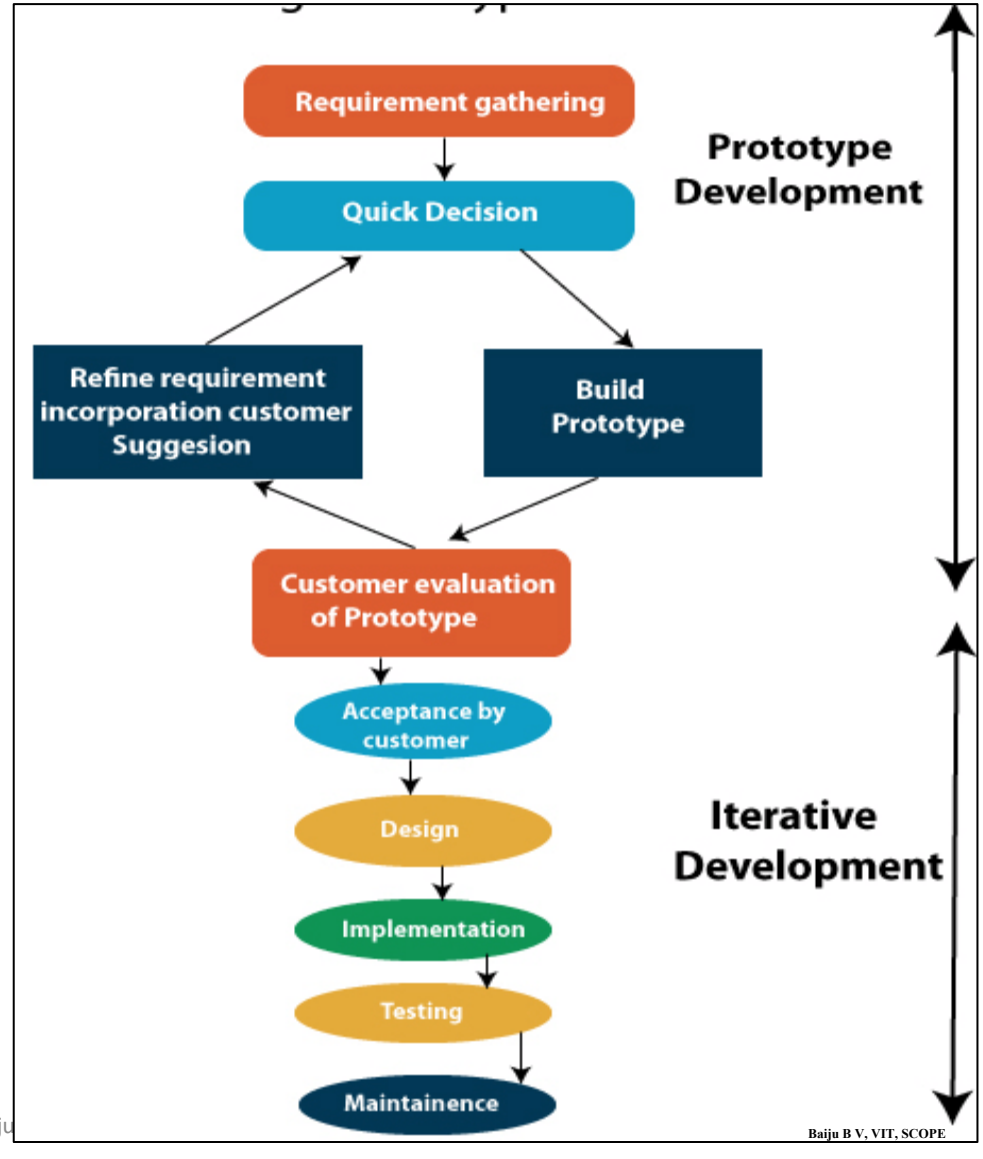


Step 1: Requirements Gathering

- A prototyping model starts with requirement analysis.
- In this phase, the requirements of the system are defined in detail.
- During the process, the ***users of the system are interviewed*** to know what is their expectation from the system.

Step 2: Quick Design

- The second phase is a preliminary design or a quick design.
- In this stage, a simple design of the system is created.
- However, it is not a complete design.
- It ***gives a brief idea of the system*** to the user.
- The quick design helps in developing the prototype.



Step 3: Build a Prototype

- An actual prototype is designed based on the information gathered from quick design.
- It is a ***small working model*** of the required system.

Step 4: Initial user evaluation

- The proposed system is presented to the client for an initial evaluation.
- It helps to find out the ***strength and weakness of the working model***.
- **Comment** and **suggestion** are collected from the customer and provided to the developer.

Step 5: Refining prototype

- If the user is not happy with the current prototype, you need to ***refine the prototype according to the user's feedback and suggestion***s.
- This phase will not over until all the requirements specified by the user are met.
- Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

Step 6: Implement Product and Maintain

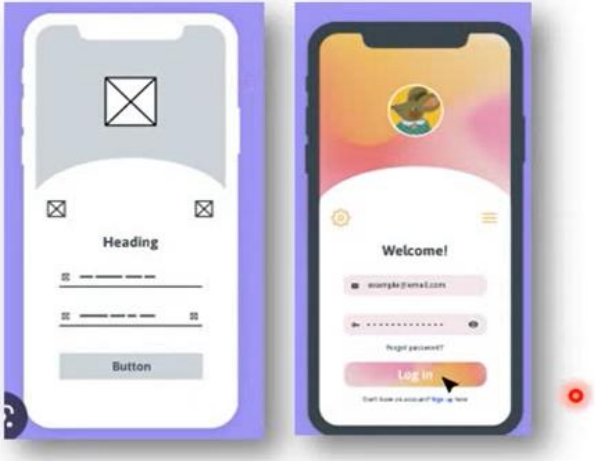
- Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production.
- The system ***undergoes routine maintenance*** for minimizing downtime and prevent large-scale failures.

Types of Prototypes

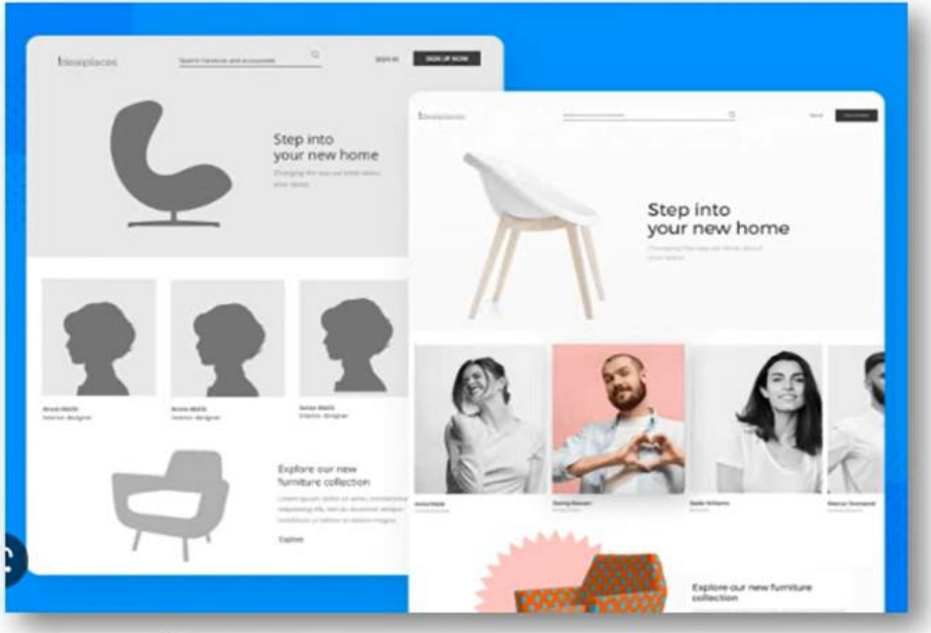
- There are four popular manufacturing methods to create a prototype model.
- Their use depends on the **project objective**, **timelines** and the **availability of resources**.
- The four types of prototype models include:
    1. **Rapid throwaway prototype**
    2. **Evolutionary prototype**
    3. **Incremental prototype**
    4. **Extreme prototype**

Example Prototype VS Actual

Prototype Actual



Prototype Actual



Prototype Actual

A company developing a new mobile app for fitness tracking may start with a basic prototype that includes core features like step tracking and user profiles. Users provide feedback on the interface, usability, and additional features, leading to multiple iterations before the final version is developed.

- Developing an initial prototype with basic features (e.g., a step counter and heart rate monitor).
- Presenting it to the client for feedback on the user interface and functionality.
- Iteratively refining the app by adding or modifying features based on the client's and end-users' feedback (e.g., adding a graph for daily activity or a calorie goal tracker).
- Finalizing the product once the prototype satisfies all requirements.

A video game studio might create a prototype of a new game level to test gameplay mechanics and user engagement. Players can provide feedback on difficulty, controls, and overall enjoyment, allowing the developers to tweak the game before full production.

Where is the prototype model of software development well suited?

