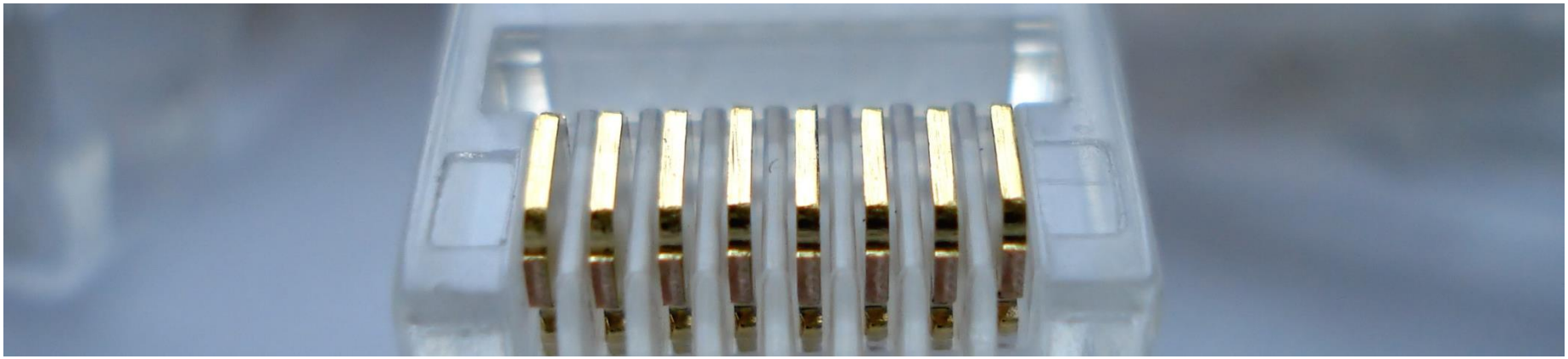
Introduction to Software Engineering

(AIM2220:SOFTWARE ENGINEERING and PROJECT MANAGEMENT)



SEPM Syllabus

AIM2220: SOFTWARE ENGINEERING & PROJECT MANAGEMENT [3 1 0 4]

Software Engineering – importance – emergence - Phases of software development - Feasibility study Phases and Life cycle models of Software Development. Requirement Analysis, Design, Implementation, Testing, and Maintenance phases Different Software Life Cycle Models - Classical waterfall, Iterative, prototyping, Spiral, and Agile - Compare Lifecycle models. Requirements Analysis and **Design Requirement Analysis** – Analysis process, Requirements specification, desirable characteristics of an SRS, structure of an SRS document, Data Flow Diagrams - Role of Software Architecture and Architecture. **Software Design** - Software design concepts - Function Oriented Design and its Complexity Metrics -Object Oriented Design and its Complexity Metrics - Detailed Design. **Software Implementation and Testing-** Software Coding- Programming principles and coding guidelines - method of incrementally developing code - managing the evolving code Testing - Unit testing and Code Inspection - Testing concepts and testing process - Design of Test case and Test plan - Black-box testing - White box testing. **Software Project Management-**Software Project Management Framework - methods to estimate project time and cost, Resource. Planning for a Software Project. Management, Identification, Analysis, mitigation, and monitoring of Project Risks - Ensuring Project. Quality and quality management, Configuration Management, Change management, CMMI, Quality standards -ISO.

References:

1. B. Hughes et al., *Software Project Management*, Sixth Edition, McGraw Hill, 2017
2. P Jalote., *Software Project Management in Practice*, First Edition, Addison Wesley Professional, 2010.
3. R. S. Pressman, *Software Engineering: A practitioner's approach*, Eighth Edition, McGraw Hill, 2014
4. S. A. Kelkar, *Software Project Management: a concise study*, Third Edition, PHI Learning-New Delhi, 2013
5. S. H. Kan, *Metrics and Models in Software Quality Engineering*, Second Edition, Pearson, 2010.

Overview

- Learning Objectives.
- What is software engineering?
- Why is software engineering important?

By the end of this chapter, you will...

- Understand what software engineering is.
- Understand why software engineering is important.
- Understand what SDLC models are.
- Know answers to key questions related to the software engineering discipline.

Why study Software Engineering?

- Building software without discipline is crazy
- Software is critical to society
- Building a large complete software project is hard
- There is a perceived crisis in our ability to build software
- It’s fun!
- $$$$ (money is involved)

Why Software Engineering?

A naive view:

Problem Specification 

Final Program

But ...

- Where did the *specification* come from?
- How do you know the specification corresponds to the *user's needs*?
- How did you decide how to *structure* your program?
- How do you know the program actually *meets the specification*?
- How do you know your program will always *work correctly*?
- What do you do if the users' *needs change*?
- How do you *divide tasks up* if you have more than a one-person team?

Why Software Engineering?

- **Software development is hard** !
- **Important to distinguish :**
  - **"easy" systems** (*one developer, one user, experimental use only*)
  - **"hard" systems** (*multiple developers, multiple users, products*)
- **Experience with "easy" systems is misleading**
  - *Single person techniques do not scale up*
- **Analogy with bridge building:**
  - Over a stream = easy, one person job
  - Over River Ganga… ? (*the techniques do not scale)*

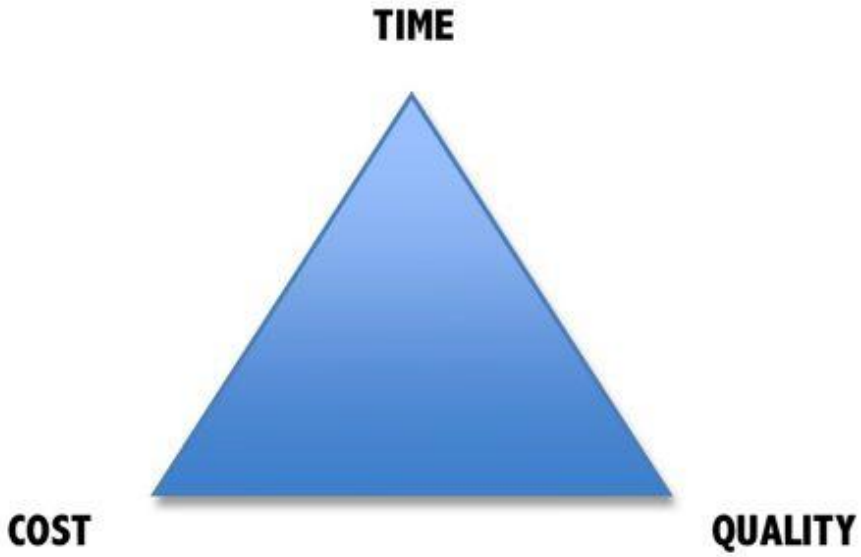
Why Software Engineering ?

- The problem is ***complexity***
- There are many sources of complexity, but size is key:
  - The Linux kernel contains >13 million lines of code
  - Windows XP contains >40 million lines of code

Software engineering is about managing this complexity.

Why is Software Engineering important?

Complex systems need a disciplined approach for designing, developing and managing them.



Software Development Crises

Projects were:

- Late.
- Over budget.
- Unreliable.
- Difficult to maintain.
- Performed poorly.

Software errors….*the cost*

Errors in computer software can have devastating effects.

Software Crisis

Example 1: 2009,Computer glitch delays flights

Saturday 3rd October 2009-London, England (CNN)

- Dozens of flights from the UK were delayed Saturday after a glitch in an air traffic control system in Scotland, but the problem was fixed a few hours later.
- The agency said it reverted to backup equipment as engineering worked on the system.
- The problem did not create a safety issue but could cause delays in flights.
- Read more at: http://edition.cnn.com/2009/WORLD/europe/10/03/uk.flights.delayed

Software Crisis

Example 2: Ariane 5 Explosion

- European Space Agency spent 10 years and $7 billion to produce Ariane 5.
- Crash after 36.7 seconds.
- Caused by an overflow error. Trying to store a 64-bit number into a 16-bit space.
- Watch the video: http://www.youtube.com/watch?v=z-r9cYp3tTE

Software Crisis

Example 3: 1992, London Ambulance Service

- Considered the largest ambulance service in the world.
- Overloaded problem.
- It was unable to keep track of the ambulances and their statuses. Sending multiple units to some locations and no units to other locations.
- Generates many exceptions messages.
- 46 deaths.

Therefore…

A well-disciplined approach to software development and management is necessary. This is called engineering.

What is Software Engineering?

- **Software:** It is a collection of integrated programs. Means it carefully-organized instructions and code written by developers on any of various particular computer languages.

- **Engineering:** It is the application of scientific and practical knowledge to invent, design, build, maintain and improve frameworks, processes, etc.

- **Software Engineering:** It is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques and procedures.

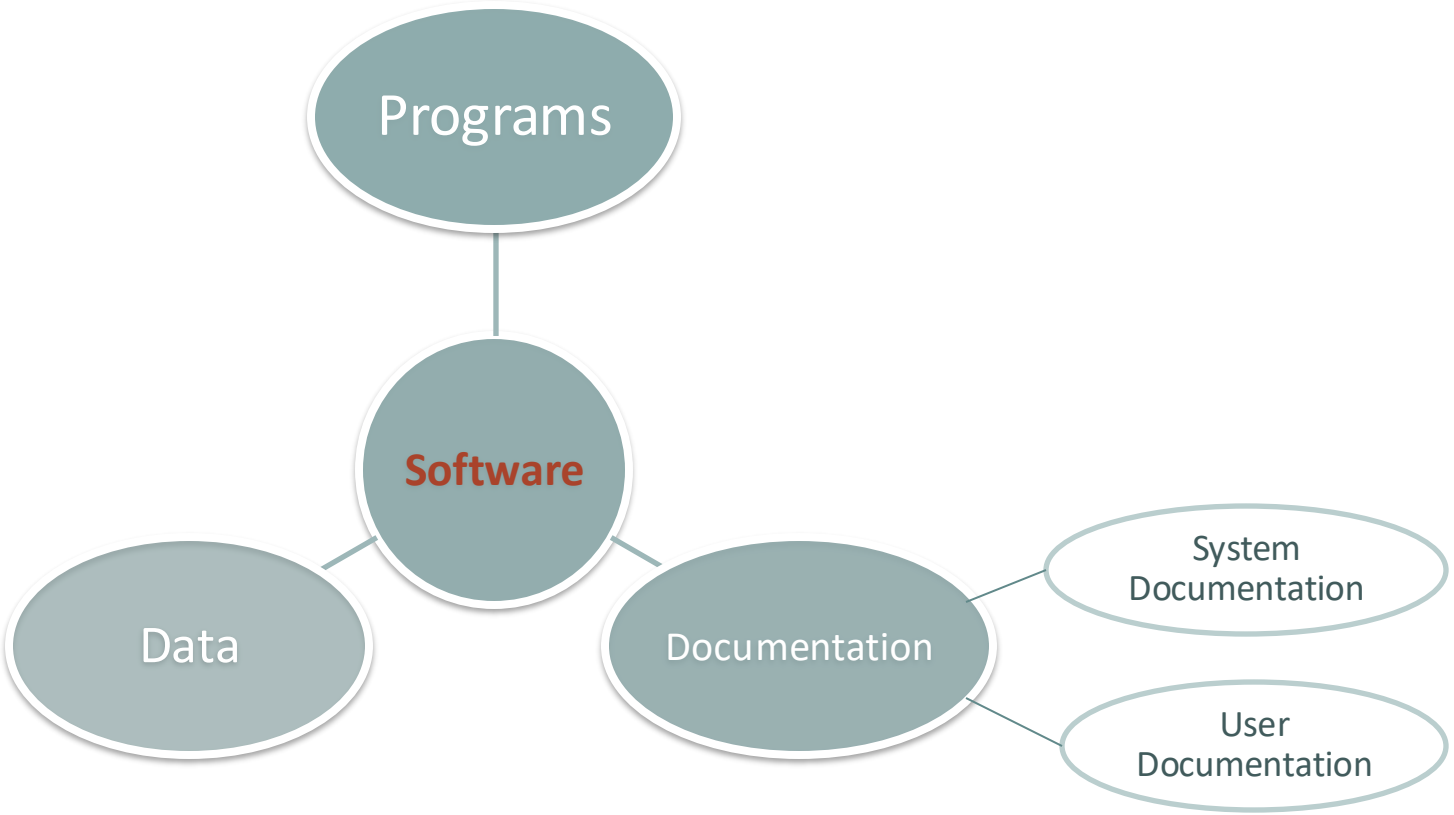
  The result of software engineering is an effective and reliable software product.

Software Engineering

- The term *software engineering* first appeared in the **1968** NATO Software Engineering Conference and was meant to provoke thought regarding what was then called the **"software crisis"..**

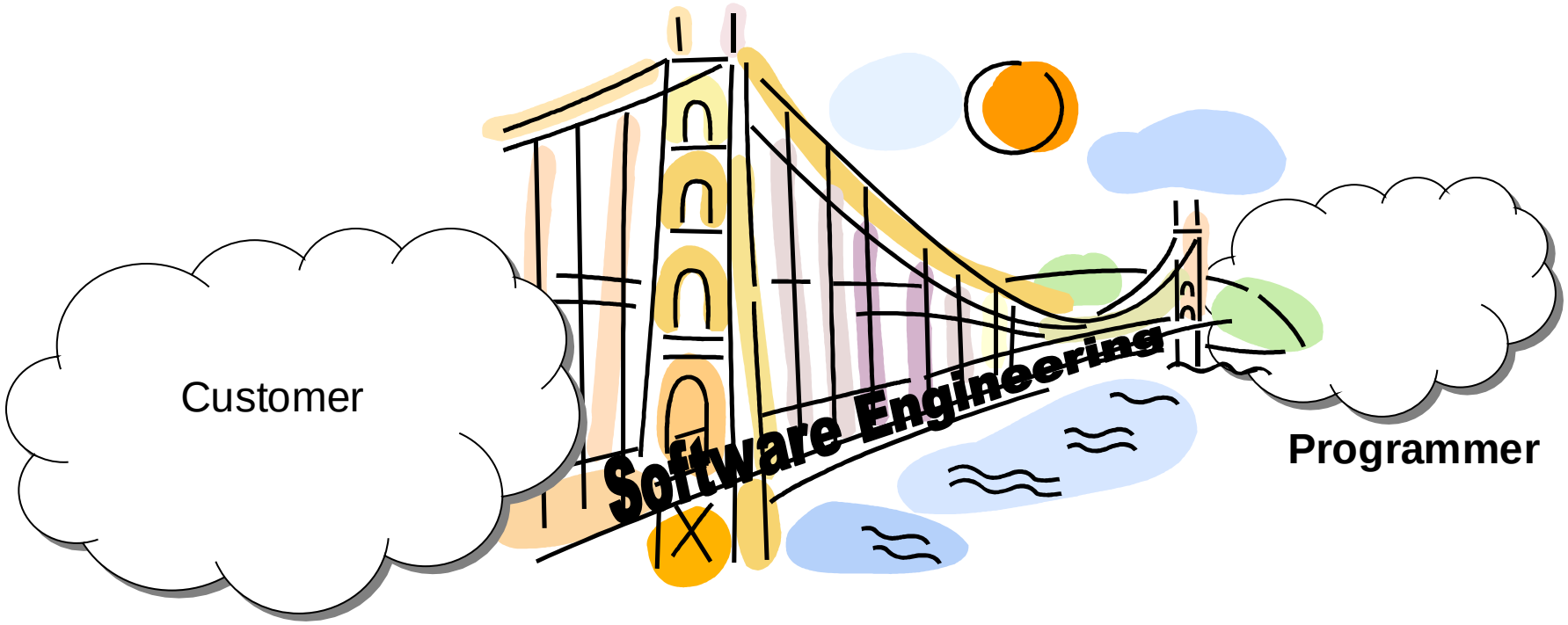
- ".. An engineering discipline that is concerned **with all aspects of software production** from the **early stages** of system specification to **maintaining the system after it has gone into use**." *Sommerville, pg.7*

What is Software?



