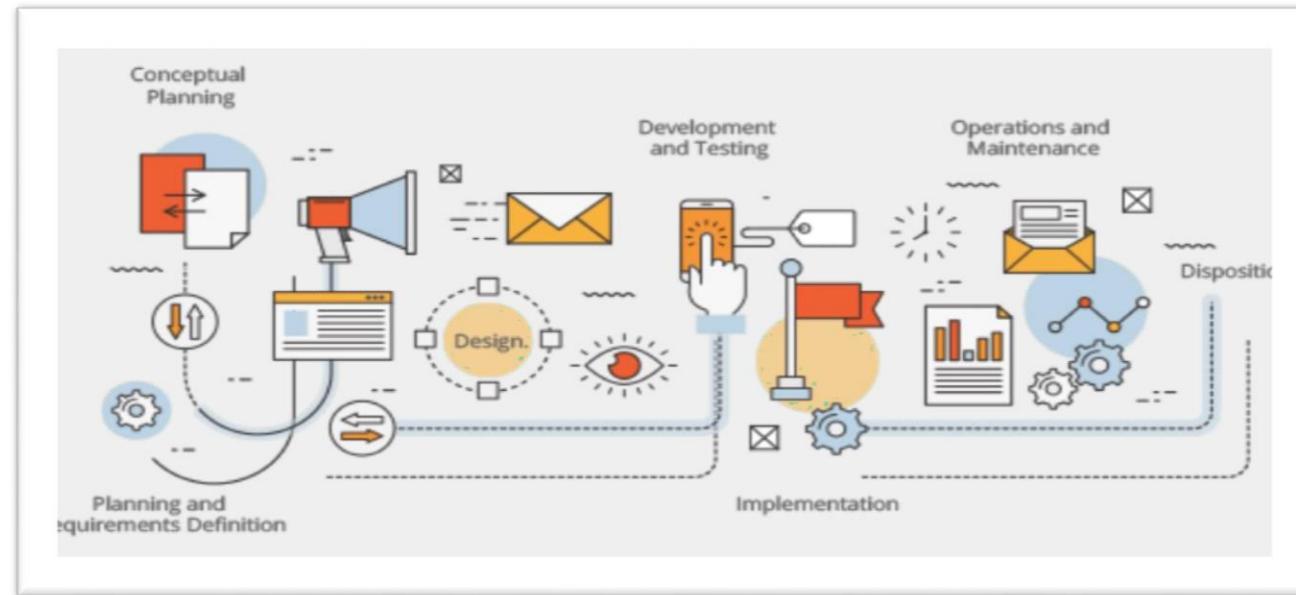


Software Engineering

Software Process Models Lecture#04



Revision Lecture 03

1. What is a software development process?
2. What are the generic activities in a software development process?
3. What are the three main classes of software process?
4. Discuss each phase of SDLC?
5. Difference between iterative and incremental development?



Categories of software process models

Software development process models can be broadly classified into three main categories based on their structure, flexibility, and approach to software engineering.

Model Type	Characteristics	Example Models	Suitable For
Prescriptive	Structured, rigid, sequential	Waterfall, V-Model	Well-defined, stable projects
Evolutionary	Iterative, flexible, user-driven	Agile, Spiral, Prototype	Dynamic projects with changing requirements
Specialized	Domain-specific, hybrid approaches	DevOps, Formal Methods	High-security, automated, or component-driven projects



1. Prescriptive Process Models (Traditional Models)

These models follow a **structured, well-defined, and sequential** approach to software development. They focus on **planning and documentation** before implementation.

Examples:

- **Waterfall Model** → Linear and sequential.
- **V-Model (Verification & Validation)** → Testing is mapped to each phase.
- **Incremental Model** → Builds software in small increments.

Best For: Well-defined requirements, mission-critical applications.

Drawback: Rigid; difficult to accommodate changes.



2. Evolutionary Process Models (Iterative & Agile)

These models focus on **incremental and iterative development**, allowing changes throughout the lifecycle. Feedback from stakeholders is **incorporated continuously**.

Examples:

- **Spiral Model** → Risk-driven, iterative development.
- **Prototype Model** → Quick prototypes for requirement validation.
- **Agile Models (Scrum, Kanban, XP)** → Fast, adaptive, and customer-centric.

✓ **Best For:** Projects with evolving requirements.

✗ **Drawback:** Can be difficult to manage for large-scale systems.



3. Specialized Process Models (Hybrid & Industry-Specific)

These models are designed for **specific domains or unique project needs**, often combining features of traditional and evolutionary models.

Examples:

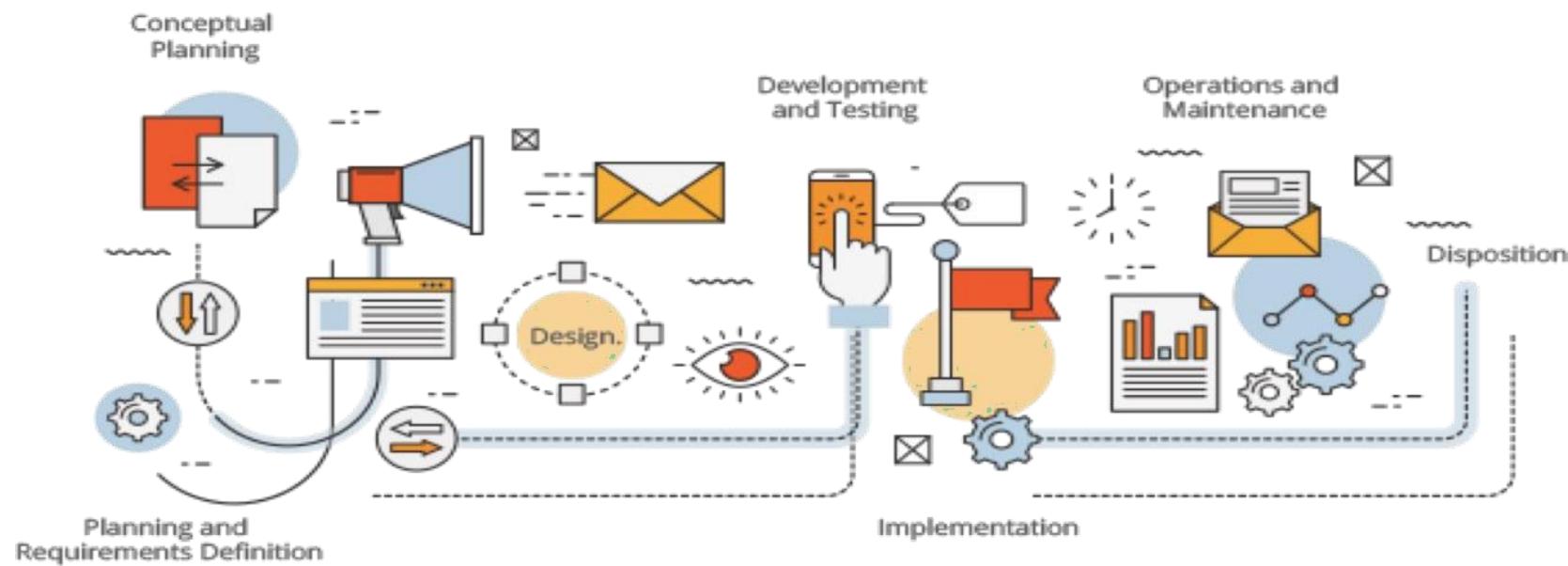
- **Component-Based Development (CBD)** → Uses pre-built components.
- **DevOps Model** → Integrates development and operations for continuous deployment.
- **Formal Methods Model** → Mathematically based, used in safety-critical systems (e.g., aviation, healthcare).

✓ **Best For:** Complex systems requiring high security, automation, or rapid delivery.

✗ **Drawback:** Requires expertise and may have high implementation costs.