- The answer is when a customer cannot define requirements clearly and when the user is ready to be actively involved and can provide feedback.
- Example
  - ***Online systems and web interfaces*** typically involve a high degree of end-user interaction and are best suited for prototype models.
  - A ***shopping website*** is also an example where a prototyping approach can be implemented. You can develop various web page prototypes for shopping pages, such as catalog pages, product order pages, etc., and submit them to customers for approval.

Advantages	Disadvantages
A prototype model may reduce costs in the final stages of development.	The upfront costs and the cost of developing prototypes are pretty high.
The prototype model has a flexible design and also has scope for innovative design.	The prototype model is a time-consuming process since a lot of testing is done on the prototype.
With a prototype model, the errors in the model are detected at an early stage.	Customers may have a misconception that the prototype model is the final product which may lead to difficulties.
The customers give quick feedback regarding the model, which helps to develop the final product faster.	Developers may make poor decisions regarding the quality of the prototype, which may affect the actual product.

SPIRAL MODEL

- The spiral model, initially proposed by ***Boehm*** in 1986.
- Spiral model is a ***risk driven software development process model***.
- A Spiral model SDLC represents a highly systematic approach to software development that essentially combines Waterfall and Iterative models.
- In its diagrammatic representation, looks like a spiral with many loops.
- The exact number of loops of the spiral is unknown and can vary from project to project.
- Each loop of the spiral is called a ***Phase of the software development process***.
- Each phase of spiral model in software engineering begins with a design goal and ends with the client reviewing the progress.
- The spiral model for software development starts with collecting a small set of requirements and goes through a small development phase.
- The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.

1. Planning (Requirement Gathering)

- The first phase of the Spiral Model is the planning phase, where the ***scope of the project is determined*** and a ***plan is created*** for the next iteration of the spiral.
- It also gathers elaborative details of every small requirement.

2. Risk Analysis (Risk Management)

- The development team ***identifies***, ***assesses***, and ***mitigates potential risks*** associated with the project.
- The team ***evaluates alternative solutions*** and selects the best approach to address the identified risks effectively.
- Design a prototype of model

3. Engineering (Designing and Coding Phase)

- The engineering phase focuses on the ***actual development*** of the software product.
- This phase involves ***designing the architecture***, ***developing the software modules***, and ***conducting frequent testing and integration***.
- **Designer** design the product as per final prototype.
- **Developer** performs actual coding and implementation.
- **Tester** performs all testing methods.
- **Deploy** or release product to customer environement.

4. Evaluate (Review And Deployment)

- The evaluation phase involves rigorous ***testing*** and ***evaluation*** of the software product.
- The team assesses the developed features against the established requirements and objectives.
- Any discrepancies or deviations are documented and addressed promptly.
- This phase also includes ***user feedback*** and involvement to ensure the software meets their expectations.

Imagine developing a new e-commerce website:

Requirement Analysis:
- Gather and document requirements.
- Identify alternative solutions.

Identify features like user registration, product catalog, shopping cart, and payment gateway.

Risk Analysis:
- Evaluate risks associated with each solution.
- Develop strategies to mitigate risks.

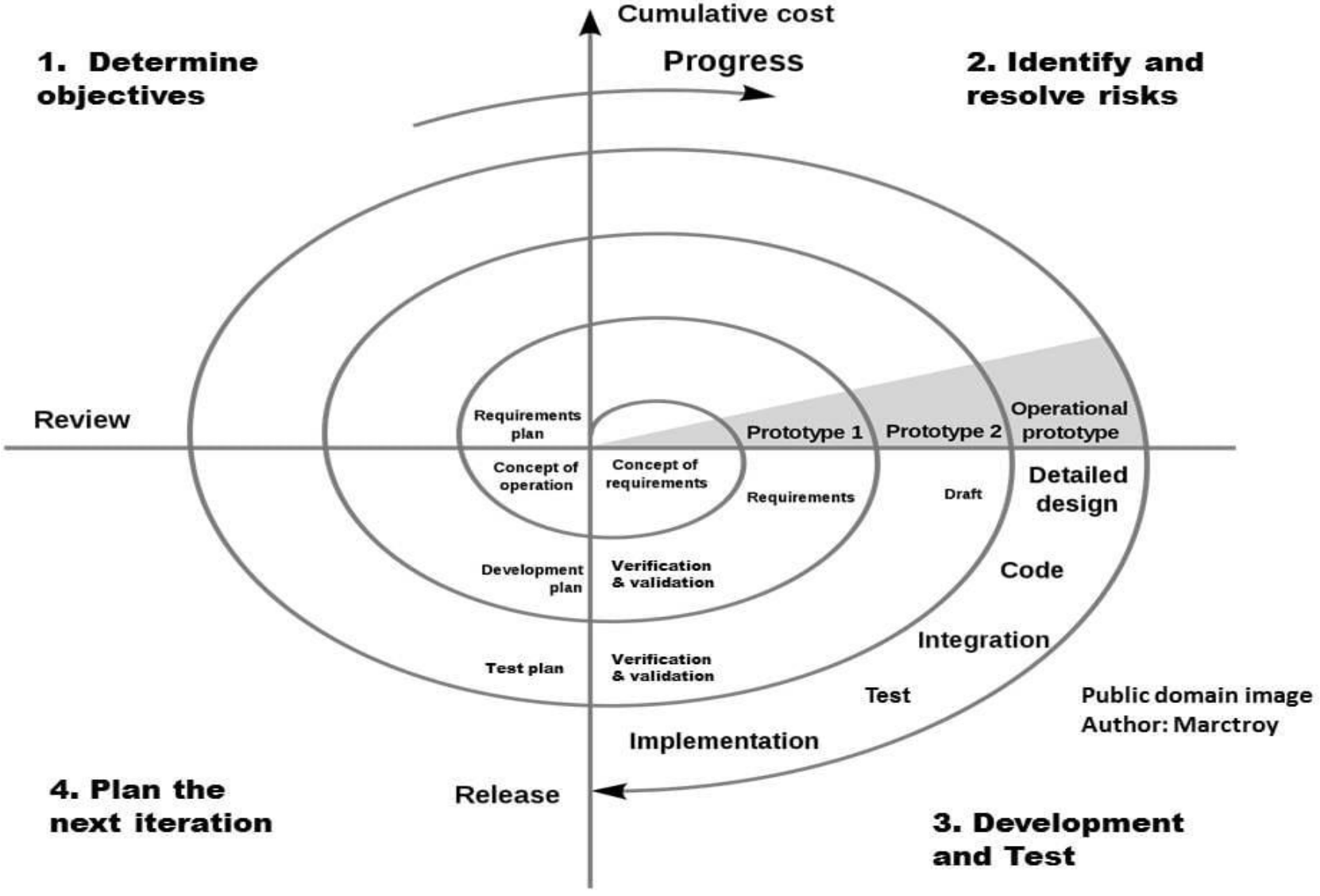
Assess risks such as security vulnerabilities, payment gateway integration issues, and scalability concerns.
Plan how to address these risks, like implementing secure coding practices and choosing a reliable payment gateway.

Prototyping:
- Develop a prototype based on the chosen solution.
- Test and refine the prototype.

Create a basic version of the website with essential features.
Test the prototype with a small group of users to gather feedback.

Evaluation:
- Review the progress and performance.
- Plan the next iteration.

Review user feedback and performance metrics.
Plan the next iteration to add more features or improve existing ones.


Cumulative cost
Progress
1. Determine objectives
2. Identify and resolve risks
Review
Requirements plan
Concept of operation
Concept of requirements
Prototype 1
Prototype 2
Operational prototype
Requirements
Draft
Detailed design
Development plan
Verification & validation
Code
Integration
Test plan
Verification & validation
Test
Implementation
Release
4. Plan the next iteration
3. Development and Test
Public domain image
Author: Marctroy

Real-World Examples of Spiral Model Implementation

The following are where programmers use the Spiral Model.

- Developing ***large-scale gaming applications***.
- Developing ***electronic health record systems*** in the healthcare industry.
- In the ***e-commerce platforms*** for incorporating new features or improving existing ones without compromising system stability or security.
- Development of ***mobile applications***.

Advantages

Continuous risk analysis

It lessens the possibility of the software project failing!

Convenient software revisions

Big tasks are broken down into step-by-step processes.

Open communication

Clients can see what is happening in every step of the software development.

Transparency

Developers review and assess each step in the spiral.

Best for large and expensive projects

Especially if the client's priority is controlling the budget!

Applications

GANTTPRO

Pros

A high amount of risk analysis which helps risk avoidance is enhanced.

Estimating costs is as easy as completing a prototype in a small fragment.

Good application for large and important projects.

Strict document control and approval.

Additional functionality or changes can be added at a later stage.

Software will be produced early in the life of the software.

Applications are developed rapidly and features are added systematically.

There are always chances for customers to give feedback on the product.

Cons

Need a highly qualified specialist to perform the analysis.

Not useful for small-scale projects.

The project time and cost can be limitless

The project documentation can be very long because of the intermediate stages.