The Role of Software Engg.

A bridge from customer needs to programming implementation

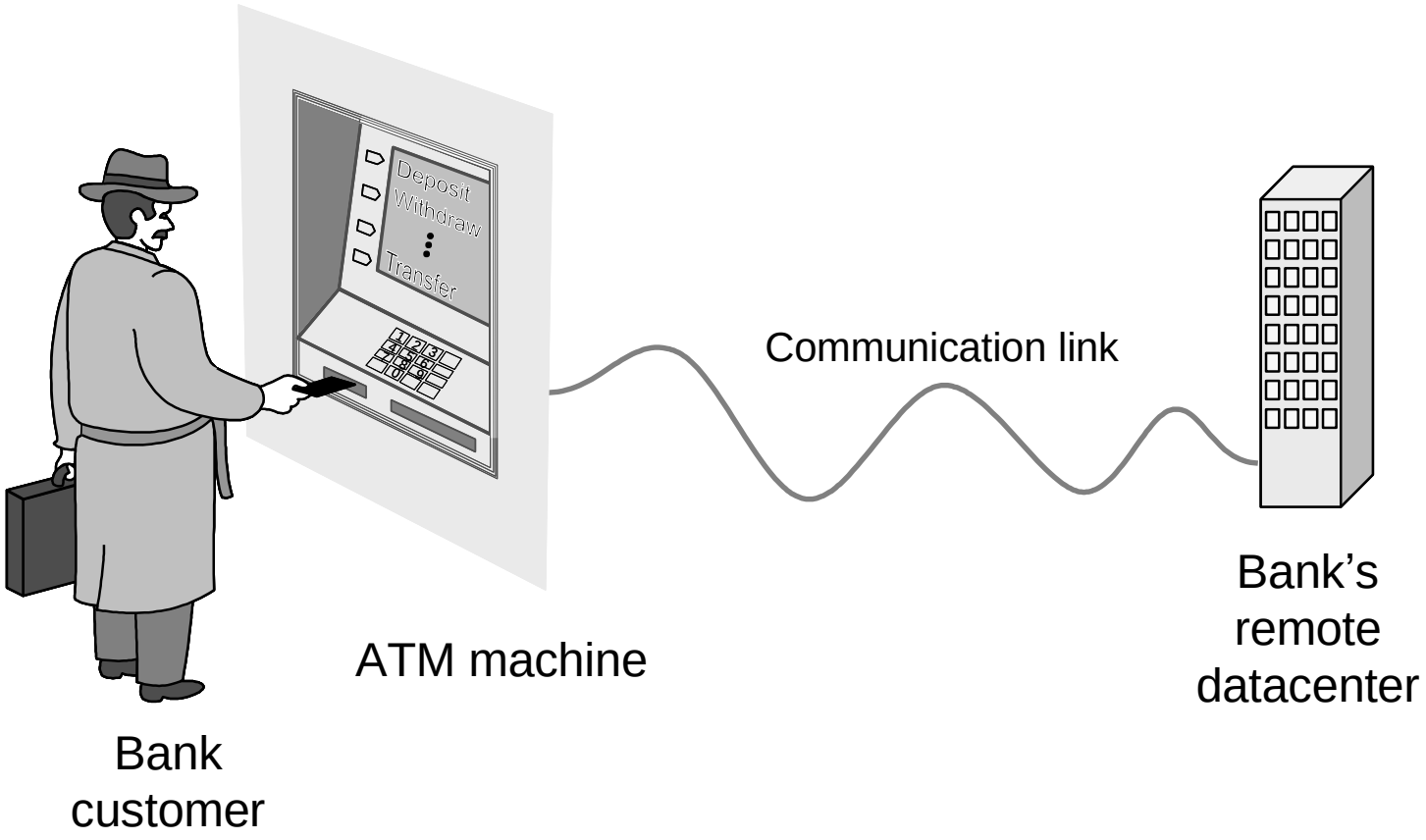


First law of software engineering

Software engineer is willing to learn the problem domain
(problem cannot be solved without understanding it first)

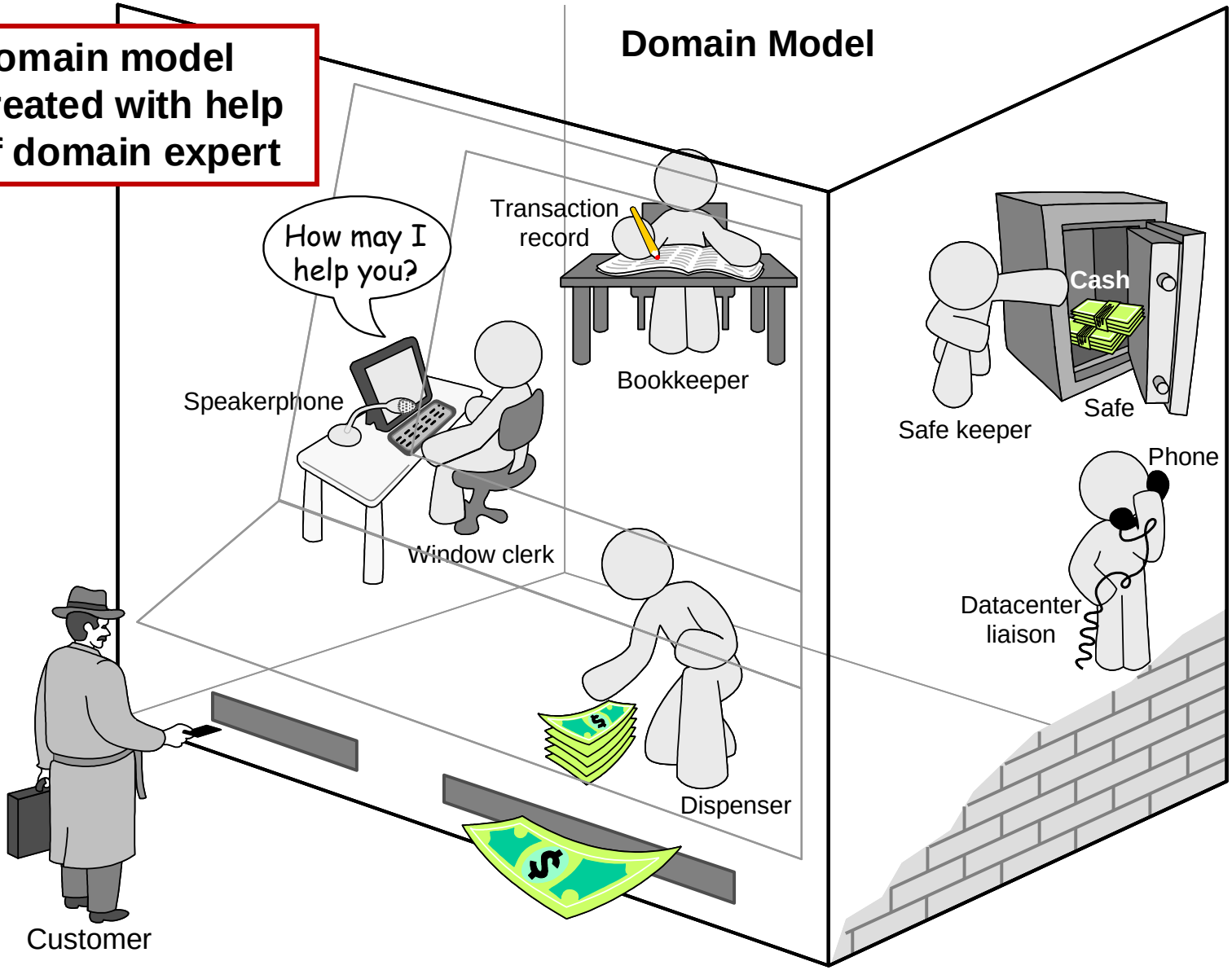
Example: ATM Machine

Understanding the money-machine problem:

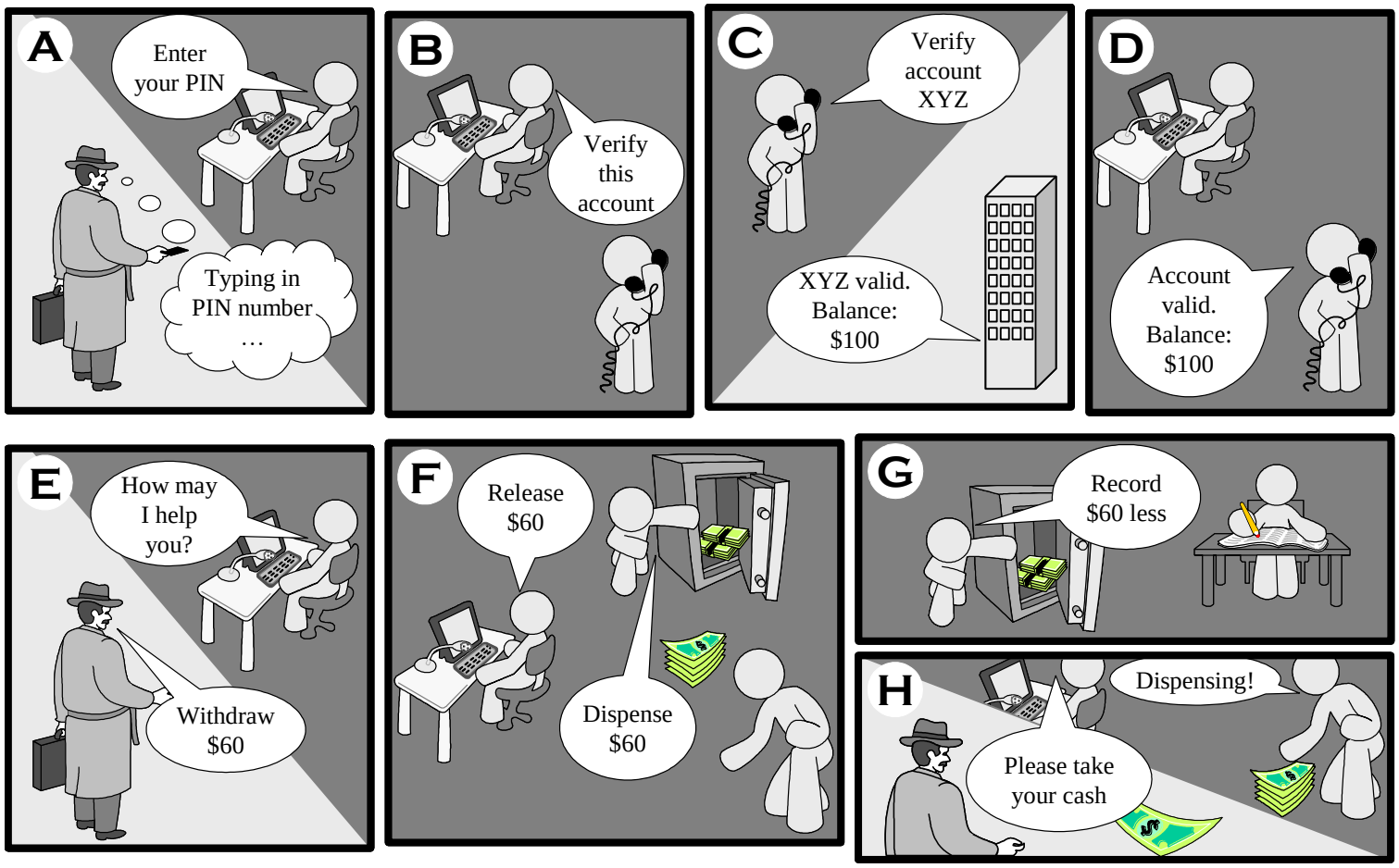


How ATM Machine Might Work

Domain model created with help of domain expert



Cartoon Strip: How ATM Machine Works



Second Law of Software Engineering

- **Software should be written for people first**
  - (Computers run software, but hardware quickly becomes outdated)
  - Useful + good software lives long
  - To nurture software, people must be able to understand it

Software is critical to society

- Economically important
- Essential for running most enterprises
- Key part of most complex systems
- Essential for designing many engineering products
- In many if not in most cases, the software is embedded in the system you are using — you don’t see the software

Need of Software Engineering

1. **Handling Big Projects:** Corporation must use SE to handle large projects without any issues.
2. **To manage the cost:** Software engineering programmers plan everything and reduce all those things that are not required.
3. **To decrease time:** It will save a lot of time if you are developing software using a software engineering technique.
4. **Reliable software:** It is the company's responsibility to deliver software products on schedule and to address any defects that may exist.
5. **Effectiveness:** Effectiveness results from things being created in accordance with the software standards.

Characteristics of Good Software

Operational	Transitional	Maintenance
Budget	Interoperability	Flexibility
Efficiency	Reusability	Maintainability
Usability	Portability	Modularity
Dependability	Adaptability	Scalability
Correctness		
Functionality		
Safety		
Security		

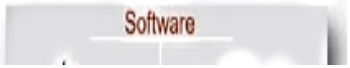
Nature of Software

1. System Software:

- System software is software designed to provide a platform for other software's.
- It is a interaction between hardware & application software.
- **Examples :** Operating Systems like macOS, Linux, Android and Microsoft Windows.

2. Application Software:

- Application Software is a computer program designed to carry out a specific task as per user & business need.
- **Examples:** Social medias apps, Gaming apps, Word processing apps, Multimedia apps, Banking apps, Shopping apps, Booking apps etc.



3. Engineering and Scientific Software:

- This software is used to facilitate the engineering function and task take real time.
- It has very high accuracy, complex formula evolution & data analysis.
- **Examples:** Weather prediction apps, Stock Market apps, Stress Analysis, Body measurement apps

4. Embedded Software:

- Embedded software resides within the system or product and is used to implement and control feature and function for the end-user and for the system itself.
- **Example:** Switches, Routers, Digital camera, Washing machine functionalities, Traffic control etc.

5. Web Applications:

- It is a client-server computer program which the client runs on the web browser.
- Web apps can be little more than a set of linked hypertext files that present information using text and limited graphics.
- **Examples:** Online forms, Shopping carts, Gmail, Yahoo, Photo editing, File conversion etc.

6. Artificial Intelligence Software:

- It makes use of a nonnumerical algorithm to solve a complex problem.
- Application within this area includes robotics, expert system, pattern recognition, artificial neural network, theorem proving and game playing.
- **Examples:** Google Cloud, Azure studio, Tensor Flow, Salesforce etc.

Software Myths

- **Management myths**
  - *Standards and procedures for building software*
  - *Add more programmers if behind schedule*
- **Customer myths**
  - *A general description of objectives enough to start coding*
  - *Requirements may change as the software is flexible*
- **Practitioner myths**
  - *Task accomplished when the program works*
  - *Quality assessment when the program is running*
  - *Working program the only project deliverable*

What is a Software ?

- Requirements and specification documents
- Design documents
- Source Code
- Test suites and test plans
- Interface to hardware and software operating environment
- Documentation, internal and external

What Makes Large Software Different ?