Waterfall Model

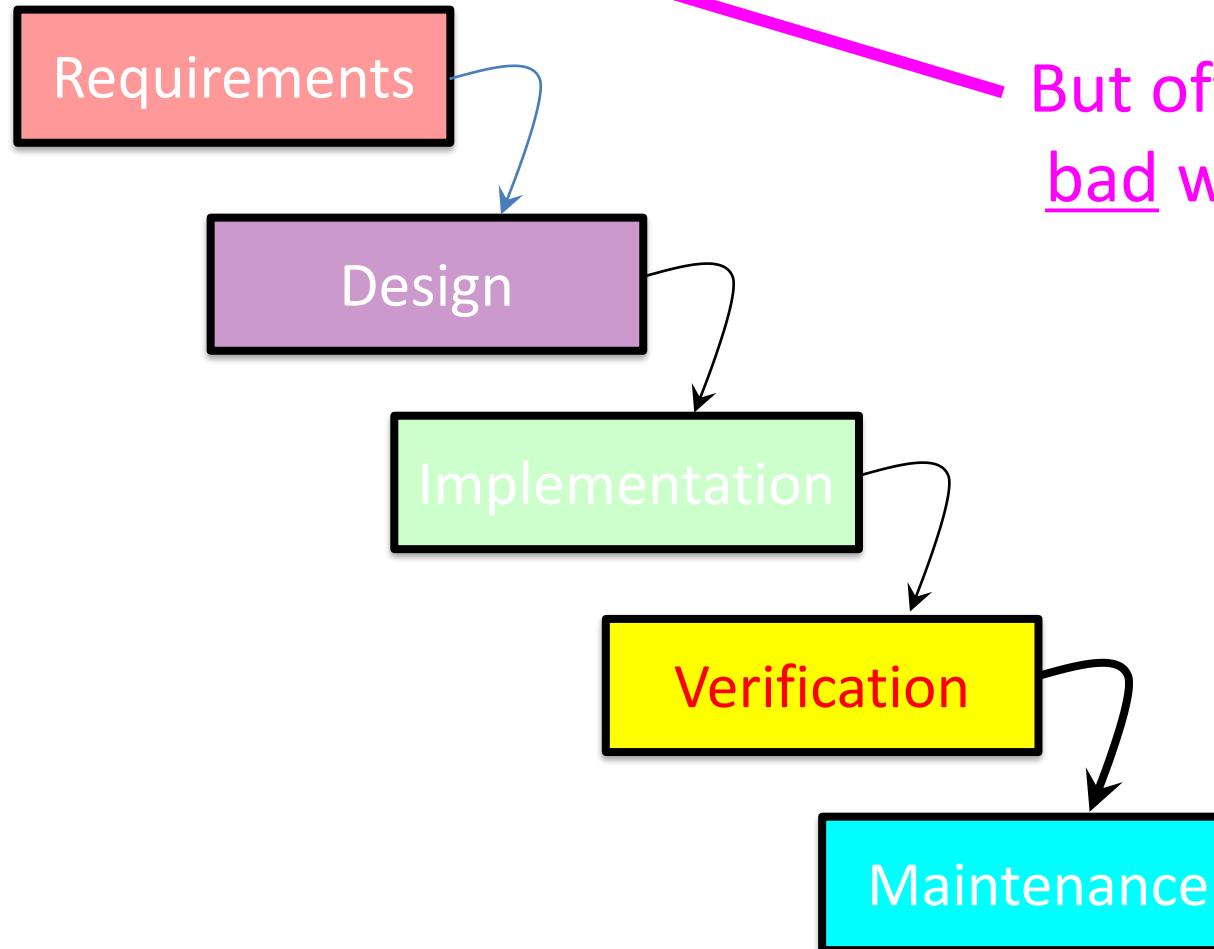


Prescriptive Process Models

Waterfall Model

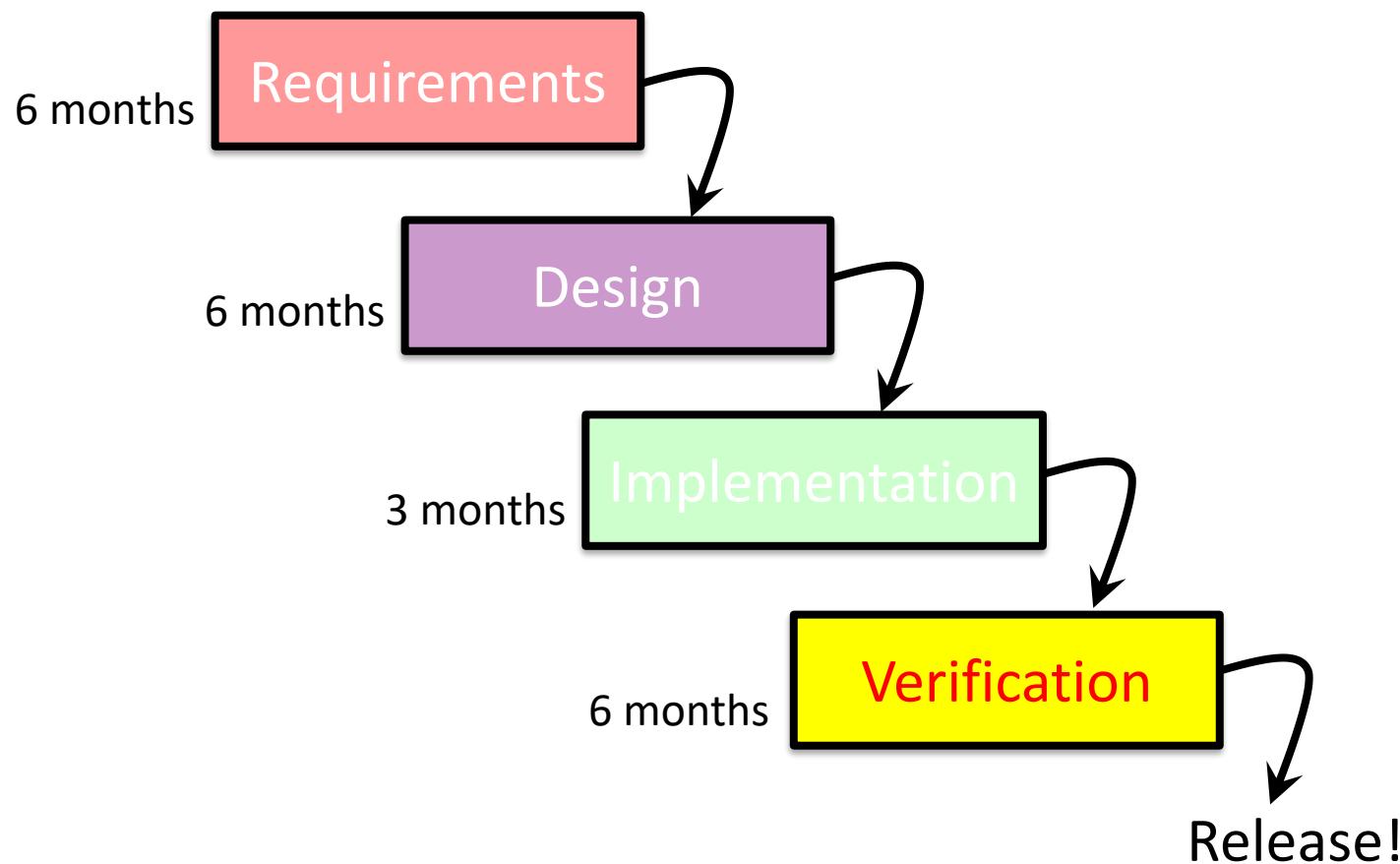
- The Waterfall Model was the first Process Model to be introduced.
- It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use.
- In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.
- The outcome of one phase acts as the input for the next phase sequentially.
- The Waterfall model is the earliest SDLC approach that was used for software development.