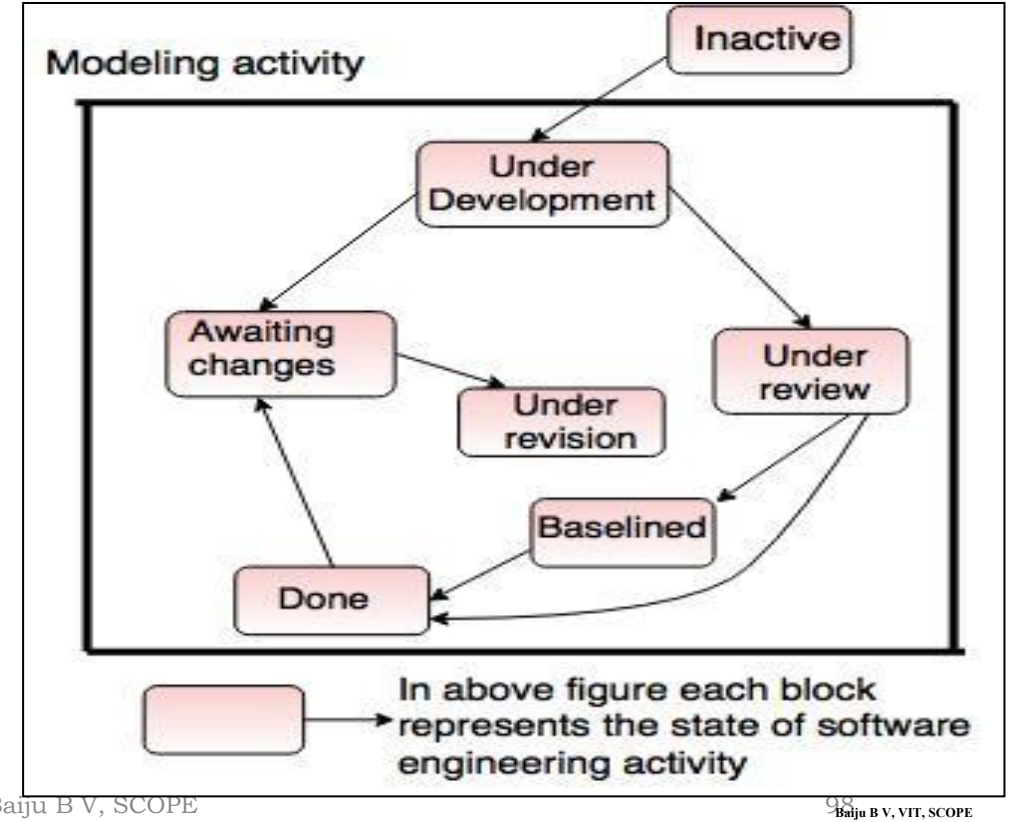
The risk may not meet the schedule or budget.

The success of the project depends heavily on the risk analysis phase.

Concurrent Models

- The concurrent development model, sometimes called concurrent engineering.
- The term concurrent mean “done at the same time”.
- Designed to handle multiple tasks simultaneously, improving efficiency and responsiveness.

- **Inactive** : No any activity / state performed
- **Under development** : Any activity performed
- **Awaiting changes** : If customer want any change
- **Under review** : Testing activity start
- **Under revision**: Do all required changes
- **Baselined** : As per SRS document
- **Done** : Project completed and deployed



- The **communication activity** has completed in the first iteration and exits in the awaiting changes state.
- The **modeling activity** completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- Concurrent process model activities moving from one state to another state.

1. System Requirements Analysis

- Gather and document the system requirements.
- Initial design considerations and risk analysis can begin while requirements are still being gathered.

2. System Design

- Develop the overall system architecture and design.
- Prototyping and initial coding of critical components can start while the design is being finalized.

3. Implementation

- Write and integrate the code for the system.
- Testing and debugging can occur in parallel with coding, allowing for immediate feedback and adjustments.

4. Testing

- Verify that the system meets the specified requirements and is free of defects.
- User training and documentation can be developed while testing is ongoing.

5. Deployment

- Release the system to the users.
- Maintenance planning and initial support activities can begin as the system is being deployed.

6. Maintenance

- Provide ongoing support and updates to the system.
- Feedback collection and planning for future iterations can occur while maintenance is being performed.

Applications

Air traffic control systems, where multiple sensors and control inputs must be processed concurrently to ensure safety and efficiency.

Mobile networks, where concurrent handling of multiple calls, messages, and data sessions is crucial for performance.

Modern operating systems like Windows, Linux, and macOS use concurrent models to manage multiple processes and threads. This enables efficient multitasking and resource management.

Big data processing frameworks like Apache Hadoop and Apache Spark use concurrent models to process large datasets in parallel. This significantly speeds up data processing and analysis.

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.

- ***Your team is developing a real-time weather monitoring system. The system includes components for collecting sensor data, analyzing patterns, and displaying results on a dashboard. Different components are at various stages of development:***
  - ***Sensor data collection is in the implementation phase.***
  - ***Data analysis algorithms are in the design phase.***
  - ***The dashboard interface is under testing.***

How would you apply the Concurrent Model to ensure seamless integration of all components?

System Requirements Analysis:

- Requirements for sensors, algorithms, and dashboards evolve with ongoing feedback from stakeholders.

System Design:

- Algorithm designs and architectural decisions adapt to testing outcomes of sensors and dashboards.

Implementation:

- Coding and integration of sensors are aligned with algorithm prototypes and dashboard requirements.

Testing:

- Dashboard is tested using real sensor data, and algorithms are verified for accuracy and efficiency.

Deployment:

- Components are deployed iteratively, allowing partial system rollouts for early user feedback.

Maintenance:

- Continuous updates and refinements occur as data from live deployments highlight areas for improvement.

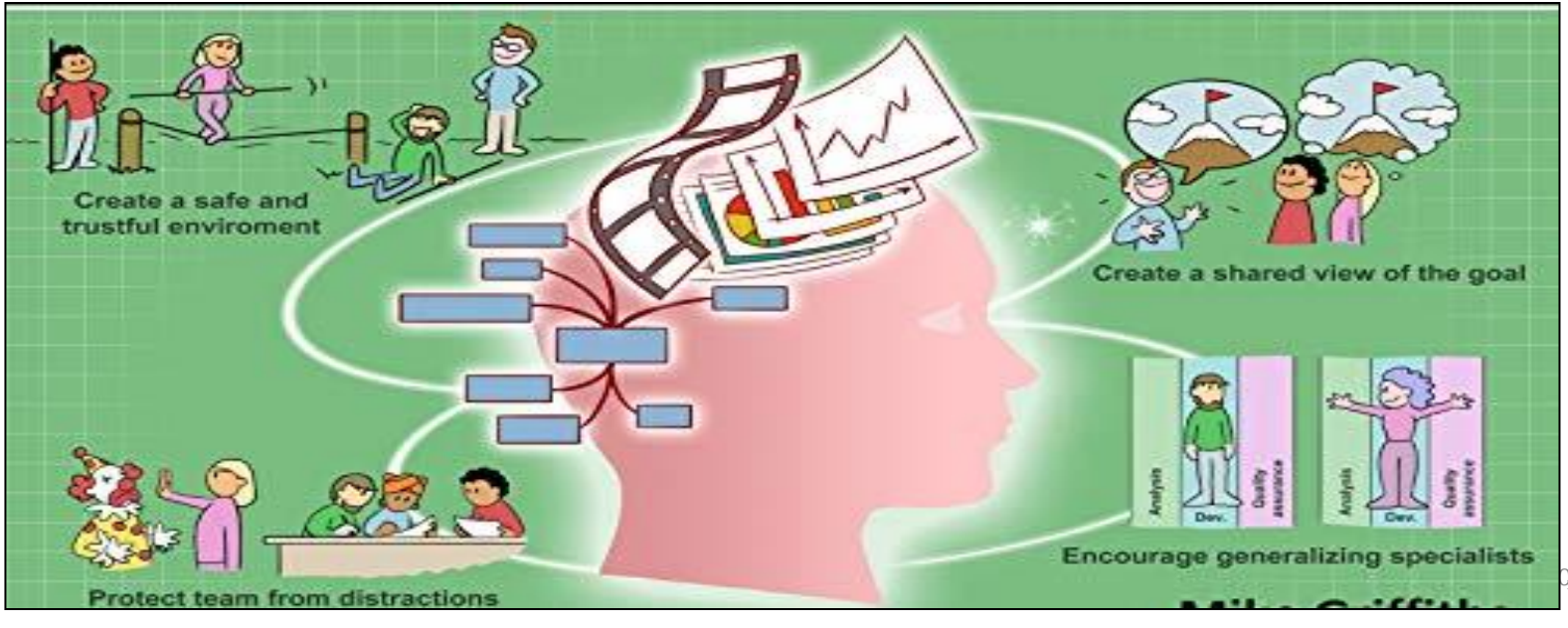
Waterfall vs. Incremental vs. Spiral vs. Rad Model: Comparison of Various SDLC Models

Properties of Model	Water-Fall Model	Incremental Model	Spiral Model	Rad Model
Planning in early stage	Yes	Yes	Yes	No
Returning to an earlier phase	No	Yes	Yes	Yes
Handle Large-Project	Not Appropriate	Not Appropriate	Appropriate	Not Appropriate
Detailed Documentation	Necessary	Yes but not much	Yes	Limited
Cost	Low	Low	Expensive	Low
Requirement Specifications	Beginning	Beginning	Beginning	Time boxed release
Flexibility to change	Difficult	Easy	Easy	Easy
User Involvement	Only at beginning	Intermediate	High	Only at beginning

Properties of Model	Water-Fall Model	Incremental Model	Spiral Model	Rad Model
Maintenance	Least	Promotes Maintainability	Typical	Easily Maintained
Duration	Long	Very long	Long	Short
Risk Involvement	High	Low	Medium to high risk	Low
Framework Type	Linear	Linear + Iterative	Linear + Iterative	Linear
Testing	After completion of coding phase	After every iteration	At the end of the engineering phase	After completion of coding
Working software availability	At the end of the life-cycle	At the end of every iteration	At the end of every iteration	At the end of the life cycle
Team size	Large Team	Not Large Team	Large Team	Small Team

Agility and Agile process

- Agility and Agile are two terms commonly used in the context of organizational adaptability and software development.
- **Agility** refers to an organization's ***ability to respond quickly*** and ***adapt to changing circumstances***.
- **Agile** is a specific software development methodology that emphasizes ***iterative and collaborative project management***.