Scale	Precludes total comprehension
Complexity	Number of functions, modules, paths
Team Effort	Continuingly changing body of programmers
Communication	Distribution of specifications and documentation
Continuous Change	During design & implementation and During lifetime of SW Development
Lifetime	Measured in **calendar years** or **man-months**
Imprecise goals	Conflicting or ambiguous, changing requirements

Characteristics of a Good Software?

1. Correct, Correct, & Correct always

2. Maintainable and easy to modify

3. Well modularized with well-designed interfaces

4. Reliable and robust

5. Has a good user interface

6. Well Documented

- internal documentation for maintenance and modification
- external documentation for end users

7. Efficient

- Not wasteful of system resources, cpu & memory
- Optimized data structures and algorithms

Software Qualities

- Correct
- Dependable
- Robustness
- Secure & Safe
- Usable
- Portable

- Maintainable
- Reusable
- Modular
- Understandable
- Interoperable

Why is Software Development Hard?

- Changing requirements and specifications
- Inability to develop complete and correct requirements
- Imprecise and incomplete Requirements and Specifications
- Programmer variability and unpredictability
- Communication and Coordination
- Inadequate Software Development Tools
- Inability to accurately estimate effort or time required
- Overwhelming complexity of large systems, more than linear growth in complexity with size of the system
- Poor software development processes
- Lack of attention to issues of Software Architecture

Ad-hoc software development

- **Ad-hoc development**: creating software without any formal guidelines or process

- what are some disadvantages of ad-hoc development?

- some important actions (testing, design) may go ignored
- not clear when to start or stop doing each task
- does not scale well to multiple people
- not easy to review or evaluate one's work

What is a Software Process?

- Activities and results that produce a software product:

SW Process Activity	What is going on there?
Specification	What does the customer need? What are the constraints?
Development	Design & programming.
Validation	Checking whether it meets requirements.
Evolution	Modifications (e.g. customer/market).

The Meaning of Process

- A **process**: a series of steps involving activities, constraints, and resources that produce an intended output of some kind.

- A process involves a set of tools and techniques

Process Characteristics

- **Prescribes all major process activities**

- Input:
- Uses resources (e.g., customer input, specifications),
- subject to set of constraints (such as schedule, platform reqts)

- Output:
- Produces intermediate and final products (e.g., models)

- Structure:
- May be composed of subprocesses
- with hierarchy or links

- Properties:
- Each process activity has entry and exit criteria
- Activities are organized in sequence, so timing is clear
- Each process guiding principles, including goals of each activity
- Constraints may apply to an activity, resource or product

What is a Process Development?

Process development involves activities, resources, and product

- Process model includes organizational, functional, behavioral and other perspectives

- A process model is useful for guiding team behavior, coordination and collaboration

The Importance of Processes

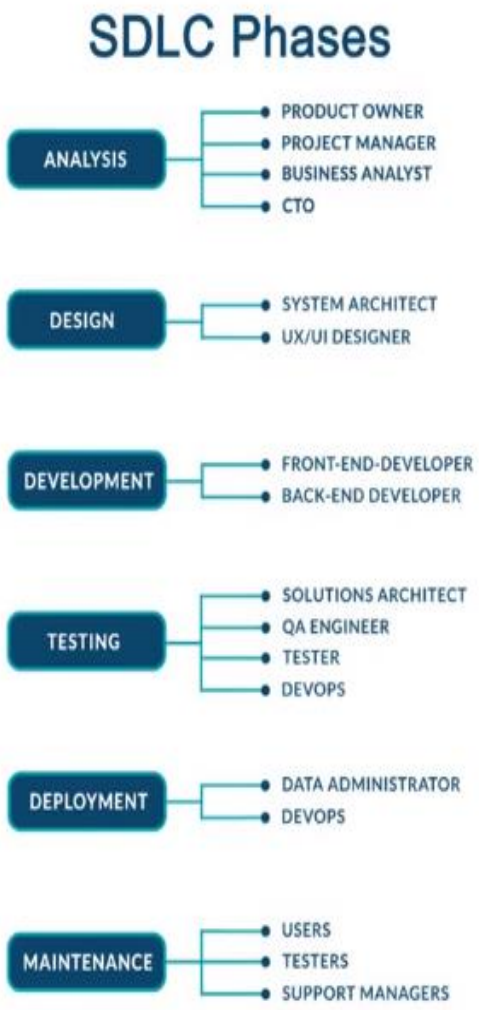
- Impose consistency and structure on a set of activities
- Guide us to understand, control, examine, and improve the activities
- Enable us to capture our experiences and pass them along

Reasons for Modeling a Process

- To form a common understanding
- To find inconsistencies, redundancies, omissions
- To find and evaluate appropriate activities for reaching process goals
- To tailor a general process for a particular situation in which it will be used

Contents

1. SDLC Process
2. SDLC Phases with Examples
3. SDLC Models



➢Need of SDLC Process:

1. Improved relation with client & development team.
2. It offers basic project planning, scheduling & cost estimation.
3. Provide standard set of activities & SE rules.
4. Increase & enhance development speed.
5. Maintain project tracking & control.
6. Decrease any type of project risk.
7. Decide entry & exit criteria of each phase.

SDLC Phase 1: Requirement Analysis & Planning

- Phase conduct by Product manager, Business Analyst or senior members in team.
- To understand the Aim, Purpose & Exact requirements of the customer.
- Conduct market research, customer interviews & surveys.
- Software Requirement Specification (SRS) document is created.
- Identify risk assessment in project.
- Planning of Project schedule, Cost estimation & other recourses.

Example: Online Shopping Application

Customer requirements like:

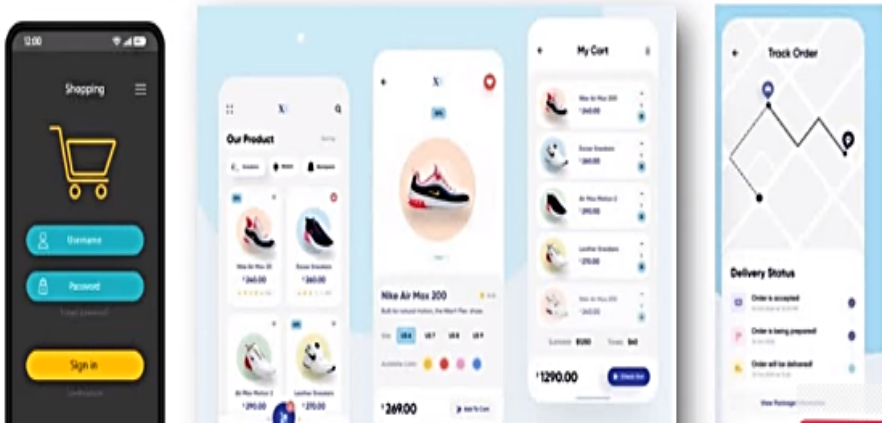
- Categories like Ladies wear, Gents wear module, Grocery module, Shopping cart module, My order module, Discount module, Notification module.
- Good & Attractive user interface.

SDLC Phase 2: Design

- Phase conduct by Product manager & UI / UX Designer team.
- High level design include Software Architecture like brief description of each module, Flow Diagram, Database tables, Interface relationships & Algorithm.
- Low level design include Input & Output of each module, Rough paper design, Detail of Interface, Listing error messages etc.
- Documented in DDS "Design Document Specification".

Example:

Online Shopping Application



SDLC Phase 3: Implementation

- Phase conduct by Developer or Programmer.
- Must follow predefine coding guidelines or discuss with management team.
- Used suitable Programming Languages like C, C++, JAVA, ANDROID, PYTHON, PHP etc.
- Used suitable Databases like SQL, ORACLE etc.
- Programming tools like Compilers, Interpreters, Debuggers etc.

Example:

Online Shopping Application

```
void customerDetails()
{
    cout << "Enter your name: ";
    string customer_name;
    getline(cin,customer_name );
    cout << "WELCOME ";
    for(int i = 0; i<customer_name.length(); i++ )
    {
        cout << char(toupper(customer_name[i] ));
    }
    cout << "\n";
```

SDLC Phase 4: Testing

- Phase conduct by QA Team or Tester.
- It perform Integration Testing, Black Box Testing, White Box Testing, System Testing, Regression Testing Performance Testing & Acceptance Testing etc.
- The testing team starts testing the functionality of the entire system.
- If find some bugs/defects which report to developers then Development team fixes the bug and send back to QA.
- This is done to verify that the entire application works according to the customer requirement.

Example:

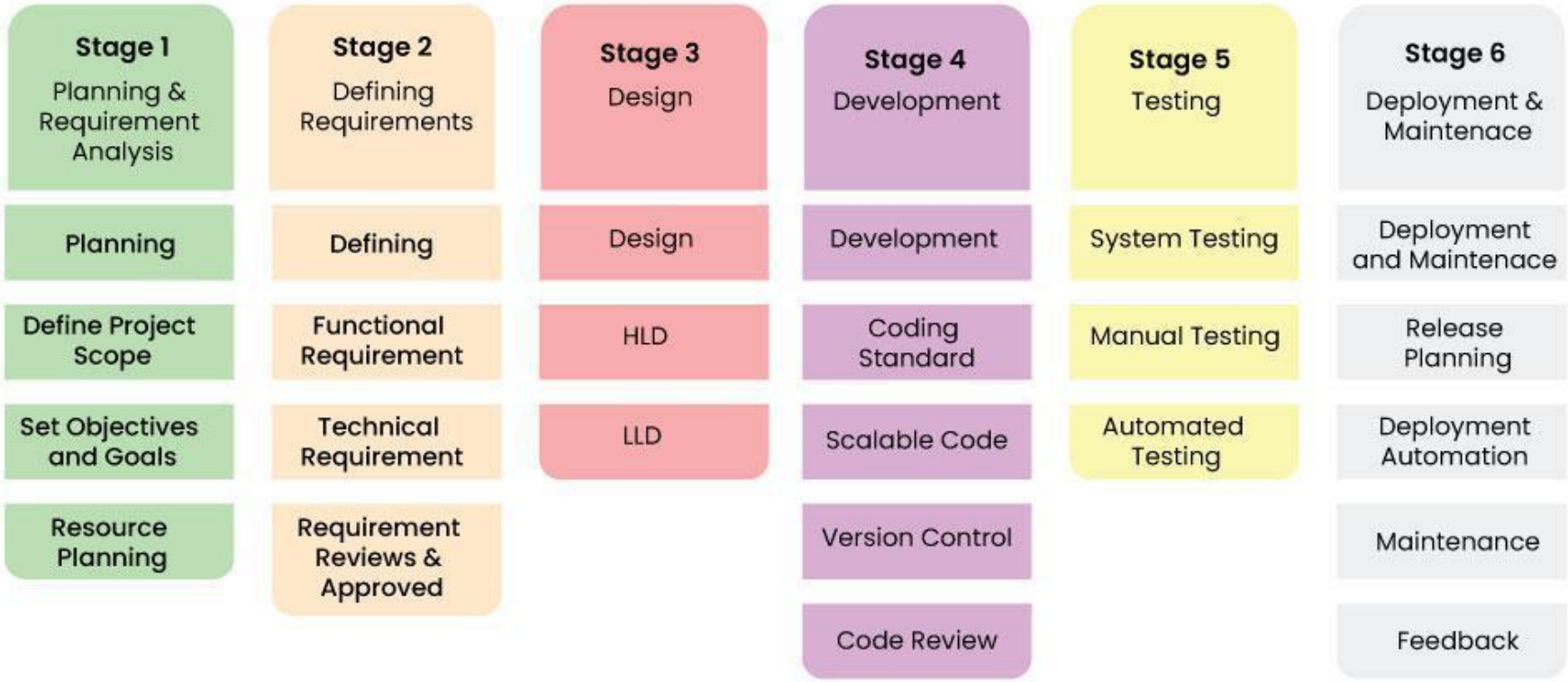
Online Shopping Application

SDLC Phase 5: Deployment & Maintenance

- Phase conduct by Project Manager, Operation or Production support team.
- Once the software is certified and no bugs or errors are stated, then final software is release in market or deployed in customer environment.
- If any issue comes up and needs to be fixed or any updating is to be done is taken care by the Support Managers in Maintenance phase.

Example:

Online Shopping Application


Stage 1
Planning & Requirement Analysis
Planning
Define Project Scope
Set Objectives and Goals
Resource Planning
Stage 2
Defining Requirements
Defining
Functional Requirement
Technical Requirement
Requirement Reviews & Approved
Stage 3
Design
Design
HLD
LLD
Stage 4
Development
Development
Coding Standard
Scalable Code
Version Control
Code Review
Stage 5
Testing
System Testing
Manual Testing
Automated Testing
Stage 6
Deployment & Maintenace
Deployment and Maintenace
Release Planning
Deployment Automation
Maintenance
Feedback

➢ Process Framework Activities

1. **Communication** – Communicate with stakeholders and customers to obtain objectives of the system and requirements for the software.
2. **Planning** – Software project plan has details of resources needed, tasks and risk factors likely to occur, schedule.
3. **Modelling** – Architectural models and design to better understand the problem and for work towards the best solution.
4. **Construction** – Generation of code and testing of the system to rectify errors and ensuring all specified requirements are met.
5. **Deployment** – Entire software product or partially completed product is delivered to the customer for evaluation and feedback.

Process flow

Process flow determines how activities, actions and tasks are arranged with respect to sequence and time.

1. Linear process flow

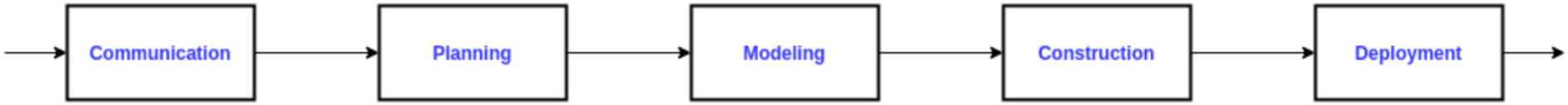
2. Iterative process flow

3. Evolutionary process flow

4. Parallel process flow

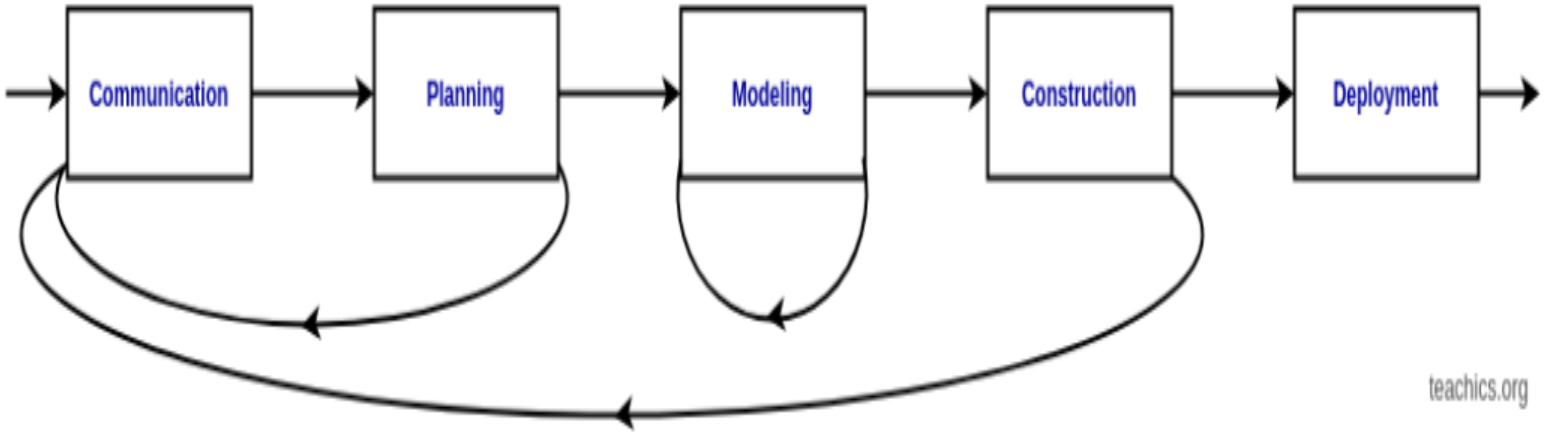
1. Linear process flow

Linear process flow executes each activity in sequence.