The classic way: Waterfall (sequential) model

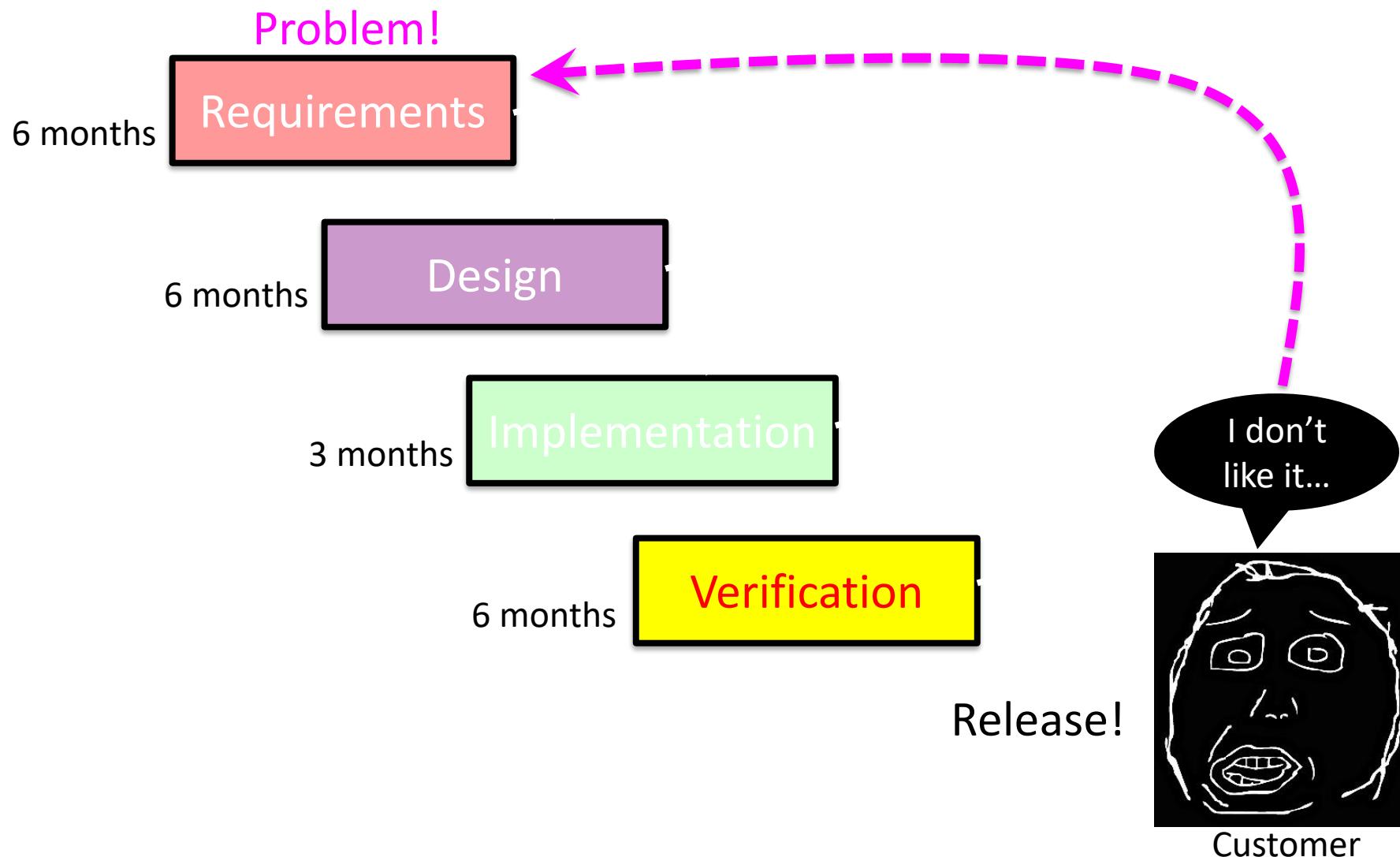


But often a
bad way...

What could go wrong?



What could go wrong?



Waterfall is often a *poor practice*

- High failure rates
- Low productivity
- High defect rates

45% of features
in requirements
never used

Early schedule
and estimates
off by up to 400%



Why Waterfall doesn't work

False assumption: Specifications ...
are predictable and stable
can be correctly defined at the start
have low change rates

Actual stats:
25% of requirements changed
35% to 50% changed in large projects

Features of Waterfall Model

- **Sequential Approach:** The waterfall model involves a sequential approach to software development, where each phase of the project is completed before moving on to the next one.
- **Document-Driven:** The waterfall model relies heavily on documentation to ensure that the project is well-defined and the project team is working towards a clear set of goals.
- **Quality Control:** The waterfall model places a high emphasis on quality control and testing at each phase of the project, to ensure that the final product meets the requirements and expectations of the stakeholders.
- **Rigorous Planning:** The waterfall model involves a rigorous planning process, where the project scope, timelines, and deliverables are carefully defined and monitored throughout the project lifecycle.