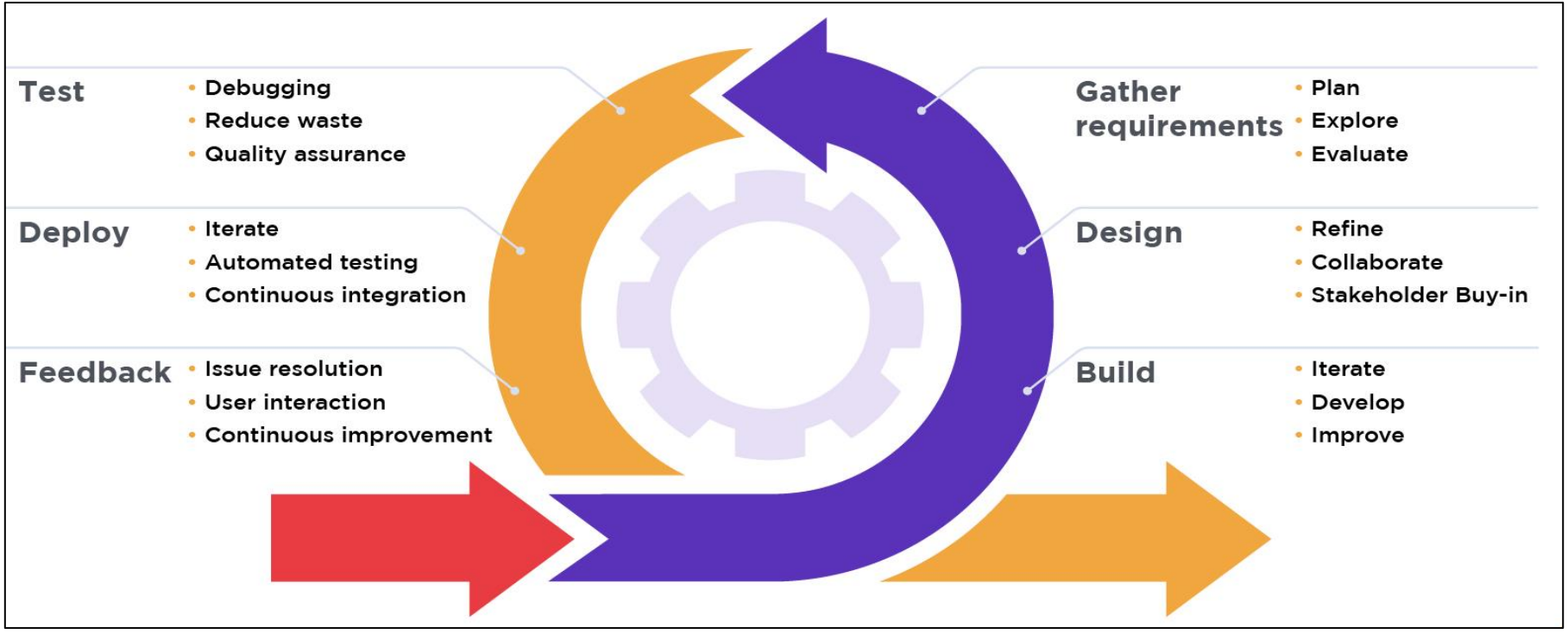
- **Agile model** is a combination of ***iterative and incremental process models*** with focus on process ***adaptability*** and ***customer satisfaction*** by ***rapid delivery of working software***.
- The agile methods were developed to overcome the weakness of conventional software engineering.
- The Agile Model ***breaks the process into manageable chunks*** called **iterations**.
- Each iteration, typically lasting ***two to four weeks***, has defined goals, duration, and scope.
- Each iteration involves a ***team of developers*** working through a complete SDLC, including **planning**, **design**, **coding**, **requirements analysis**, and **testing** before a working software product is demonstrated to the client.
- This approach allows for continuous feedback and adjustment throughout the development lifecycle.
- The Agile Model minimizes risks and optimizes project delivery time by dividing the project into smaller parts.

- By leveraging the Agile Model, software development teams
  - ***can adapt to change more effectively***
  - ***improve project visibility***
  - ***mitigate risks***
  - ***deliver high-quality software promptly.***

5 Values of Agile Modeling

Values	Description
Communication	Encouraging open and effective communication among team members rather than relying solely on extensive documentation.
Multiple Models	Recognizing that different stakeholders may need various models to understand the system which allows for flexibility in representation.
Simplicity	Promoting the creation of simple and clear models to avoid unnecessary complexity
Feedback	Seeking regular feedback from stakeholders to refine and improve models
Collaboration	Adopting collaboration and shared understanding among team members and stakeholders

Phases of Agile Model



1. Requirements gathering

- The requirements are defined in this phase, which involves ***engaging stakeholders*** (product owners and end-users)
- Explain the ***business opportunities*** and ***plan the time*** and ***effort needed to build the project***.
- Based on this information, the technical and economic feasibility are evaluated.

2. Design the Requirements:

- Once the project is identified, we must work with stakeholders to define requirements.
- The ***user flow diagram*** or the ***high-level UML***(Unified Modeling Language) diagram are used to ***show the work of features*** and ***show how it will apply to the existing system.***

3. Construction/Iteration

- The real work begins at this stage after the software development team defines and designs the requirements.
- Agile development is carried out in iterative cycles, often referred to as "***sprints***."
- Designers and developers start working on their project, which aims to deploy a working product.
- The product will undergo various stages of improvement, so it includes simple, minimal functionality.
- The focus is on ***delivering small, working increments of the product***.

4. Testing

- The **Quality Assurance(QA) team** examines the product's performance and verifies if any bug is present in the product.
- Automated testing using the right tools plays a vital role in maintaining the pace of development.

5. Deployment

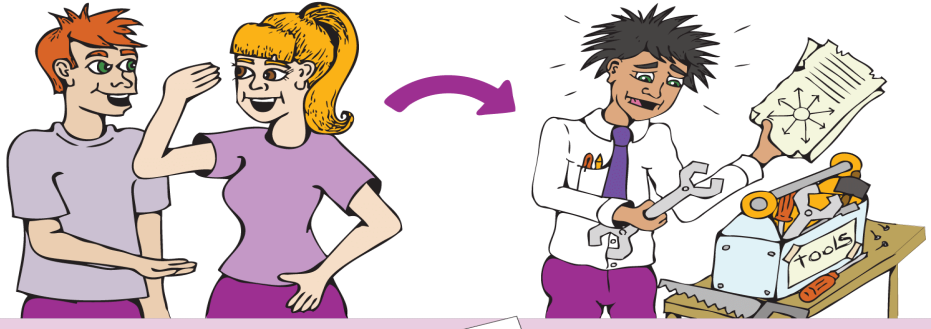
- The initial product is released to the user

6. Feedback

- The team receives feedback about the product and works on correcting bugs based on the received feedback.