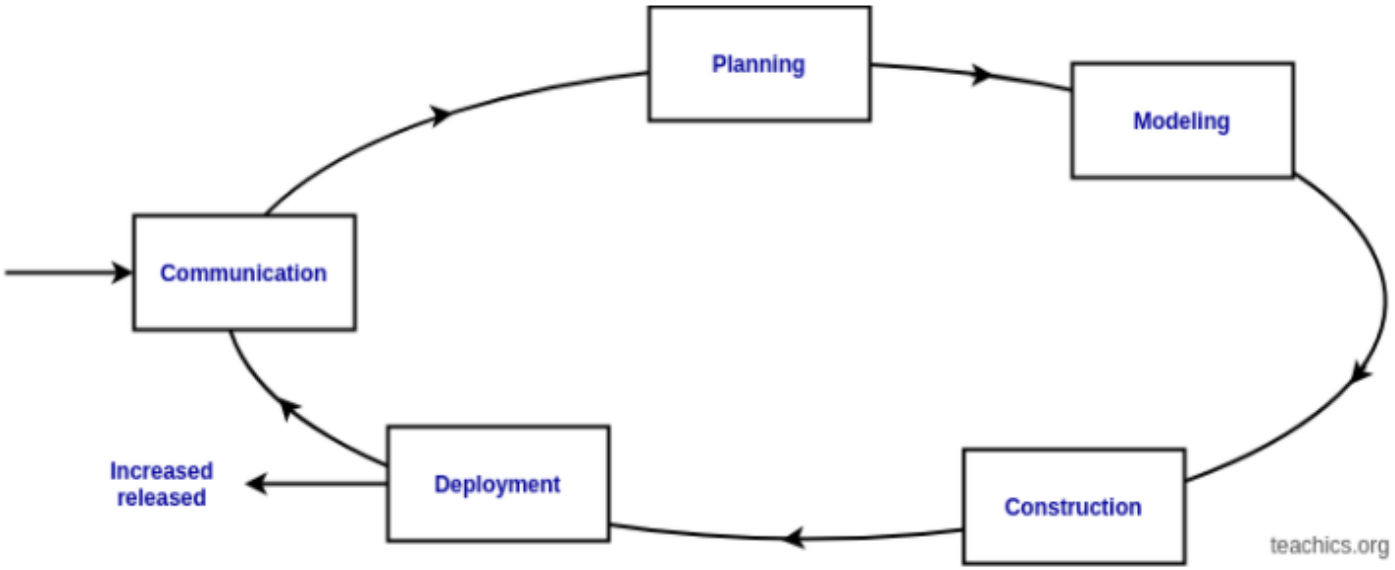
Iterative Process Flow

- Repeats one or more activities before starting next



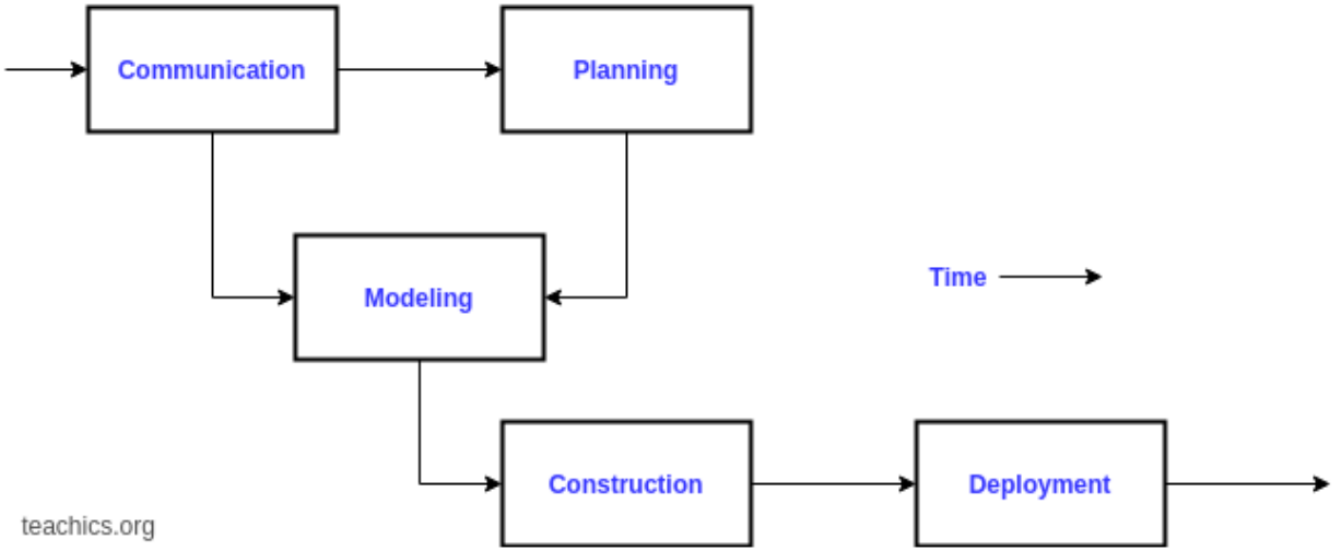
Evolutionary Process Flow

- Carries out activity in a circular way



Parallel Process Flow

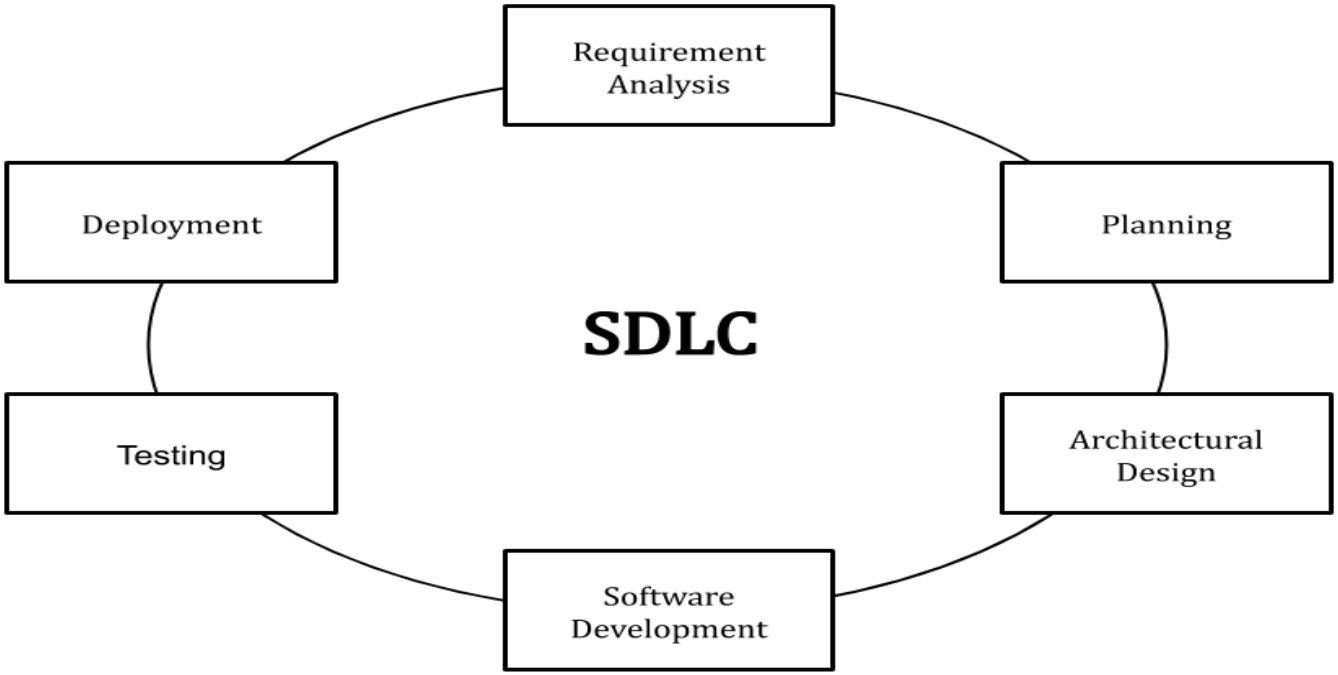
- Executes one or more activities in parallel with each other



Software Process and Software Development Lifecycle Model

- One of the basic notions of the software development process is SDLC models.
- **SDLC** stands for Software Development Life Cycle models.
- There are many development life cycle models that have been developed in order to achieve different required objectives.
- The models specify the various stages of the process and the order in which they are carried out.

Software Process and Software Development Lifecycle Model



Software Process and Software Development Lifecycle Model

- The most used, popular and important SDLC models are given below:

1. Perspective Process Model

a. Waterfall Model

b. V-Model

2. Incremental Process Model

3. Evolutionary Process Model

a. Spiral Model

b. Prototyping Model

c. RAD Model

1. Waterfall Model

- The waterfall model is a breakdown of project activities into **linear sequential phases**, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks. The approach is typical for certain areas of engineering design.

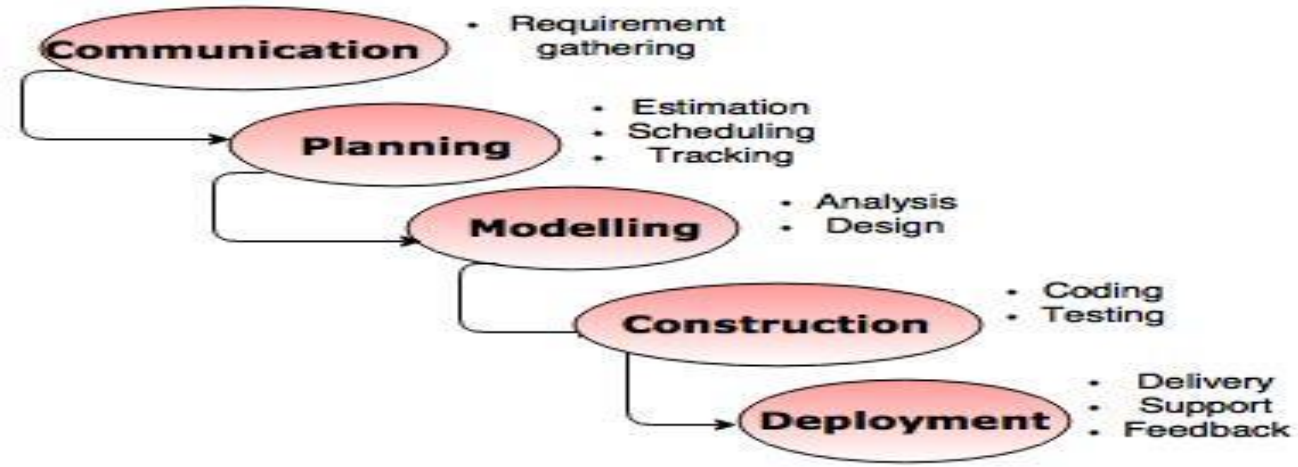


Fig. - The Waterfall model

Advantages of Waterfall Model

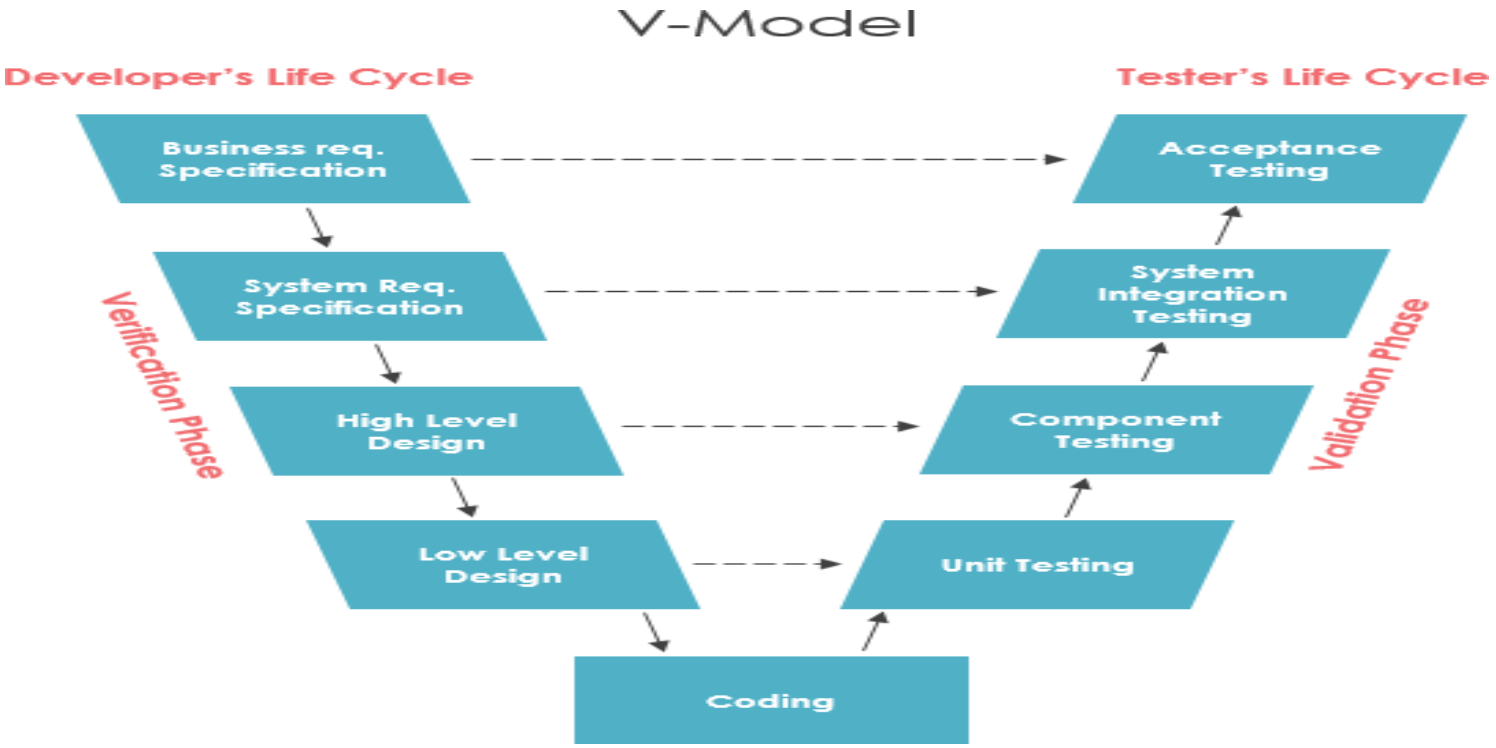
- The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model.
- **Easy to Understand**
- **Individual Processing**
- **Properly Defined**
- **Clear Milestones**
- **Properly Documented**
- **Reinforces Good Habits**