Sequential Development works best when

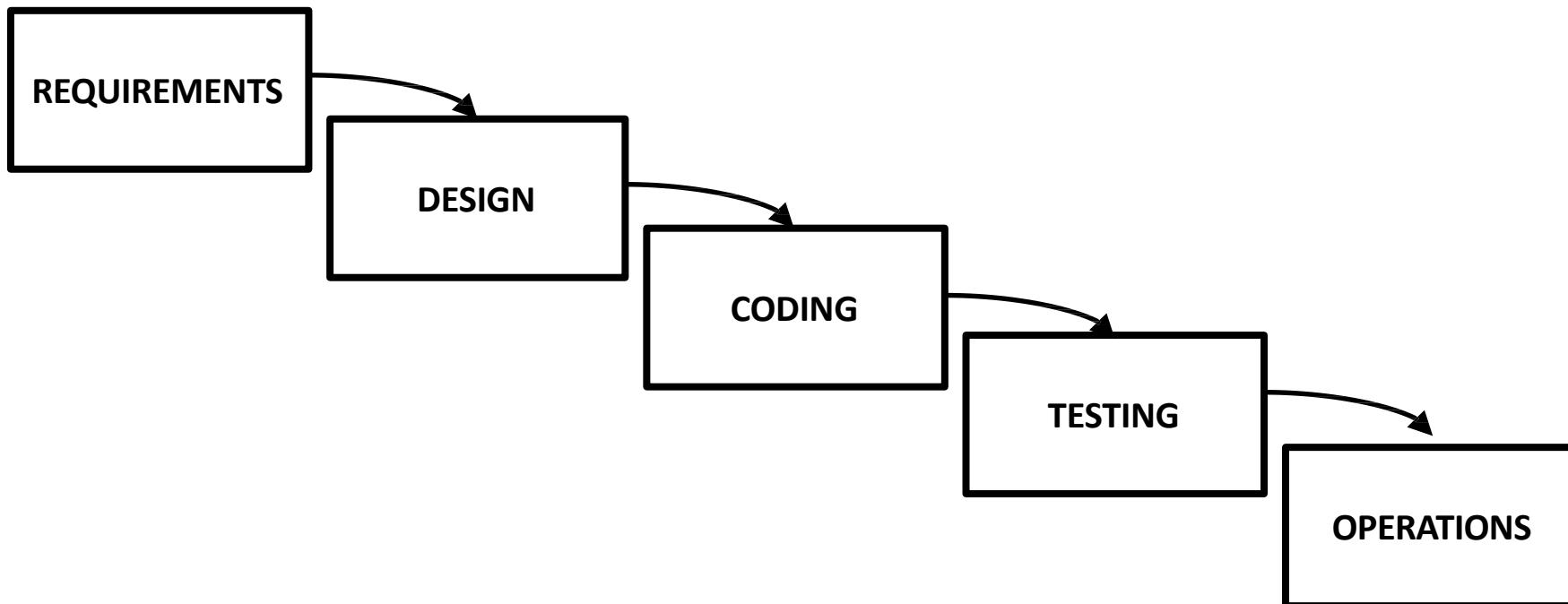
Sequential processes work best when the requirements are well understood and the design is straightforward, e.g.,

- Conversions of manual data processing systems where the requirements were well understood and few changes were made during the development (e.g., electricity billing).
- New models of a product where the functionality is closely derived from an earlier product (e.g. automatic braking system for a car).
- Portions of a large system where some components have clearly defined requirements and are clearly separated from the rest of the system.

Problems with Pure Waterfall

- No customer involvement after the Requirements phase
 - Requirements often change during software development
- Once the product is developed and if any failure occurs then the cost of fixing such issues are very high, because we need to update everything from document till the logic
- Testing does not occur until the end
 - No feedback during development
 - "We tend to go on for years, with tremendous investments to find that the system which was not well understood to begin with does not work as anticipated" – Robert Graham (MIT), 1968
 - Result: redesign and cost overruns

Waterfall Process Requires Backtracking



In practice, each stage in the process reveals new understanding of the previous stages, which often requires the earlier stages to be revised.

The Waterfall Model is not flexible enough.

Discussion of Waterfall Model

A pure sequential model is impossible

Examples:

- A feasibility study cannot create a proposed budget and schedule without a preliminary study of the requirements and a tentative design.
- Detailed design and implementation reveal gaps in the requirements specification.
- Requirements and/or technology may change during the development.

The plan must allow for some form of iteration.

Why Backtracking is Necessary

- **Uncovered Requirements:**

- During the implementation or testing phase, missing or unclear requirements may be discovered, requiring a return to the Requirements Analysis phase.

- **Design Flaws:**

- If the system design is found to be inadequate or flawed during implementation or testing, teams may need to revisit the System Design phase.

- **Bugs and Defects:**

- Testing may reveal defects that require changes to the code, design, or even requirements, leading to backtracking.