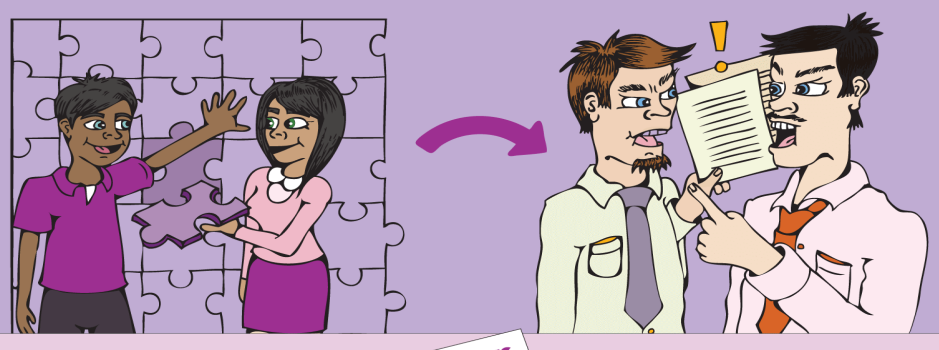
Manifesto for Agile Software Development*

"We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

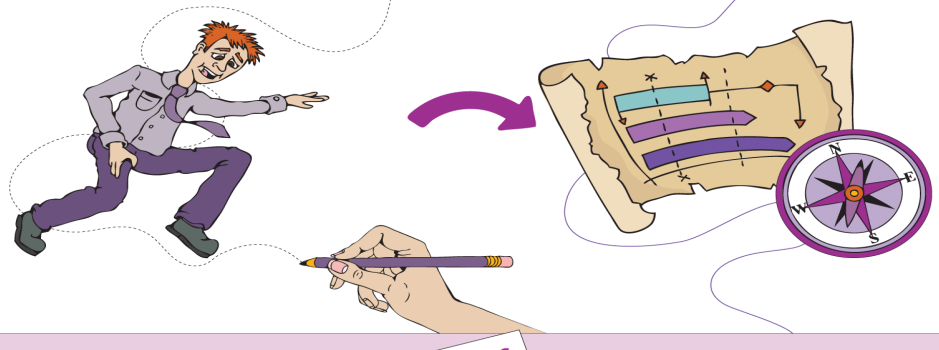


Individuals & interactions **Over** Processes & tools

Working software **Over** Comprehensive documentation



Customer collaboration **Over** Contract negotiation



Responding to change **Over** Following a plan

agilekrc

That is, while there is value in the items on the right, we value the items on the left more."

12 Agile Principles

The Agile Alliance defines 12 agility principles for those who want to achieve agility:

1. Individuals and interactions are given priority over processes and tools

- Focus is on adopting effective collaboration and communication among team members.
- The emphasis is on ***building solid relationships*** and ***enabling interactions*** that lead to better outcomes.

2. Adaptive, empowered, self-organizing team

- Agile empowers teams to take ***ownership*** and make decisions collectively.
- It encourages a self-organizing environment where team members can adapt to changing circumstances, ensuring flexibility and agility in project execution.

3. Focuses on working software rather than comprehensive documentation

- Agile values tangible results over extensive documentation.
- It ***prioritizes delivering functional software*** that **adds value to the customer**, **promoting rapid feedback** and **continuous improvement**.

4. Welcome changes in requirements, even late in the development phase

- Agile embraces change as an opportunity for improvement.
- It encourages open-mindedness and flexibility, allowing for the incorporation of new requirements and feedback at any stage of the development process.

5. Daily cooperation between businesspeople and developers

- Agile promotes constant collaboration between the development team and business stakeholders.
- The Agile Model ensures that ***everyone remains aligned with project goals*** and ***customer needs*** by maintaining regular communication and involvement.

6. Priority is customer collaboration over contract negotiation

- Agile recognizes the importance of involving customers in the development process.
- It emphasizes working closely with customers, gathering their feedback, and adapting to their evolving requirements to ensure the final product meets their expectations.

7. It enables you to satisfy customers through early and frequent delivery

- Agile aims to provide value to customers quickly and continuously.
- By delivering working software in short iterations, the Agile Model allows customers to provide feedback early, ensuring their satisfaction is prioritized throughout the development lifecycle.

8. A strong emphasis is placed on face-to-face communication

- Agile values **direct** and **personal communication** as it adopts better understanding and promotes effective collaboration.
- **Face-to-face discussions**, whether in person or via video conferencing, are encouraged to facilitate clear and efficient communication.

9. Developing working software is the primary indicator of progress

- Agile focuses on tangible outcomes.
- Progress is measured by the successful development and delivery of working software rather than relying solely on theoretical or abstract milestones.

10. Promote sustainable development pace

- Agile recognizes the importance of maintaining a workable pace throughout the project.
- It aims to avoid burnout and maintain a healthy work-life balance for team members, ensuring their productivity and well-being in the long run.

11. A continuous focus is placed on technical excellence and sound design

- Agile emphasizes the importance of quality in software development.
- It encourages ***adopting best practices***, ***continuous learning***, and the ***pursuit of technical excellence*** to deliver a robust and reliable product.

12. Regular team improvement reviews are conducted

- Agile promotes a culture of continuous improvement.
- Teams regularly reflect on their processes, identify areas for enhancement, and make necessary adjustments to optimize their performance and deliver even better results.

The 12 agile principles*

1 Satisfy the **customer**

Welcome **change**

Deliver **frequently**

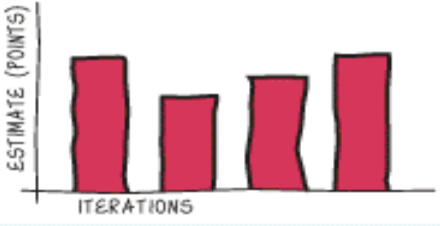
4 Work **together**

5 Trust and **support**

Face-to-face **conversation**

Working **software**

8 Sustainable **development**



9 Continuous **attention**

10 Maintain **simplicity**

11 Self-organizing **teams**

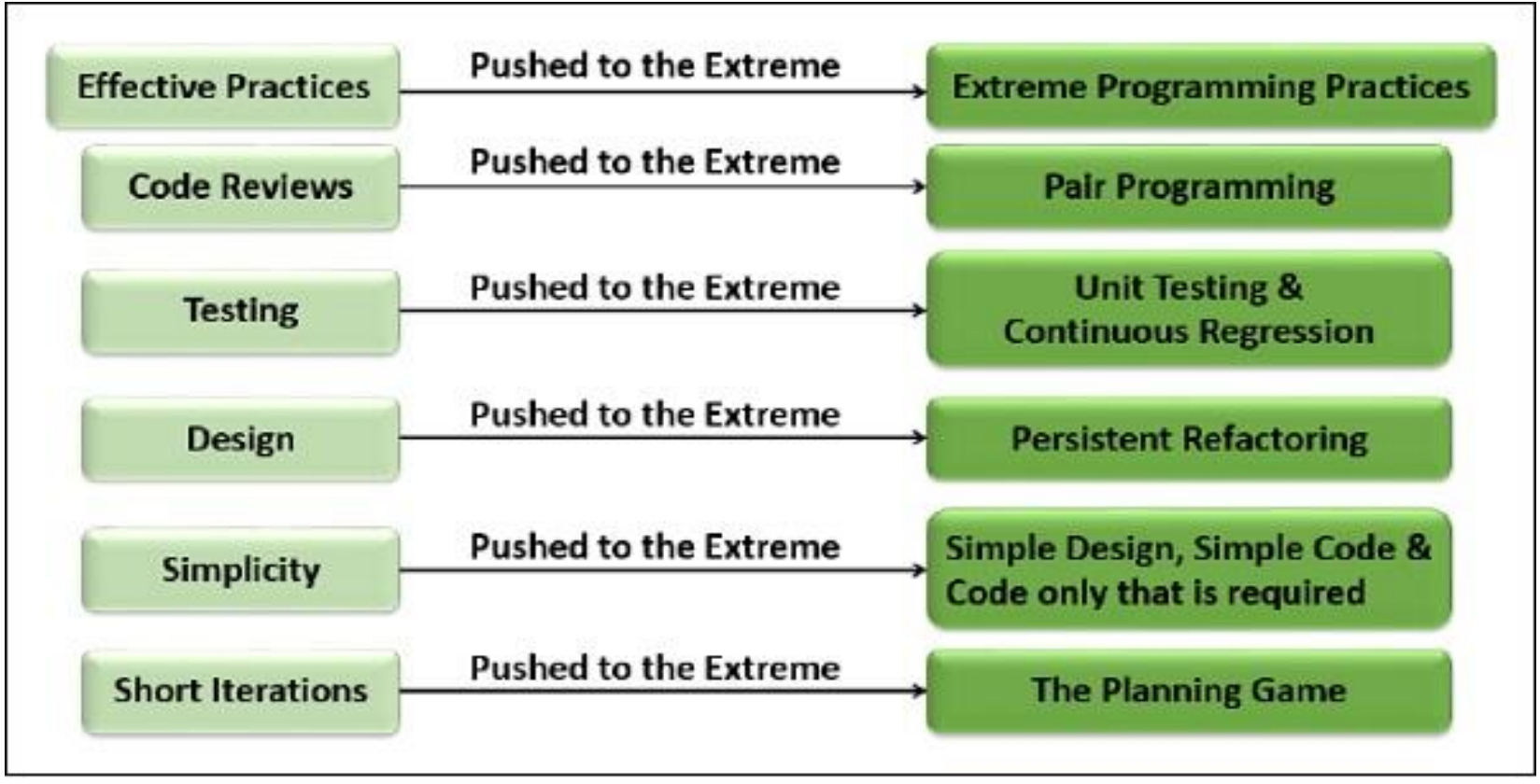
12 Reflect and **adjust**

EXTREME PROGRAMMING (XP)

- **Extreme Programming (XP)**, the most widely used approach to agile software development, originally proposed by ***Kent Beck***.
- Extreme Programming uses an ***object-oriented approach*** as its preferred development paradigm and encompasses a set of rules and practices that occur within the context of four framework activities:
  - Planning
  - Design
  - Coding
  - Testing.