Disadvantages

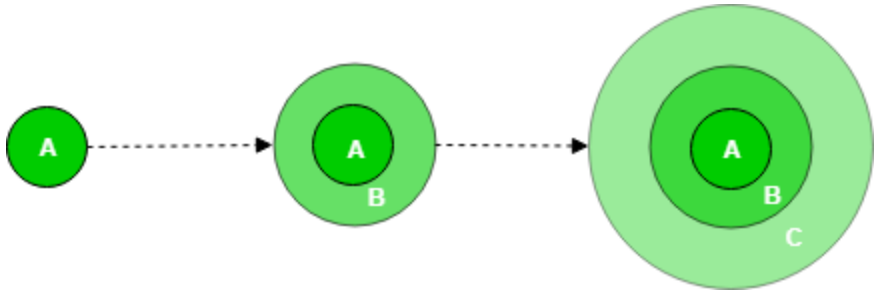
- **No Feedback Path**
- **Difficult to accommodate Change Requests**
- **No Overlapping of Phases**
- **Limited Flexibility**
- **Limited Stakeholder Involvement**
- **Late Defect Detection**
- **Lengthy Development Cycle**

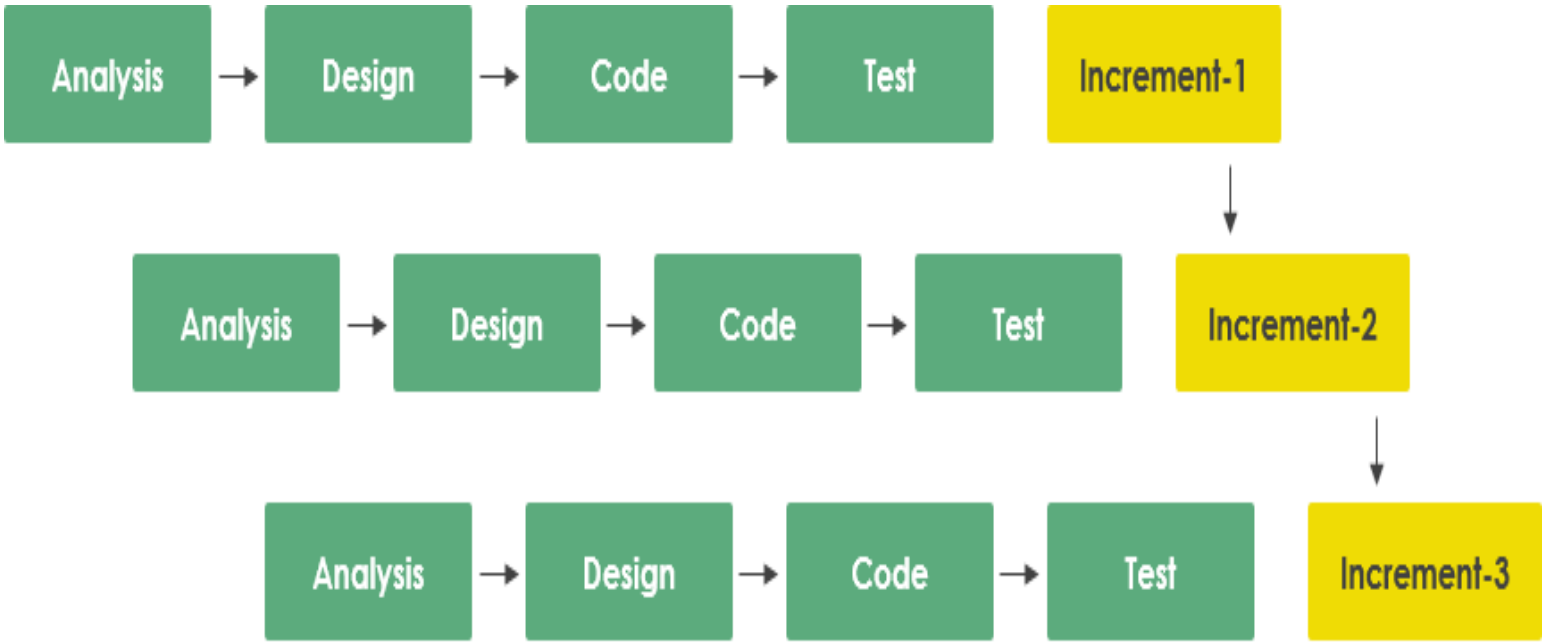
2. V Model

- The V-model represents a development process that may be considered **an extension of the waterfall model** and is an example of the more general V-model.
- Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape. The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.



II. Incremental model





Incremental Model

II. Incremental model

- The incremental build model is a method of software development where the model is designed, implemented and tested incrementally (a little more is added each time) until the product is finished.
- It involves both development and maintenance.
- The product is defined as finished when it satisfies all of its requirements.
- Each iteration passes through the requirements, design, coding and testing phases.
- And each subsequent release of the system adds function to the previous release until all designed functionally has been implemented.
- This model combines the elements of the waterfall model with the iterative philosophy of prototyping.

When to use the Incremental Process Model

- Funding Schedule, Risk, Program Complexity, or need for early realization of benefits.
- When Requirements are known up-front.
- When Projects have lengthy development schedules.
- Projects with new Technology.
- Error Reduction (core modules are used by the customer from the beginning of the phase and then these are tested thoroughly).
- Uses divide and conquer for a breakdown of tasks.
- Lowers initial delivery cost.
- Incremental Resource Deployment.

Advantages of the Incremental Process Model

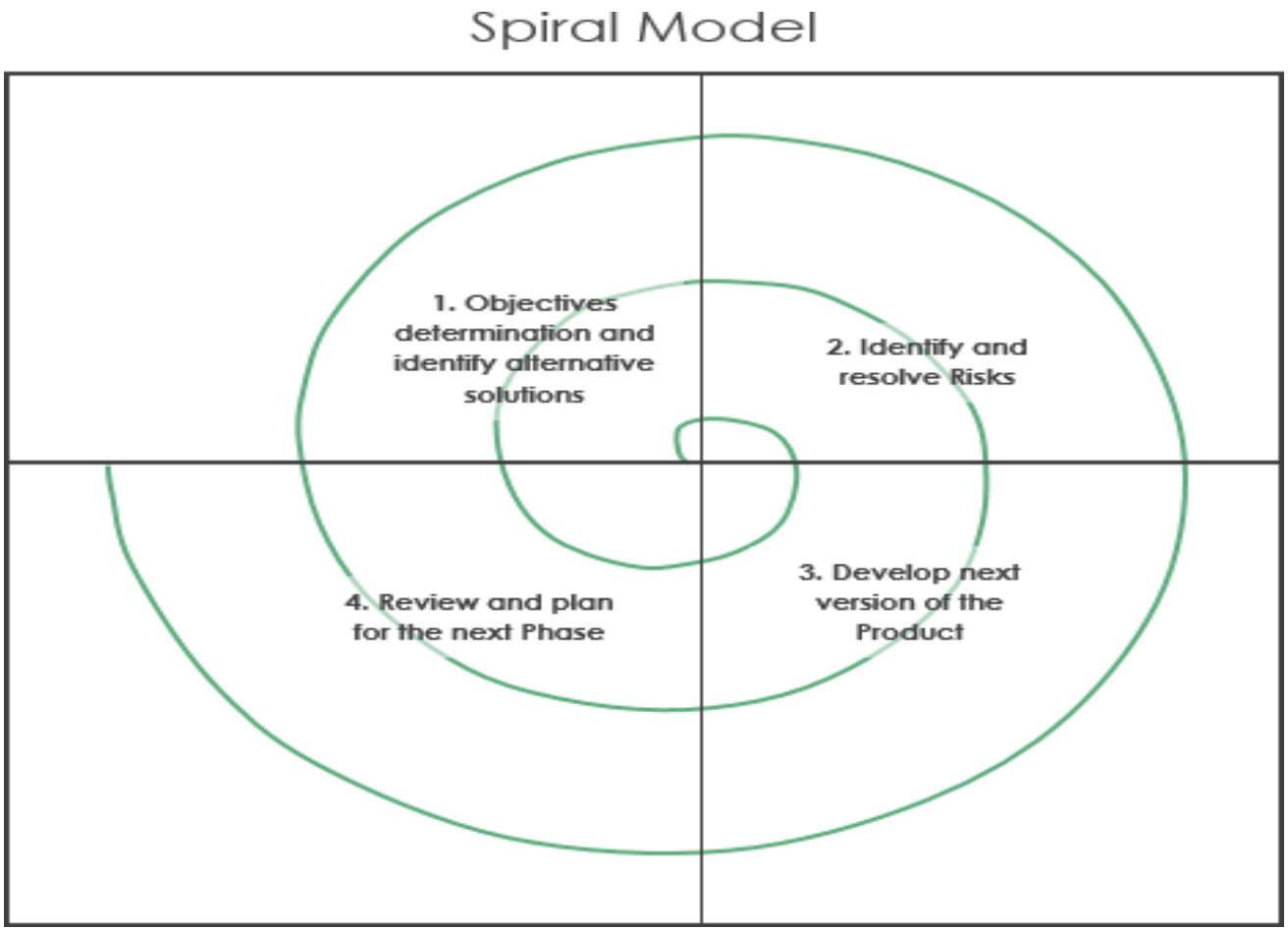
1. Prepares the software fast.
2. Clients have a clear idea of the project.
3. Changes are easy to implement.
4. Provides risk handling support, because of its iterations.
5. Adjusting the criteria and scope is flexible and less costly.
6. Comparing this model to others, it is less expensive.
7. The identification of errors is simple.

Disadvantages of the Incremental Process Model

1. A good team and proper planned execution are required.
2. Because of its continuous iterations the cost increases.
3. Issues may arise from the system design if all needs are not gathered upfront throughout the program lifecycle.
4. Every iteration step is distinct and does not flow into the next.
5. It takes a lot of time and effort to fix an issue in one unit if it needs to be corrected in all the units.

3. Evolutionary Process Model

1. Spiral model



Spiral Model

- The spiral model, first described by Barry Boehm in 1986, is a **risk-driven** software development process model which was introduced for dealing with the shortcomings in the traditional waterfall model.
- A spiral model looks like a spiral with many loops.
- The exact number of loops of the spiral is unknown and can vary from project to project.
- This model supports risk handling, and the project is delivered in loops.
- Each loop of the spiral is called a Phase of the software development process.
- The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks.
- As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using a spiral model.

Quadrants of Spiral Model

1. **Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

2. **Identify and resolve Risks:** During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.

3. **Develop the next version of the Product:** During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

4. **Review and plan for the next Phase:** In the fourth quadrant, the Customers evaluate the so-far developed version of the software. In the end, planning for the next phase is started.

Example of Spiral Model: Developing an E-Commerce Website

- **First Spiral** – The initial phase involves gathering basic requirements for the e-commerce website, like product listing, shopping cart, and payment options. The team analyzes any risks, such as security or scalability, and creates a small prototype. Plan next phase.
    - **Example:** The team builds a simple homepage with a basic product catalog to see how users interact with it and identify any design flaws.
- **Second Spiral** –After gathering feedback from the prototype, the next spiral focuses on adding more features and fixing early issues. The team addresses security risks, such as secure payment processing, and tests how well the site handles increasing user traffic.
    - **Example:** A basic shopping cart and user registration system are added. The payment system is also tested with dummy transactions to ensure security.

- **Third Spiral** – With more feedback, the team further refines the design, adding advanced features like order tracking, customer reviews, and search functionality. Risks like scalability (handling many users) are re-evaluated, and more testing is conducted.
    - **Example:** The website now supports user profiles, product reviews, and real-time inventory updates. The team tests how the system handles large volumes of orders during peak times.
- **Final Spiral** – The final phase involves full implementation, thorough testing, and launching the e-commerce website to the public. Ongoing risks like system crashes or user feedback are monitored and addressed as needed.
    - **Example:** The website goes live with all features, including secure payments, product listings, and order tracking, ready for users to shop online.

Advantages of the Spiral Model

1. Risk Handling
2. Good for large projects
3. Flexibility in Requirements
4. Customer Satisfaction
5. Iterative and Incremental Approach

Disadvantages of the Spiral Model

1. **Complex:** The Spiral Model is much more complex than other SDLC models.

2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.

3. **Too much dependability on Risk Analysis:** The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.

4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, time estimation is very difficult.

6. **Time-Consuming:** The Spiral Model can be time-consuming, as it requires multiple evaluations and reviews.

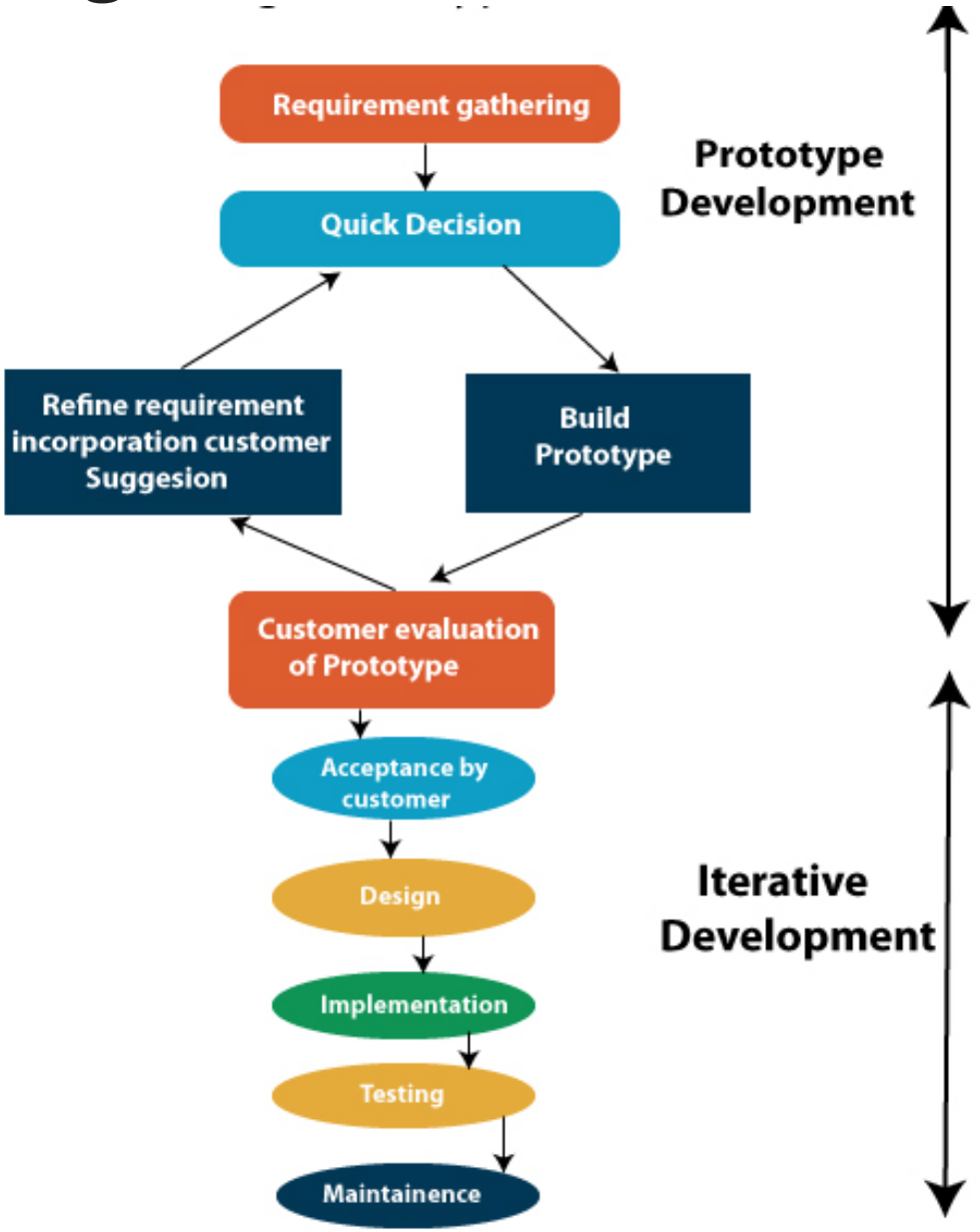
7. **Resource Intensive:** The Spiral Model can be resource-intensive, as it requires a significant investment in planning, risk analysis, and evaluations.