- **Changing Stakeholder Needs:**

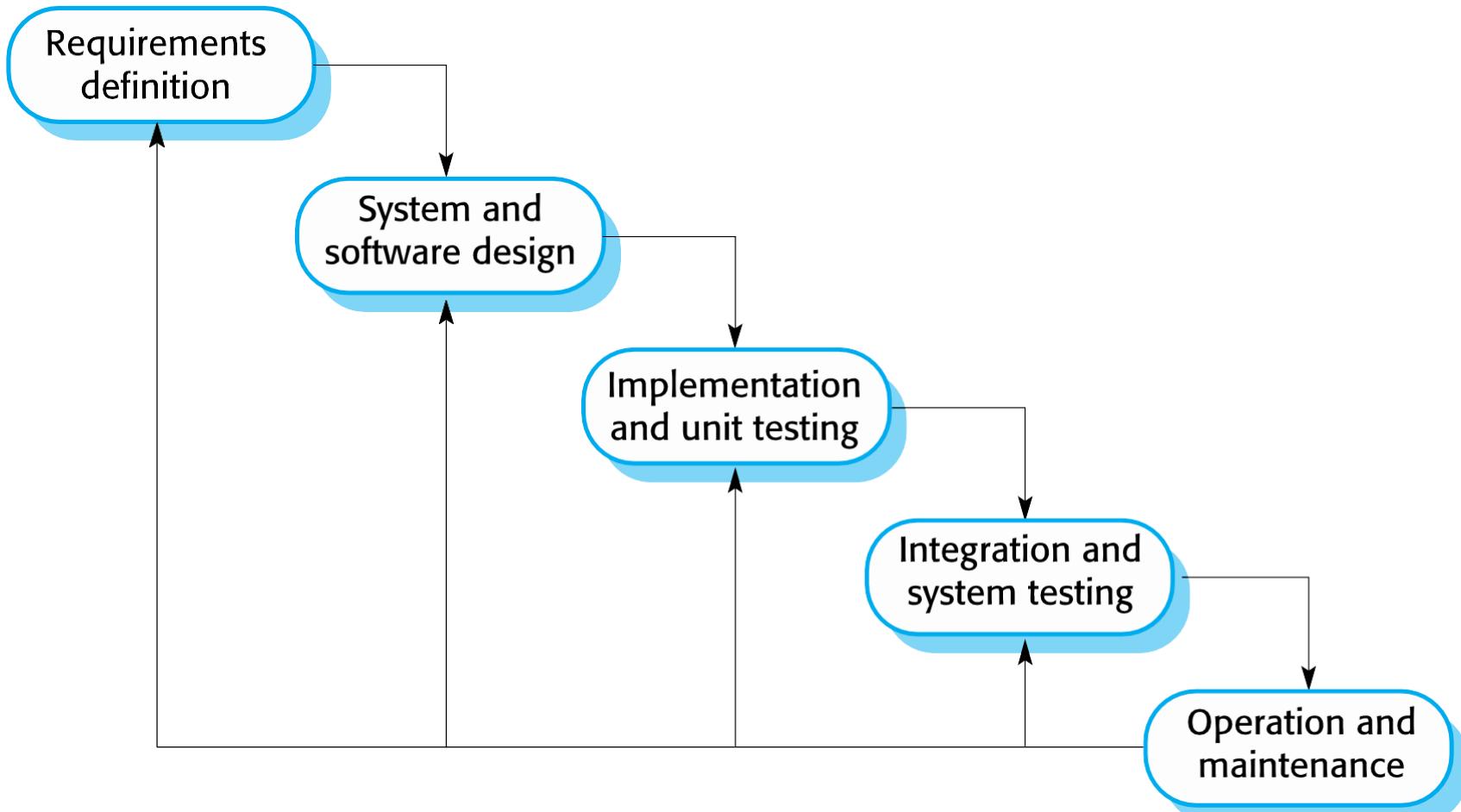
- Although Waterfall assumes stable requirements, stakeholders may request changes due to evolving business needs or market conditions.

- **Regulatory or Compliance Issues:**

- New regulations or compliance requirements may necessitate changes to the software, forcing teams to revisit earlier phases.



Incremental waterfall model



Incremental waterfall model

Instead of completing the entire project in one go (as in the traditional Waterfall model), the Incremental Waterfall Model **divides the system into smaller modules or increments**. Each increment is developed and delivered sequentially, while still following the Waterfall approach for each increment.



Steps in the Incremental Waterfall Model:

- 1. Requirements Gathering:** Identify all system requirements upfront and divide them into increments.

- 2. Incremental Development:**
 - Develop the first increment by following the traditional Waterfall steps.
 - Test and deliver the increment.

- 3. Iterative Refinement:**
 - Develop and deliver the next increment based on the predefined requirements and any feedback received.

- 4. Final Integration:** Combine all increments into the final system.



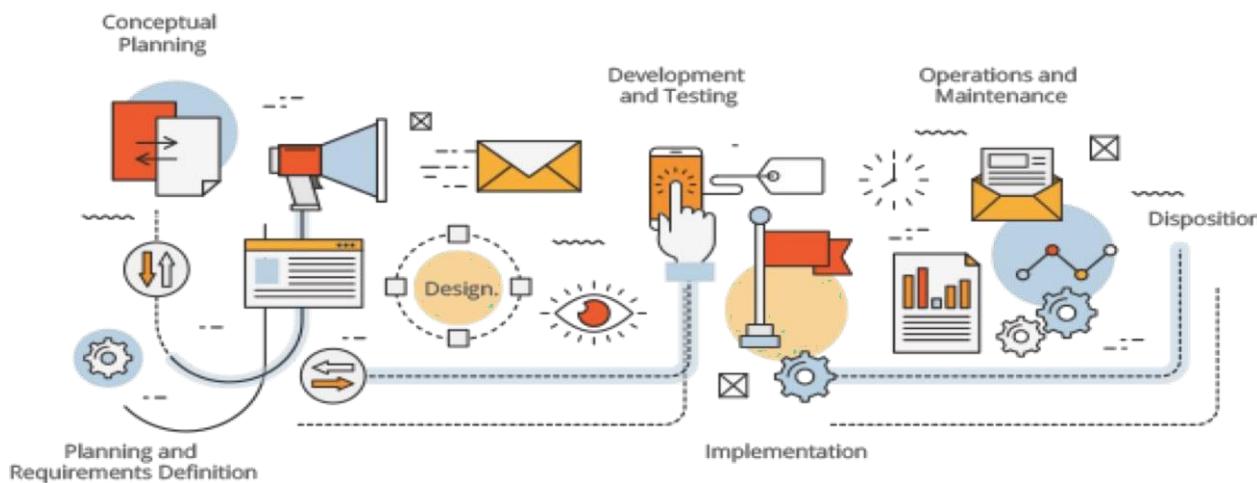
Summary

- Limited flexibility once an increment starts.
- Any change after an increment starts is difficult to accommodate.
- Each increment has a fixed scope, and functionality is delivered gradually until the entire system is completed
- Feedback for one increment may not influence increments already in development.