- According to the Alliance, "***Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software and higher quality of life for the development team***".
- XP is built upon values, principles, and practices, and its goal is to allow small to mid-sized teams to produce high-quality software and adapt to evolving and changing requirements.

- Extreme Programming takes the effective principles and practices to extreme levels.



XP Values

• XP has simple rules that are based on 5 values to guide the teamwork

1. Communication

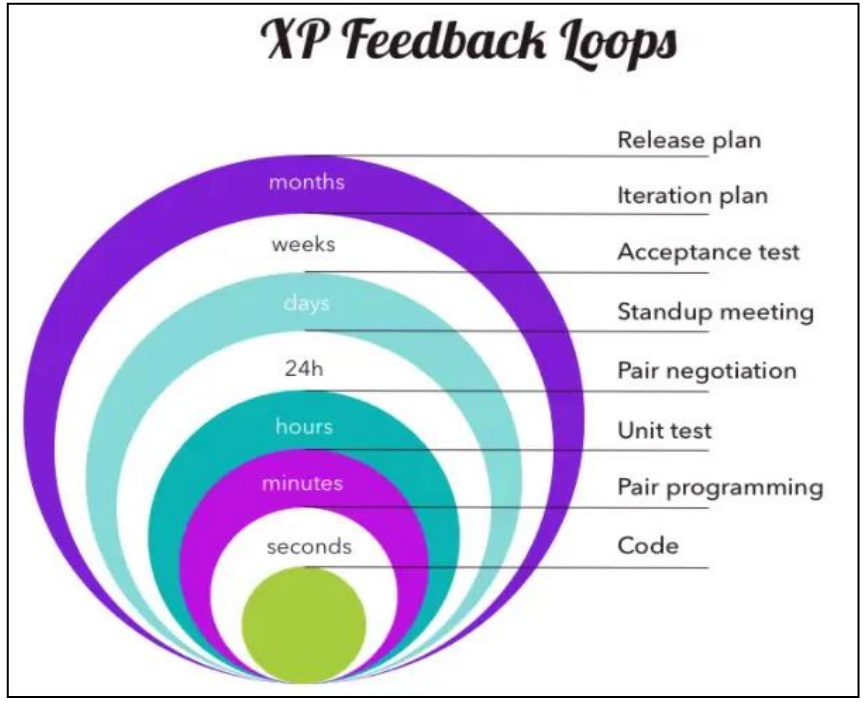
- Effective communication between the team members helps to transfer knowledge among them.
- XP particularly stresses the importance of proper communication, including ***face-to-face discussion*** with the aid of different ***drawing mechanisms***, including whiteboard discussion.

2. Simplicity

- Developers strive to ***write simple code bringing more value to a product***, as it saves time and effort.
- In extreme programming, the design addresses only the requirements and doesn't predict the future

3. Feedback

- Constant feedback helps to ***identify areas for improvement***.
- With the review of previous efforts, teams can revise their practices.
- This also helps to maintain a simple design.



4. Respect

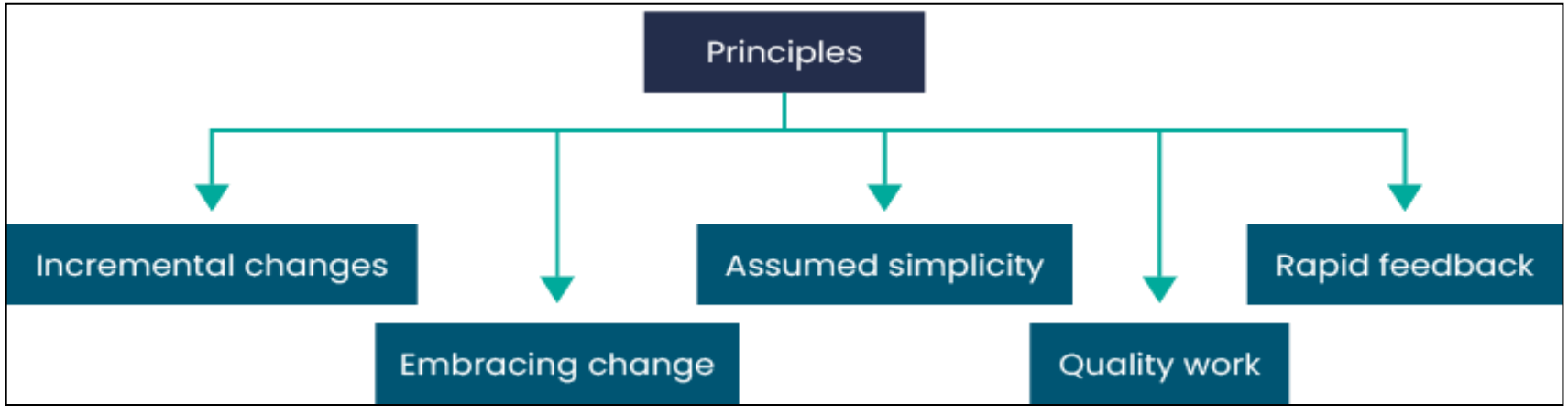
- To streamline effective communication and feedback among the team members, ***mutual respect among the team members*** is essential.
- This helps the team members to work together to identify simple designs and solutions.

5. Courage

- Courage is more ***about taking action while tackling fearful situations***.
- ***Giving and receiving honest feedback*** is necessary.
- Also, accepting that the solution provided to a problem or bug that received substantial investments is a failure is equally required.

XP Principles

- Principles guide individuals in specific ways that make the values more transparent and less ambiguous.



1. Incremental changes

- It's better to make small changes step-by-step than to let them accumulate and handle them all simultaneously.

2. Embrace change

- Speaking of changes, if the client wants to modify the product, programmers should support the idea and map out how they will incorporate the new changes.

3. Assumed simplicity.

- Developers need to focus on the job that is important at the moment and follow YAGNI (You Ain't Gonna Need It) and DRY (Don't Repeat Yourself) principles.

If you're building a login feature, focus only on the current authentication method rather than adding multiple login options (like social media login) that may not be necessary yet.

If you find yourself writing the same validation code for different forms, create a single validation function that can be used across all forms to reduce redundancy and make the code easier to maintain.

4. Quality work.

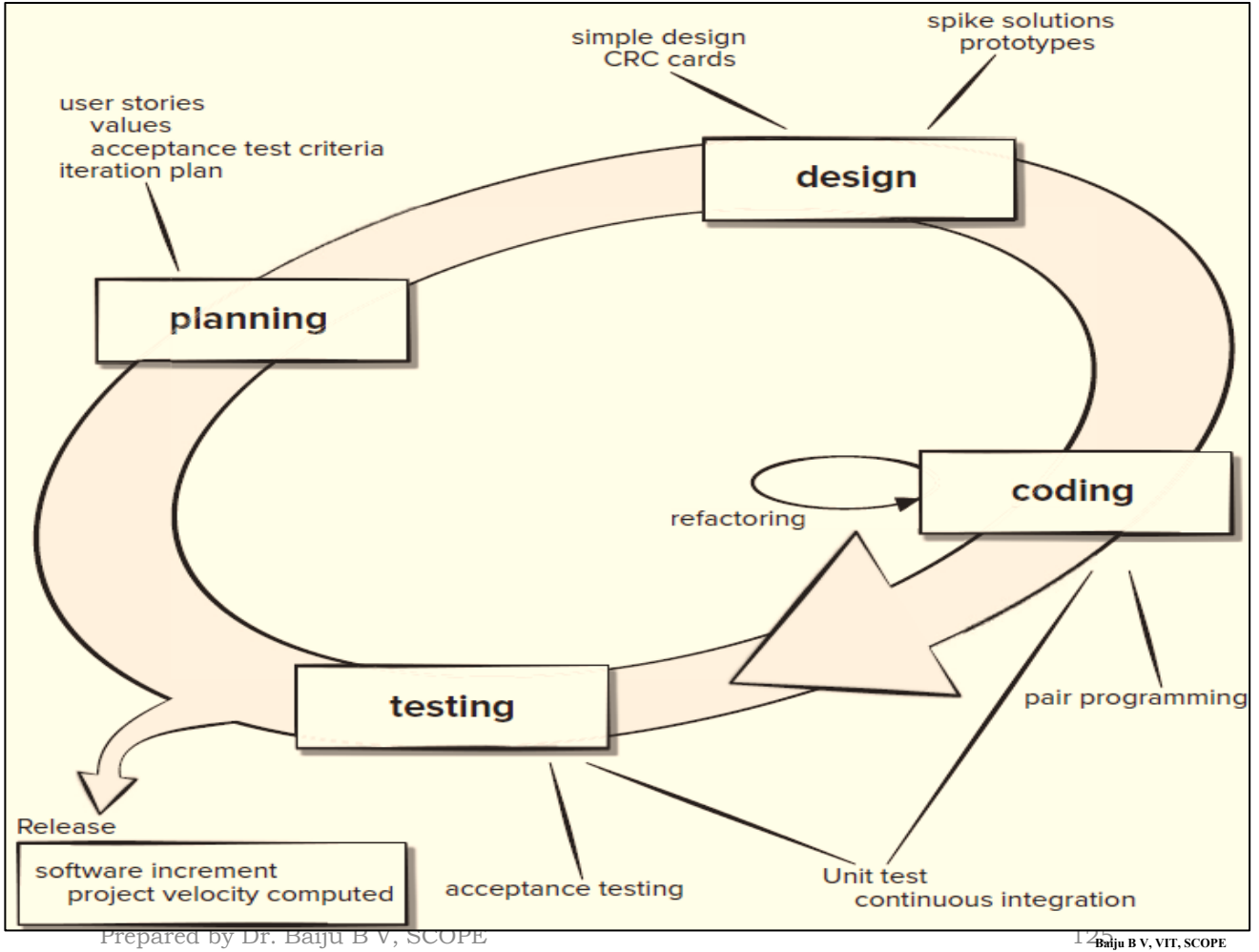
- A team that works well, makes a valuable product and feels proud of it.