When To Use the Spiral Model?

1. When a project is vast
2. A spiral approach is utilized when frequent releases are necessary.
3. When it is appropriate to create a prototype
4. When evaluating risks and costs is crucial
5. The spiral approach is beneficial for projects with moderate to high risk.
6. The SDLC's spiral model is helpful when requirements are complicated and ambiguous.
7. If modifications are possible at any moment
8. When committing to a long-term project is impractical owing to shifting economic priorities.

Prototyping Model

- Before development working prototype of the system should be built.
- Toy implementation of the system.
- Very crude version of the actual system,
- Exhibits limited functional capabilities
- Low reliability
- Inefficient performance as compared to actual software.



Prototyping Model has following six SDLC phases as follow

- **Step 1: Requirements gathering and analysis**

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**

In this phase, 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**

In this stage, 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 suggestions.

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.

Advantage of Prototype Model

- Reduce the risk of incorrect user requirement
- Good where requirement are changing/uncommitted
- Regular visible process aids management
- Support early product marketing
- Reduce Maintenance cost
- Errors can be detected much earlier as the system is made side by side

Disadvantages of Prototype Model

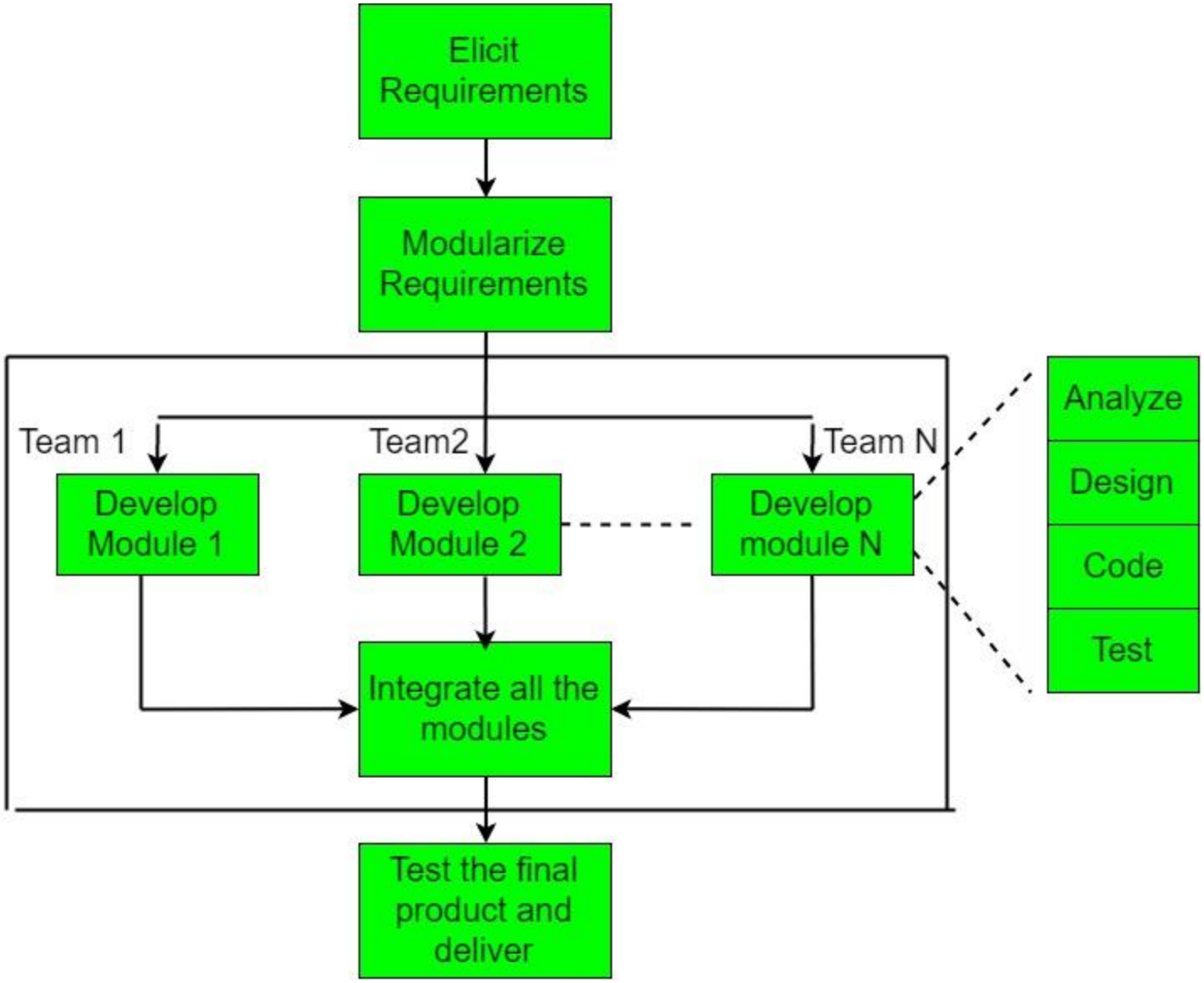
- An unstable/badly implemented prototype often becomes the final product.
- Require extensive customer collaboration
- Costs customer money
- Needs committed customer
- Difficult to finish if customer withdraws
- May be too customer specific, no broad market
- Difficult to know how long the project will last.
- Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
- Prototyping tools are expensive.
- Special tools & techniques are required to build a prototype.
- It is a time-consuming process.

When to use Prototyping Model

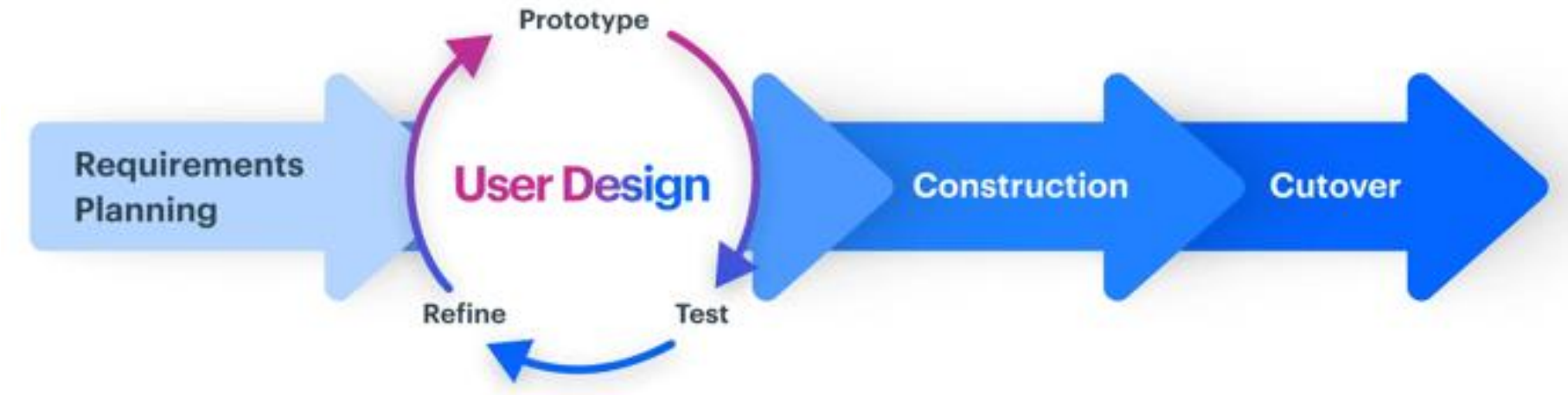
- The Prototyping Model should be used when the requirements of the product are not clearly understood or are unstable.
- The prototyping model can also be used if requirements are changing quickly.
- This model can be successfully used for developing user interfaces, high-technology software-intensive systems, and systems with complex algorithms and interfaces.
- The prototyping Model is also a very good choice to demonstrate the technical feasibility of the product.

RAD Model

- Project can be broken down into small modules wherein each module can be assigned independently to separate teams.
- These modules can finally be combined to form the final product.
- Development of each module involves the various basic steps as in the waterfall model i.e. analyzing, designing, coding, and then testing, etc.
- Time frame for delivery(time-box) is generally 60-90 days.
- Use of Reusable components


Elicit Requirements
Modularize Requirements
Team 1
Team2
Team N
Develop Module 1
Develop Module 2
Develop module N
Integrate all the modules
Test the final product and deliver
Analyze
Design
Code
Test

Phases of RAD



Phases of RAD

1. **Requirements Planning** – This involves the use of various techniques used in requirements elicitation like brainstorming, task analysis, form analysis, user scenarios, FAST (Facilitated Application Development Technique), etc. It also consists of the entire structured plan describing the critical data, methods to obtain it, and then processing it to form a final refined model.

2. **User Description** – This phase consists of **taking user feedback** and building the prototype using developer tools. In other words, it includes re-examination and validation of the data collected in the first phase. The dataset attributes are also identified and elucidated in this phase.

Phases of RAD

3. **Construction** – In this phase, refinement of the prototype and delivery takes place. It includes the actual use of powerful automated tools to transform processes and data models into the final working product. All the required modifications and enhancements are to be done in this phase.

4. **Cutover** – All the interfaces between the independent modules developed by separate teams have to be tested properly. The use of powerfully automated tools and subparts makes testing easier. This is followed by acceptance testing by the user.

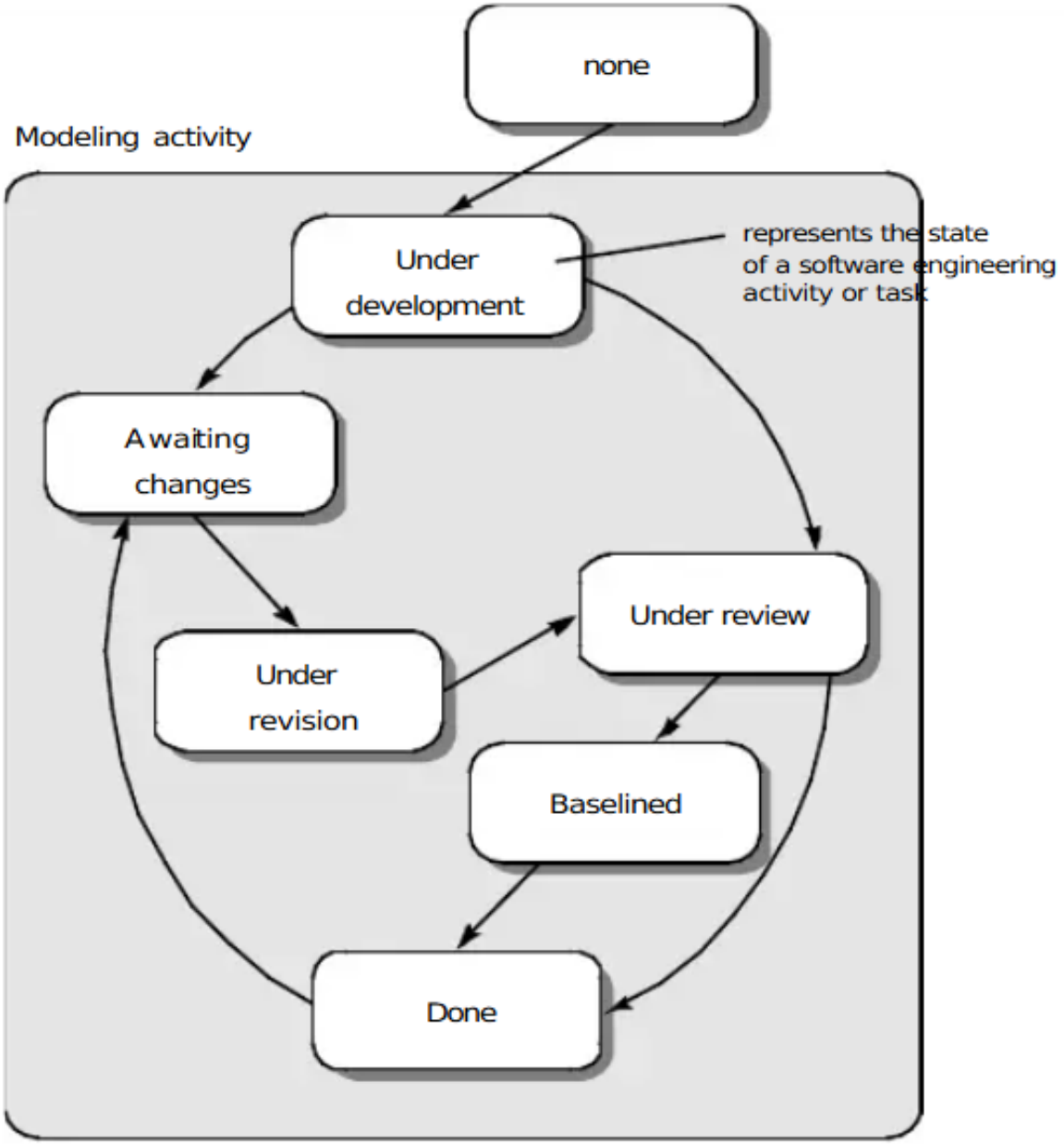
Advantages of RAD

- The use of reusable components helps to reduce the cycle time of the project.
- Feedback from the customer is available at the initial stages.
- The use of powerful development tools results in better quality products in comparatively shorter periods.
- It is easier to accommodate changing requirements due to the short iteration time spans.

Disadvantages of RAD

- The use of powerful and efficient tools requires highly skilled professionals.
- The absence of reusable components can lead to the failure of the project.
- The team leader must work closely with the developers and customers to close the project on time.
- The systems which cannot be modularized suitably cannot use this model.
- Customer involvement is required throughout the life cycle.
- It is not meant for small-scale projects as in such cases, the cost of using automated tools and techniques may exceed the entire budget of the project.
- Not every application can be used with RAD.

The Concurrent Development Model



Key Features of the Concurrent Development Model

1. **Parallelism**:

   Multiple activities (e.g., requirements gathering, design, coding, and testing) are carried out concurrently rather than sequentially.

   This **reduces development time** and allows teams to work on different parts of the system simultaneously.

2. **State-Based Progression**:

   Each activity is represented as a "state" (e.g., under development, under review, or approved).

   The state of one activity can influence or trigger progress in another.

The concurrent development model

- 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.
- The concurrent process model activities moving from one state to another state

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.