Use Incremental Waterfall if your project is large, well-defined, and needs to follow structured documentation (e.g., government systems).



V – Model



V-model

- The V-Model (Validation and Verification Model) is a software development lifecycle (SDLC) model that **extends** the Waterfall model by **emphasizing verification and validation** at each development stage.
- The V-model is an SDLC model where execution of processes happens in a **sequential manner** in a V-shape.

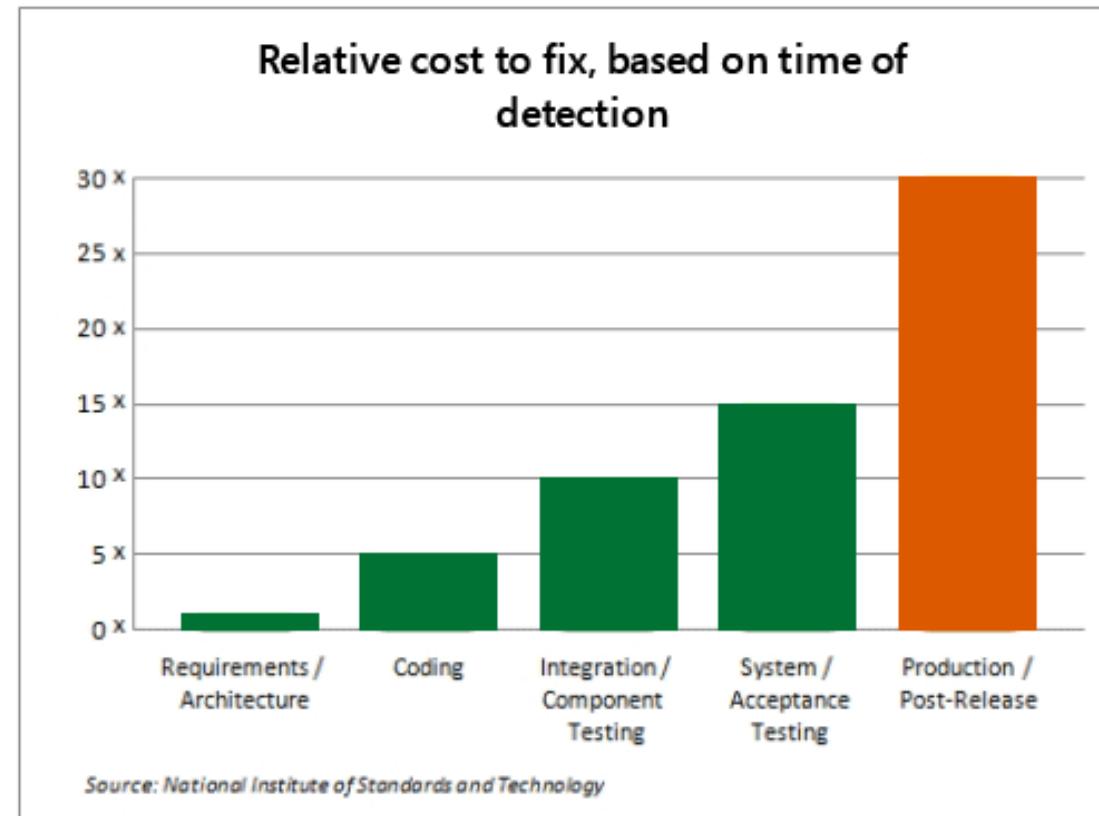
Problem with the Waterfall Model

As you may observe, that **testing in the model starts only after implementation is done.**

But if you are working in the large project, where the systems are complex, it's easy to miss out the key details in the requirements phase itself. In such cases, an entirely wrong product will be delivered to the client and you might have to start a fresh with the project OR if you manage to note the requirements correctly but make serious mistakes in design and architecture of your software you will have to redesign the entire software to correct the error.