5. Rapid feedback.

- Team members understand the given feedback and react to it right away.

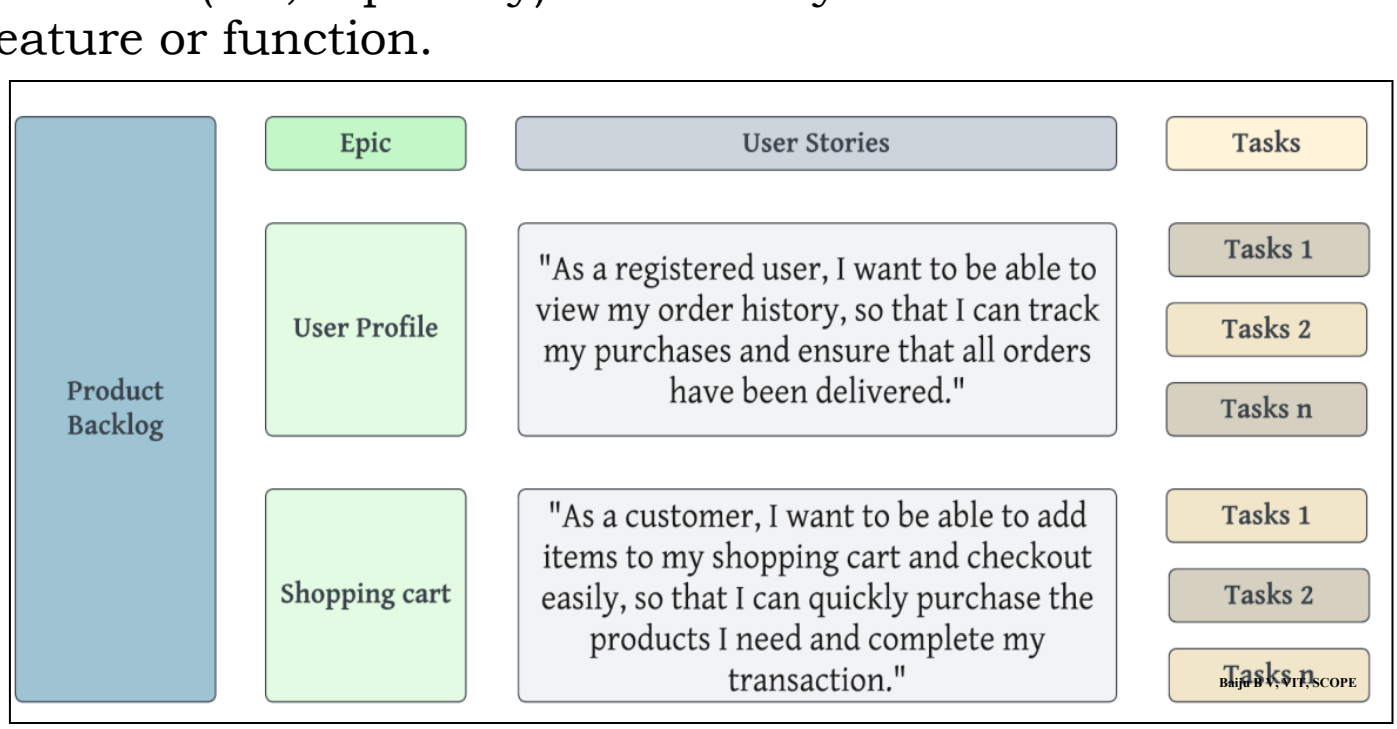
XP PROCESS

- Extreme Programming encompasses a set of rules and practices that occur within the context of four framework activities:
  - ***Planning***
  - ***Design***
  - ***Coding***
  - ***Testing***



1. Planning

- The planning activity (also called the ***planning game***) begins with a requirements activity called ***listening***.
- Listening leads to the creation of a set of "stories" (also called ***user stories***) that describe ***required output***, ***features***, and ***functionality*** for software to be built.
- Each *user story* is written by the customer and is placed on an ***index card***.
- The customer assigns a ***value*** (i.e., a priority) to the story based on the overall business value of the feature or function.
- Members of the XP team then assess each story and assign a ***cost*** (measured in development weeks) to it.
- It is important to note that ***new stories can be written at any time.***



- Customers and developers work together to decide how to group stories into the next release (the next software increment) to be developed by the XP team.
- Once a ***basic commitment*** (agreement on stories to be included, delivery date, and other project matters) is made for a release, the XP team orders the stories that will be developed in one of three ways:

Title:	Priority:	Estimate:
User Story: As a [description of user], I want [functionality] so that [benefit].		
Acceptance Criteria: Given [how things begin] When [action taken] Then [outcome of taking action]		

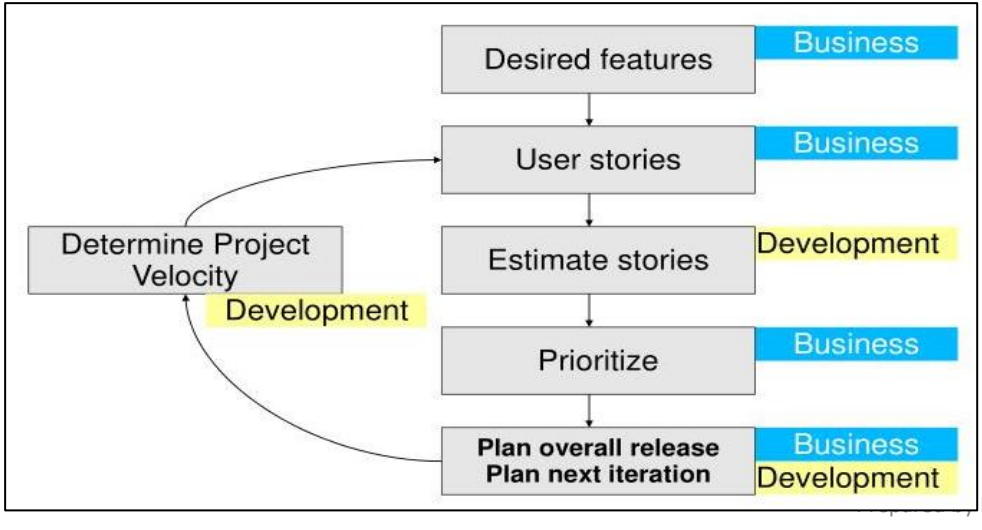
(1) **All stories will be implemented immediately (within a few weeks)**
(2) **Stories with highest value will be moved up in the schedule and implemented first**, or
(3) **Riskiest stories will be moved up in the schedule and implemented first**.

As an Account Manager
I want a sales report of my account to be sent to my inbox daily
So that I can monitor the sales progress of my customer portfolio

Acceptance criteria:
1. The report is sent daily to my inbox
2. The report contains the following sales details: …
3. The report is in csv format.

- After the **first project release** (also called a ***software increment***) has been delivered, the XP team computes project velocity.

- ***Project velocity*** is the **number of customer stories implemented during the first release**.
- Project velocity can then be ***used to help estimate delivery dates*** and ***schedule for subsequent releases***.
- The XP team modifies its plans accordingly.



STORY CARD NO: 16	Project Name E-Commerce	Estimation: 4 Hours
Story Name: User Registration		Date: 16/08/2007 1:30 PM
STORY: User needs to register with unique username and password before purchasing anything from the online store		Acceptance Test: 1. User Id must be unique 2. Try to register with duplicate user id and Password 3. Try to register user name only 4. Try to register with password only 5. Forget Password Link
Note: User Can View or Visit store as a Visitor but needs to register before purchasing anything		Risk: Low
Points to be Consider: There isn't any non-functional requirement at this stage		

2. Design

- XP design rigorously follows the ***KIS*** (keep it simple) principle.
- XP encourages the use of ***CRC cards*** as an effective mechanism for thinking about the software in an object-oriented context.
- CRC (class responsibility collaborator) cards identify and organize the object-oriented classes that are relevant to the current software increment.
- CRC cards are the only ***design work product*** produced as part of the XP process.