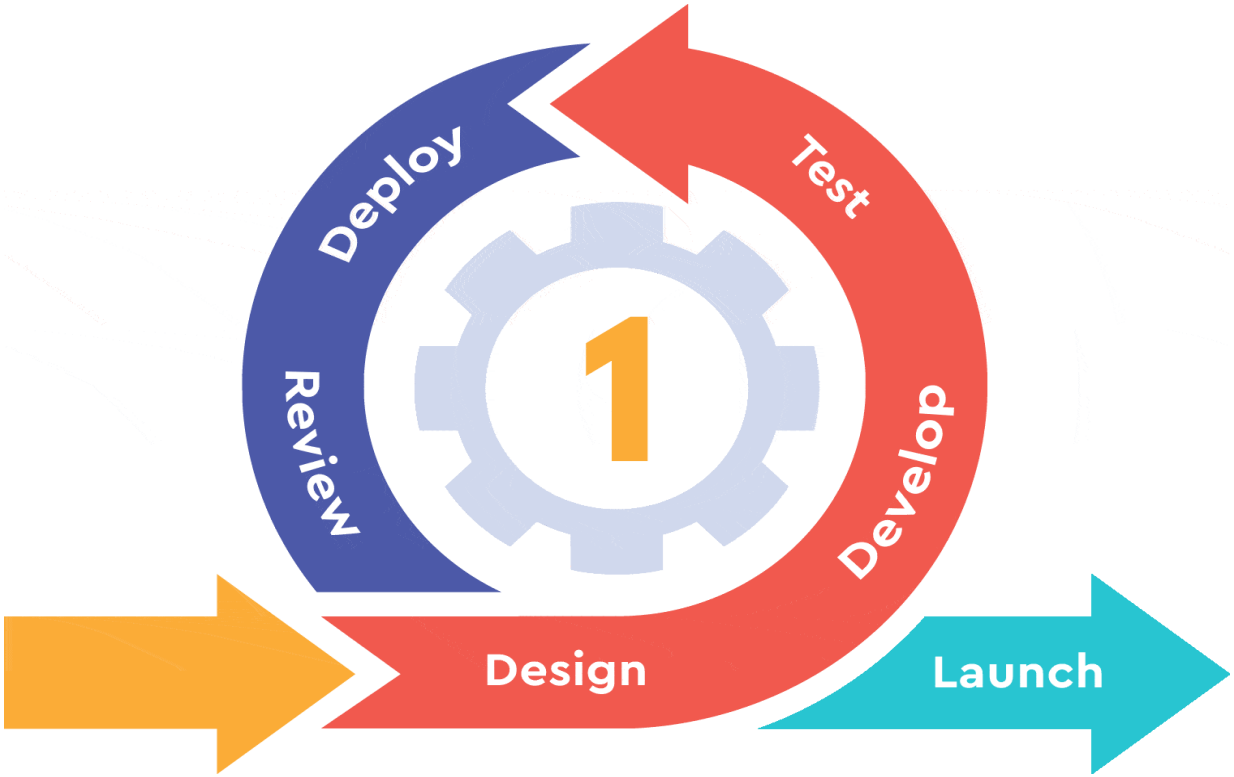
Disadvantages of the concurrent development model

- 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.

Agile Model

- **The Agile Model** was primarily designed to help a project adapt quickly to change requests.
- Main aim of the Agile model is to **facilitate quick project** completion.
- To accomplish this task, **agility** is required. Agility is achieved by **fitting the process to the project** and **removing activities** that may not be essential for a specific project.
- Agility in software engineering is the ability of a development team to **respond quickly** to changing requirements and deliver quality products on time
- Anything that is a waste of time and effort is avoided.

Agile Model



Agile Manifesto

4 Core Values

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

12 Principles

- Customer Satisfaction
- Changing Requirement
- Frequent Delivery
- Promoting Collaboration
- Motivated Individuals
- Face to Face Communication
- Maintain Constant Pace
- Measure Progress
- Technical Excellence
- Simplicity
- Self Organized Teams
- Continuous Improvements

Agile model	Incremental development model
The agile model is an incremental delivery process where each incremental delivered part is developed through an iteration after each time box. The main principle of the Agile model is to achieve agility by **removing unnecessary activities that waste time and effort.**	The requirements of the software are divided into several modules that can be incrementally developed and delivered. **The core features are developed first** and the whole software is developed by adding new features in successive versions.
In the Agile model, **the end date for an iteration is fixed**, it cannot be changed. The development team may have to decide to reduce the delivered functionality to complete that iteration on time.	In the Incremental development model, there is **no fixed time** to complete the next iteration.

When to use the Agile Model?

- When frequent changes are required.
- When a highly qualified and experienced team is available.
- When a customer is ready to have a meeting with a software team all the time.
- When project size is small.

Advantage(Pros) of Agile Method:

1. Frequent Delivery

2. Face-to-Face Communication with clients.

3. Anytime changes are acceptable.

4. It reduces total development time.

Disadvantages(Cons) of Agile Model:

1. Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
2. Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

Agile Testing Methods

1. Scrum
2. Crystal
3. Dynamic Software Development Method (DSDM)
4. Feature Driven Development(FDD)
5. Lean Software Development
6. Extreme Programming(XP)

Scrum

- Scrum is a management framework that teams use to self-organize and work towards a common goal.
- Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members).
- It describes a set of meetings, tools, and roles for efficient project delivery.
- Much like a sports team practicing for a big match, Scrum practices allow teams to self-manage, learn from experience, and adapt to change.
- Software teams use Scrum to solve complex problems cost effectively and sustainably.

Time Box

- The time required to complete an iteration is known as a **Time Box**.
- Time-box refers to the **maximum amount** of time needed to deliver an iteration to customers.
- So, the **end date** for an iteration does not change.
- However, the development team can decide to reduce the delivered functionality during a Time-box if necessary to deliver it on time.
- The Agile model's central principle is delivering an increment to the customer after each Time-box.

- The essence of Scrum is a self-organizing team delivering customer value in a time-boxed period called a *Sprint*.
- Scrum defines artifacts, roles, and events associated with each Sprint.

What are Scrum artifacts?

- Scrum Teams use tools called **Scrum artifacts** to solve problems and manage projects.
- Scrum artifacts provide critical planning and task information to team members and stakeholders.
- There are three primary artifacts:
  - Product Backlog
  - Sprint Backlog
  - Increment

Product Backlog

- The Product Backlog is a dynamic list of features, requirements, enhancements, and fixes that must be completed for project success.
- It is essentially the team's to-do list, which is constantly revisited and reprioritized to adapt to market changes.
- The product owner maintains and updates the list, removing irrelevant items or adding new requests from customers.

Sprint Backlog

- The Sprint Backlog is the list of items to be completed by the development team in the current Sprint cycle.
- Before each Sprint, the team chooses which items it will work on from the Product Backlog.
- A Sprint Backlog is flexible and can evolve during a Sprint.

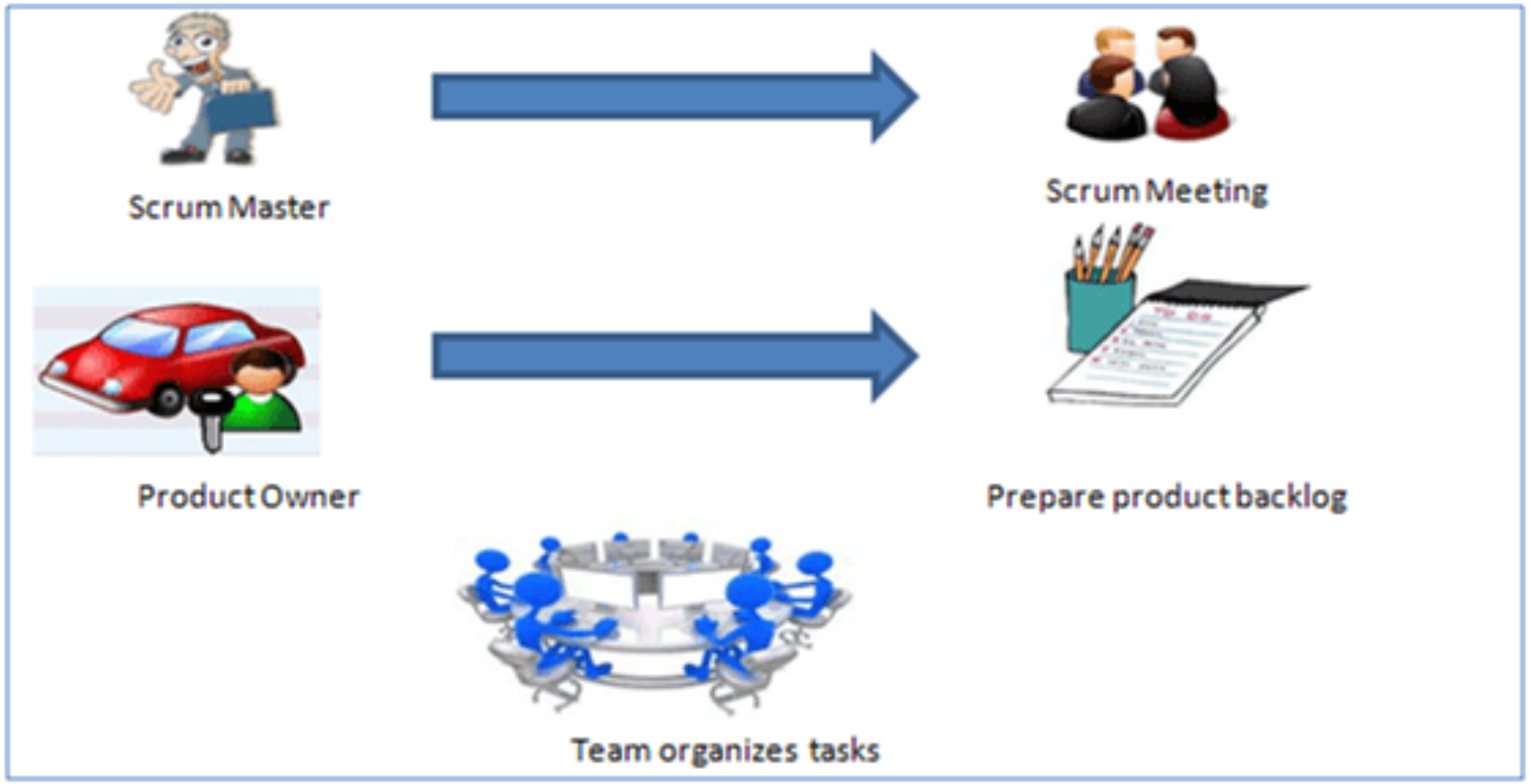
Increment

- The Increment is a step towards a goal or vision.
- **It is the usable end product from a Sprint.**
- Teams can adopt different methods to define and demonstrate their Sprint Goals.
- Despite the flexibility, the fundamental Sprint Goal—what the team wants to achieve from the current Sprint—can't be compromised.

For example, some teams choose to release something to their customers at the end of the Sprint, so their Sprint Goal would be completed once the software change is released. Other teams might work on completing a set of features that will be released together. In this case, the Sprint Goal would be completed when a feature is tested successfully.

Scrum Roles

Product Owner, Scrum leader/Master, and Development Team



Product Owner

- The Product Owner focuses on ensuring the development team delivers the most value to the business.
- They understand and prioritize the changing needs of end users and customers.
- Effective product owners do the following:
  - Give the team clear guidance on which features to deliver next.
  - Bridge the gap between what the business wants and what the team understands.
  - Decide when and how frequently releases should happen.

Scrum Leader/Master

- Scrum leaders are the champions for Scrum within their teams.
- They are accountable for the Scrum Team's effectiveness.
- They coach teams, Product Owners, and the business to improve its Scrum processes and optimize delivery.
- Scrum leaders are also responsible for doing the following:
  - Schedule the resources needed for each Sprint.
  - Facilitate other Sprint events and team meetings.
  - Lead digital transformation within the team.
  - Facilitate any team training when adopting new technologies.
  - Communicate with external groups to solve any challenges the team might be facing as a whole.

Scrum development team

- The Scrum Team consists of testers, designers, UX specialists, Ops engineers, and developers.
- Team members have different skill sets and cross-train each other, so no one person becomes a bottleneck in delivering work.
- **Jeff Bezos, the founder of Amazon, recommends the two-pizza rule when deciding team size: A team should be small enough to share two pizzas.**
- Scrum development teams do the following:
  - Work collaboratively to ensure a successful Sprint completion.
  - Champion sustainable development practices.
  - Self-organize and approach their projects with an evident we attitude.
  - Drive the planning and estimating for how much work they can complete for each Sprint.

Agile and Scrum responsibilities

1. **Scrum Master**
   - Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
2. **Product owner**
   - The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
3. **Scrum Team**
   - Team manages its own work and organizes the work to complete the sprint or cycle

Product Backlog

- This is a repository where requirements are tracked with details on the no of requirements(user stories) to be completed for each release.
- It should be maintained and prioritized by Product Owner, and it should be distributed to the scrum team.
- Team can also request for a new requirement addition or modification or deletion

Scrum Events

Scrum events or Scrum ceremonies are a set of sequential meetings that Scrum Teams perform regularly.

Some Scrum events include the following:

- **Sprint Planning**

In this event, the team estimates the work to be completed in the next Sprint. Members define Sprint Goals that are specific, measurable, and attainable. At the end of the planning meeting, every Scrum member knows how each Increment can be delivered in the Sprint.

- **Sprint**

A Sprint is the actual time period when the Scrum Team works together to finish an Increment. Two weeks is the typical length for a Sprint but can vary depending on the needs of the project and the team. The more complex the work and the more unknowns, the shorter the Sprint should be.

- **Daily Scrum or stand-up**

A Daily Scrum is a short meeting in which team members check in and plan for the day. They report on work completed and voice any challenges in meeting Sprint Goals. It is called a stand-up because it aims to keep the meeting as short as practical—like when everybody is standing.

- **Sprint Review**

At the end of the Sprint, the team gets together for an informal session to review the work completed and showcase it to stakeholders. The Product Owner might also rework the Product Backlog based on the current Sprint.

- **Sprint Retrospective**

The team comes together to document and discuss what worked and what didn't work during the Sprint. Ideas generated are used to improve future Sprints.

Extreme Programming (XP)

- The Extreme Programming is commonly used agile process model.
- It uses the concept of object-oriented programming.
- A developer focuses on the framework activities like planning, design, coding and testing. XP has a set of rules and practices.

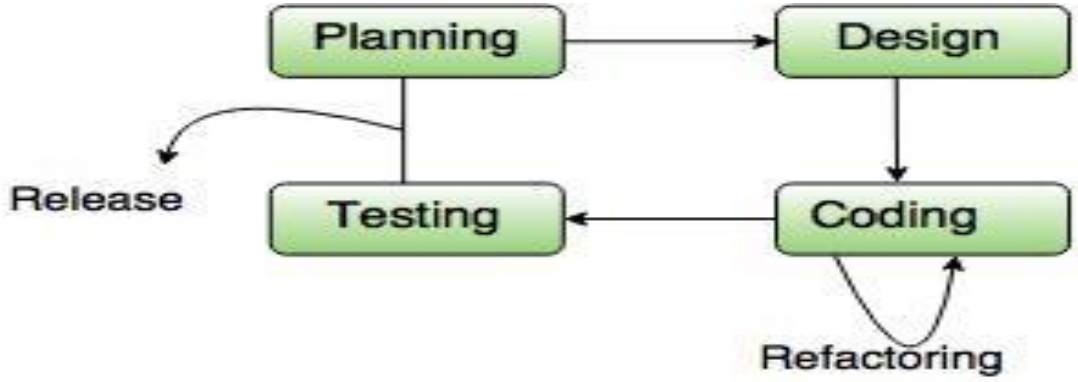


Fig. - The Extreme Programming Process

XP values

- **Following are the values for extreme programming:**

1. Communication:

- Building software development process needs communication between the developer and the customer.
- Communication is important for requirement gathering and discussing the concept.