Template		**Example**	
Class Name *(collection of similar objects)*		Customer	
Responsibilities *(something that the class knows or does)*	Collaborators *(another class that this class interacts with)*	Places order Knows name Knows address Knows customer # Knows order history	Order

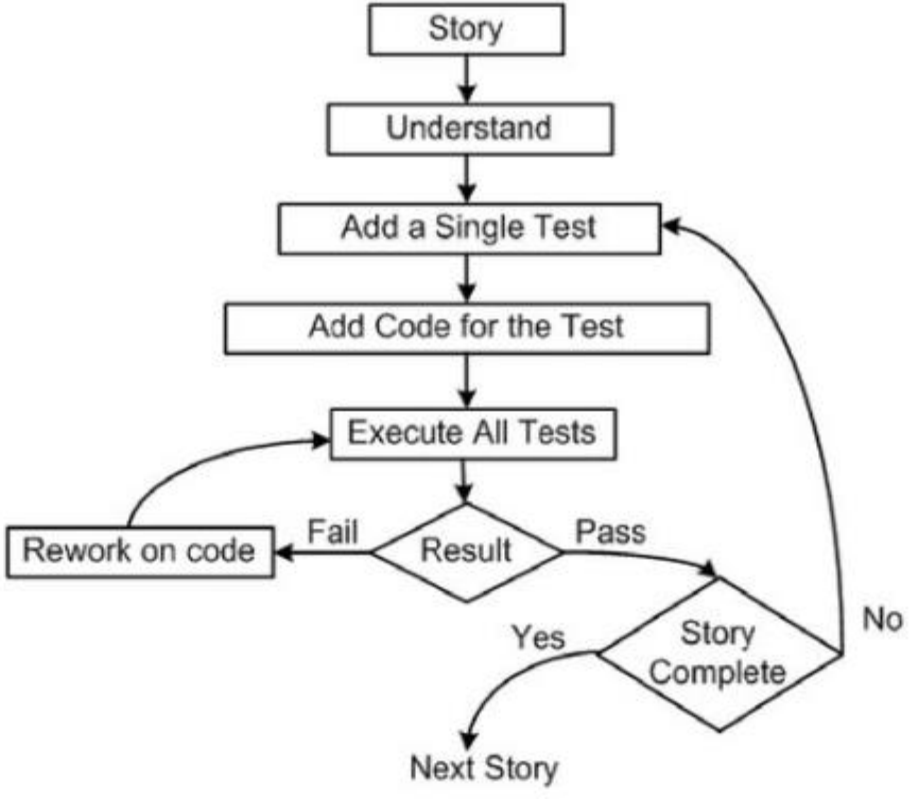
- If a **difficult design problem is encountered** as part of the design of a story, XP ***recommends the immediate creation of an operational prototype*** of that portion of the design.
- A central notion in XP is that design occurs both before *and after* coding commences.
- **Refactoring**—modifying/optimizing the code in a way that does not change the external behavior of the software means that design occurs continuously as the system is constructed.
- In fact, the construction activity itself will provide the XP team with guidance on how to improve the design.

3. Coding

- After user stories are developed and preliminary design work is done, the ***team does not move to code***, but rather ***develops a series of unit tests*** that will exercise each of the stories that is to be included in the current release (software increment).
- Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the test.
- Once the code is complete, it can be unit-tested immediately, thereby providing instantaneous feedback to the developers.

Unit Testing in eXtreme Programming

1. Pick a requirement, i.e., a ***story***
2. Write a test case that will verify a small part of the story and assign a fail verdict to it
3. Write the code that implement particular part of the story to pass the test
4. Execute all tests
5. Rework on the code, and test the code until all tests pass
6. Repeat step 2 to step 5 until the story is fully implemented

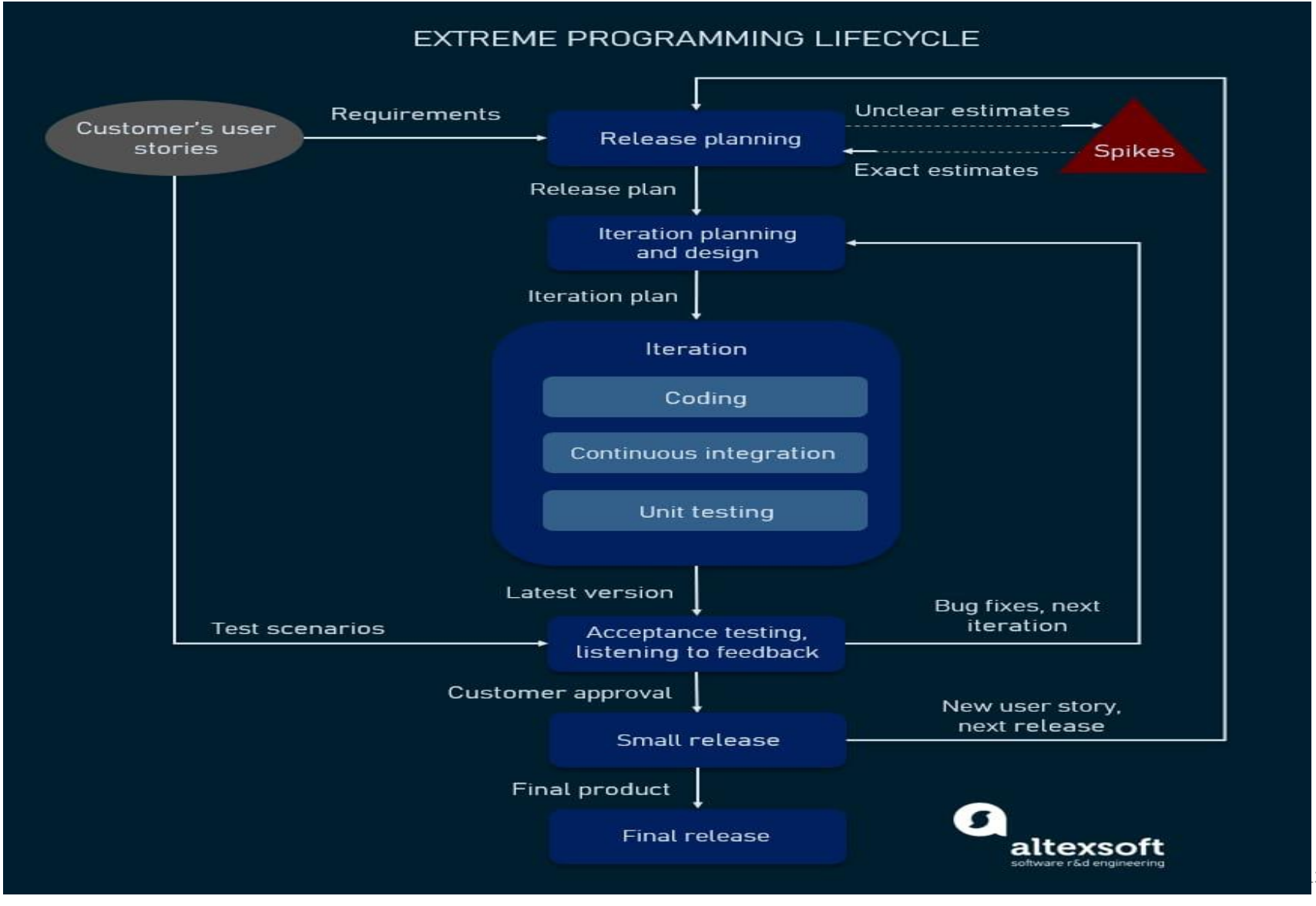


- A key concept during the coding activity is ***pair programming***.
- XP recommends that ***two people work together at one computer*** to create code for a story.
- This provides a mechanism for real-time problem solving (two heads are often better than one) and real-time quality assurance (the code is reviewed as it is created).

- As pair programmers complete their work, the ***code they develop is integrated with the work of others.***
- This "**continuous integration**" strategy helps uncover compatibility and interfacing errors early.

4. Testing.

- The unit tests that are created should be implemented using a framework that enables them to be automated (hence, they can be executed easily and repeatedly).
- This encourages implementing a **regression testing strategy** whenever code is modified (which is often, given the XP refactoring philosophy).
- XP ***acceptance tests***, also called *customer tests*, are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer.
- They are derived from user stories that have been implemented as part of a software release.


EXTREME PROGRAMMING LIFECYCLE
Customer's user stories
Requirements
Release planning
Unclear estimates
Spikes
Exact estimates
Release plan
Iteration planning and design
Iteration plan
Iteration
Coding
Continuous integration
Unit testing
Latest version
Test scenarios
Acceptance testing, listening to feedback
Bug fixes, next iteration
Customer approval
Small release
New user story, next release
Final product
Final release
altexsoft
software r&d engineering

EXTREME PROGRAMMING PROS AND CONS

Advantages

- Stable system
- Clear code
- Fast MVP delivery
- Less documentation
- No overtime
- High visibility
- Team collaboration
- Customer satisfaction

Disadvantages

- Unclear estimates
- Time waste
- Not enough documentation
- Big cultural change needed
- Pair programming takes longer
- Collocated teams only
- Stressful
- Code over design