- **2) Simplicity**
  The simple design is easy to implement in code.

  3. Feedback
  Feedback guides the development process in the right direction.

  4. Courage
  In every development process there will always be a pressure situation. The courage or the discipline to deal with it surely makes the task easy.

  5. Respect
  Agile process should inculcate the habit to respect all team members, other stake holders and customer.

The XP Process

- **The XP process comprises four framework activities:**

1. Planning

- Planning starts with the requirements gathering which enables XP team to understand the rules for the software.
- The customer and developer work together for the final requirements.

2. Design

The XP design follows the 'keep it simple' principle.

- A simple design always prefers the more difficult representation.

3. Coding

The coding is started after the initial design work is over.

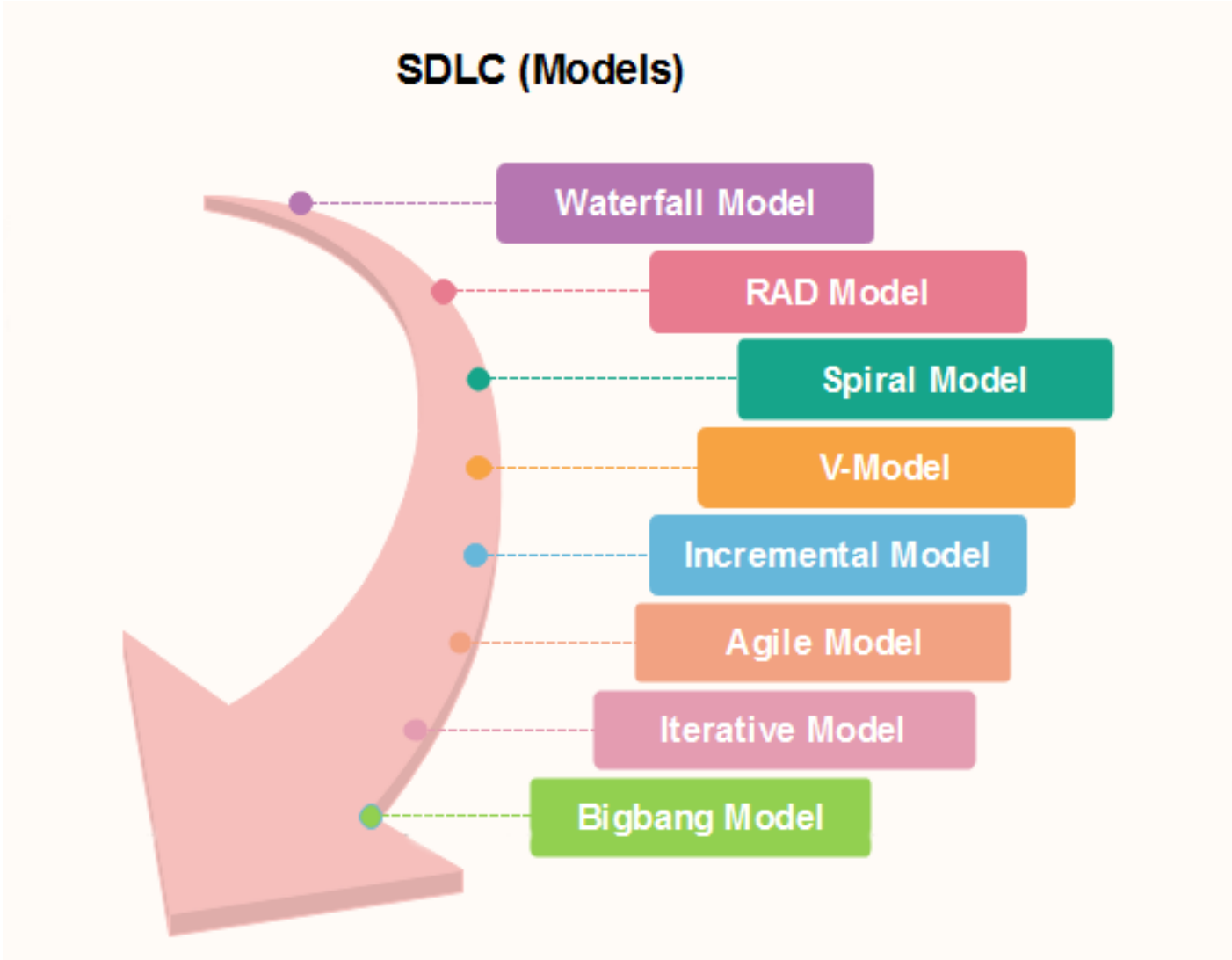
- After the initial design work is done, the team creates a set of unit tests which can test each situation that should be a part of the release.
- The developer is focused on what must be implemented to pass the test.
- Two people are assigned to create the code. It is an important concept in coding activity.

4. Testing

Validation testing of the system occurs on a daily basis. It gives the XP team a regular indication of the progress.

- 'XP acceptance tests' are known as the customer test.

Parameter / Process Model→	Waterfall Model	Incremental Model	Prototype Model	Rad Model	Spiral Model	Agile Model	Xp programming
Clear Requirement Specifications	Initial level	Initial level	At medium level	Initial level	Initial level	Change incrementally	Initial level
Feedback from user	No	No	Yes	No	No	No	Yes
Speed to change	Low	High	Medium	No	High	High	High
Predictability	Low	Low	High	Low	Medium	High	High
Risk identification	At initial level	No	No	No	Yes	Yes	Yes
Practically implementation	No	Low	Medium	No	Medium	High	High
Loom	Systematic sequence	Iterative sequence	Priority on customer feedback	Use readymade component	Identification of risk at each stage	Highly customer satisfaction and incremental development[09]	Customer satisfaction and incremental development
Any variation done	Yes-v model	No	No	No	Yes-win win spiral[6]	No	No
Understandability	Simple	Intermediate	Intermediate	Intermediate	Hard	Much complex	Intermediate
Precondition	Requirement clearly defined	Core product should clearly define	Clear idea of Quick Design	Clean idea of Reuse component	No	No	No
Usability	Basic	Medium	High	Medium	Medium	Most use now a days	medium
Customer priority	Nil	Nil	Intermediate	Nil	Intermediate	High	Intermediate
Industry approach	Basic	Basic	Medium	Medium	Medium	High	Medium
Cost	Low	Low	High	very high	Expensive	Much Expensive	High
Resource organization	Yes	Yes	Yes	Yes	No	No	Yes
Elasticity	No	No	Yes	Yes	No	Very high	Medium


SDLC (Models)
Waterfall Model
RAD Model
Spiral Model
V-Model
Incremental Model
Agile Model
Iterative Model
Bigbang Model

Frequently asked questions about software engineering

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

Frequently asked questions about software engineering

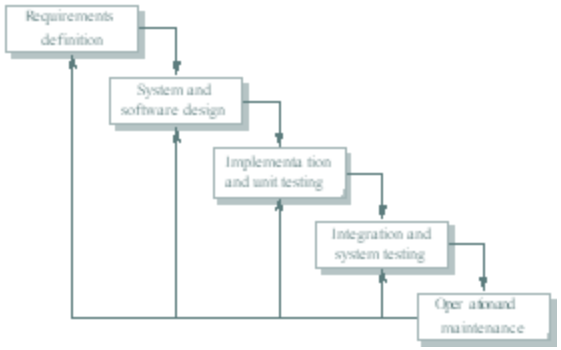
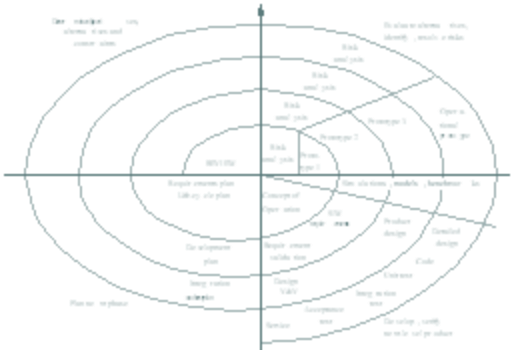
Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

What is a Software Process Model?

- Description of the software process that represents one view, such as the activities, data or roles of people involved.

Examples of views	Focus on…
Workflow	Activities = human actions. What is input, output, and dependencies.
Dataflow	Activities = transformations of information. How the input is transformed into output.
Role/Action	What is the role of people involved in each step of the process?

Software Process Models

Waterfall approach	Iterative development	Component-Based Software Engineering CBSE
		*assembled form existing components*

The Cost of Software Engineering

- Depends on:
  - The **process** used, and
  - The **type** of software being developed.
- Each generic approach has a different profile of cost distribution.
- Roughly **60%** of costs are development costs, **40%** are testing costs.
- For **custom software**, evolution costs often exceed development costs.

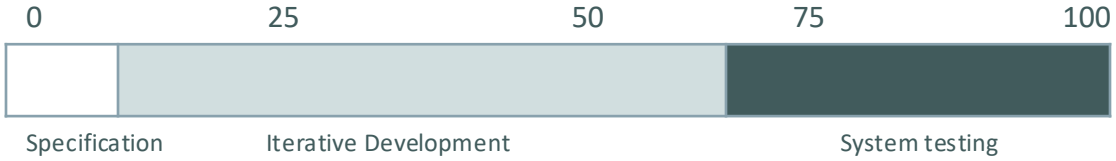
Cost distribution

Custom software development (Bespoke)

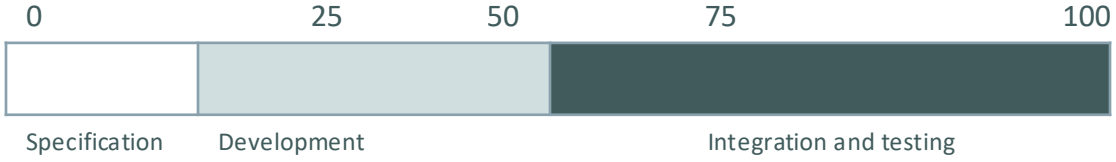
Software Model →
Cost units →
Cost distribution →
Software development activity →

Waterfall Model

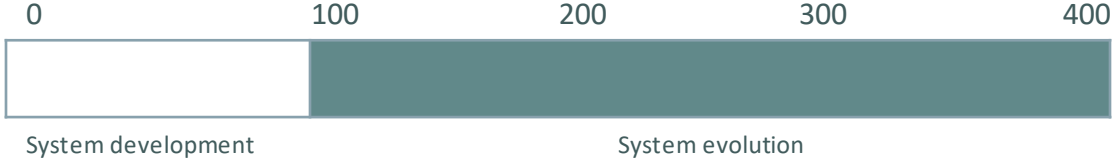
Iterative Development



Component-based Software Engineering

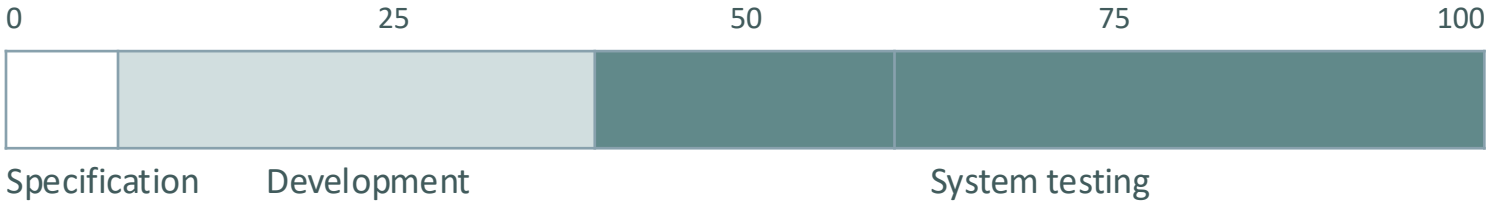


Development and evolution costs for long-lifetime systems



Cost distribution

Generic software development

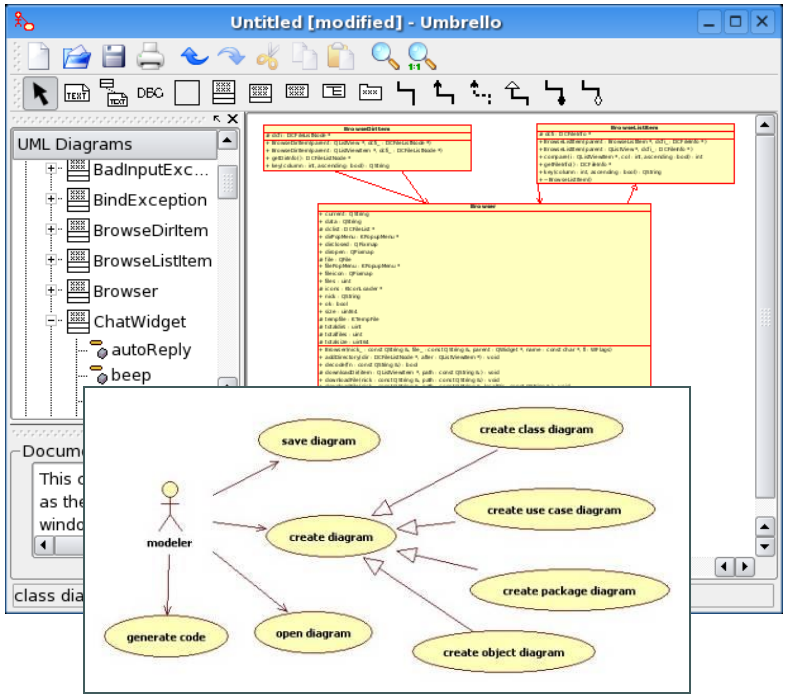


Product development costs

What is CASE?

- **C**omputer **A**ided **S**oftware **E**ngineering.

- Programs that support:
  - Requirements analysis.
  - System modeling.
  - Debugging.
  - Testing.



Attributes of good software

- Functional attributes (performance; what the system does).
- Non-functional attributes (quality; how the system does it).

Product Characteristic	Description
Maintainability	Evolution qualities such as Testability, extensibility.
Dependability	Reliability, security, safety.
Efficiency	Response time, processing time, memory utilization.
Usability	Easy to *learn* how to use the system by target users. *Efficient to use* the system by users to accomplish a task. *Satisfying* to use by intended users.

Activity

- What are the key attributes for..

Interactive game	**Banking system**	**Cardiac monitor in an ICU unit**
Players, score, scenes, theme.	Client accounts, stocks bonds, money transfers.	heart rate, temperature, blood pressure.
		

Challenges facing software engineering

Challenge	Why?	Software needs to ..
Heterogeneity	Different computers, different platforms, different support systems.	Cope with this variability.
Delivery	Businesses are more responsive → supporting software needs to evolve as rapidly.	Be delivered in shorter time without compromising quality.
Trust	Software is a part of many aspects of our lives (work, study, leisure).	Demonstrate that it can be trusted by users.

References

- PRESS&SUN-BULLETIN, The Binghamton Press Co., Binghamton, NY, October 1,1999.
- “Software Hell: Is there a way out?”, BUSINESS WEEK, December 6, 1999.
- *IEEE Standards Collection: Software Engineering, IEEE standard 610.12-1990, IEEE 1993.*
- Sommerville, Ian “*Software Engineering*”, 9th edition, Addison-Wesley.