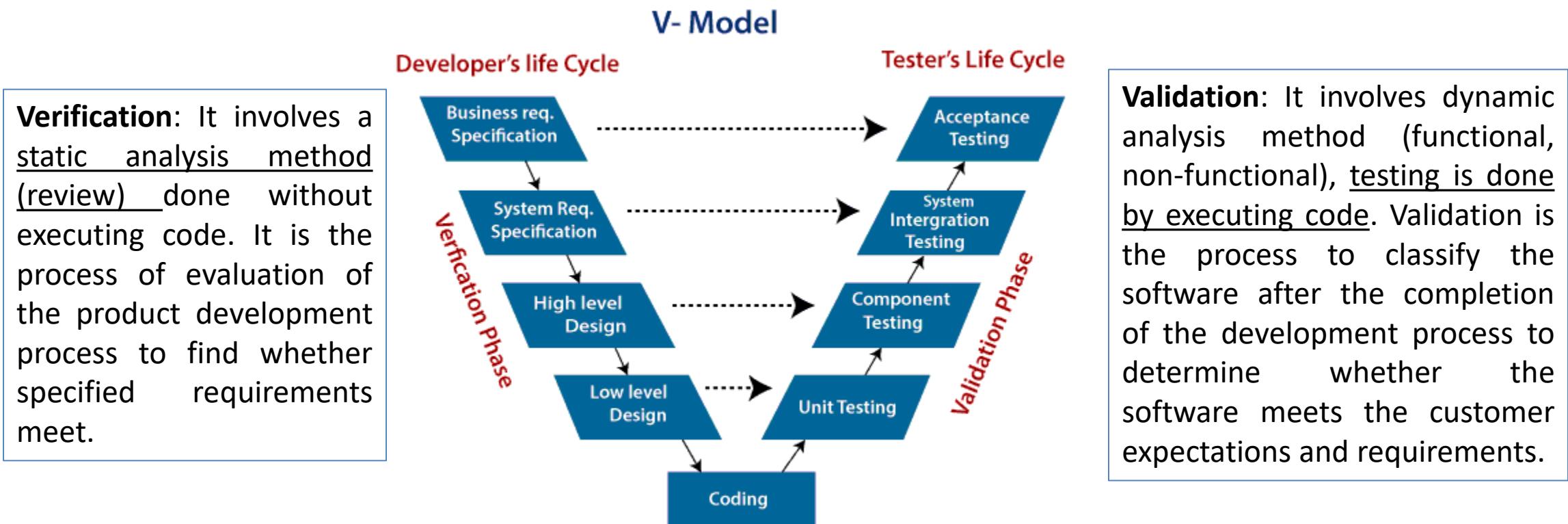
Numerous studies on software development projects have shown that **defects introduced during the requirements and design phases account for nearly 50% of total defects found in a project**. This finding underscores the importance of early validation and verification in software engineering.



Also, the costs of fixing a defect increase across the development lifecycle. The earlier in life cycle a defect is detected, the cheaper it is to fix it.

Solution: The V Model

To address this concern, **the V model of testing** was developed where **for every phase, in the Development life cycle there is a corresponding Testing phase**



Testing Life cycle

Phase	Description	Corresponding Testing Phase
Requirement Analysis	Gather & define system requirements with stakeholders.	Acceptance Testing
System Design	Design system architecture based on requirements.	System Testing
High-Level Design (HLD)	Break system into modules, define data flow & interactions.	Integration Testing
Low-Level Design (LLD)	Define internal logic, algorithms & component details.	Unit Testing
Implementation (Coding)	Developers write and compile code based on design.	Code Review & Static Testing



There are the various phases of Validation Phase of V-model:

Unit Testing: Once the system design is completed, individual components or units of the software are tested in isolation to ensure they function correctly.

Integration Testing: After unit testing, the units are combined and tested as a group to ensure they work together as intended.

System Testing: The entire system is tested as a whole to verify that it meets the specified requirements.

Acceptance Testing: This is the final phase where the software is tested by the end-users to determine if it meets their acceptance criteria and is ready for deployment.



Example: How the V-Model Works in a Real Project

Scenario: Developing a Banking Application

Step 1: Requirements Gathering

The business analyst defines requirements (e.g., "Users should be able to transfer money securely").

Corresponding Test: Acceptance test cases are written (e.g., "Verify user can transfer funds between accounts").

Step 2: System Design

Architects design a secure transaction system.

Corresponding Test: System tests ensure data security and transaction speed.

Step 3: High-Level Design

Developers define modules (e.g., Login, Account, Transfer).

Corresponding Test: Integration tests check if login, account management, and transfer modules work together.

Advantages of the V-Model

- ✓ **Early Bug Detection** → Reduces late-stage failures.
- ✓ **Clear Structure** → Well-documented phases prevent missing steps.
- ✓ **Reliable for Critical Systems** → Used in **medical, automotive, aerospace** industries.
- ✓ **Strict Verification & Validation** → Ensures high-quality software.



Disadvantages

- **Rigid:** The V-model can be inflexible and provide very little room for changes or deviations from the plan. This rigidity can make it difficult to adapt to changing project requirements or new information.
- **Time-Consuming:** The V-model can be time-consuming due to its focus on thorough planning and documentation at every stage. These factors can slow down the development process and lead to longer project timelines.
- **Resource Intensive:** The V-model requires a significant amount of resources including time, budget and personnel, thereby making it a difficult model for small teams or organizations with limited resources to implement.
- **Limited Agility:** The V-model may not be well suited for Agile development approaches, which rely on flexibility, iterative development and continuous feedback.
- **Overemphasis on Testing:** While thorough testing is a critical component of software development, the V-Model may place too much emphasis on testing, which can lead to production delays and increased costs.

The V-Model is suitable for projects with well-defined requirements, stable technology, and a clear understanding of the desired end product. It is most appropriate for projects where:

1. Requirements are Stable: The V-Model works best when the project requirements are well-understood and unlikely to change significantly throughout the development process. Projects with evolving or uncertain requirements may struggle with the V-Model's sequential nature.

2. Clear and Predictable Outputs: Projects that produce relatively predictable outputs, such as safety-critical systems, medical devices, or regulatory compliance software, can benefit from the V-Model's emphasis on comprehensive testing and documentation.

3. Risk Management is Critical: The V-Model is suitable for projects where risk management is a high priority. By integrating testing activities throughout the development lifecycle, the V-Model helps identify and mitigate risks early, reducing the likelihood of critical issues in the final product.

4. Experienced Development Teams: Projects with experienced development teams who are familiar with the V-Model and its processes are more likely to succeed with this approach. A solid understanding of the sequential nature of the model and the importance of thorough testing is crucial for successful implementation.

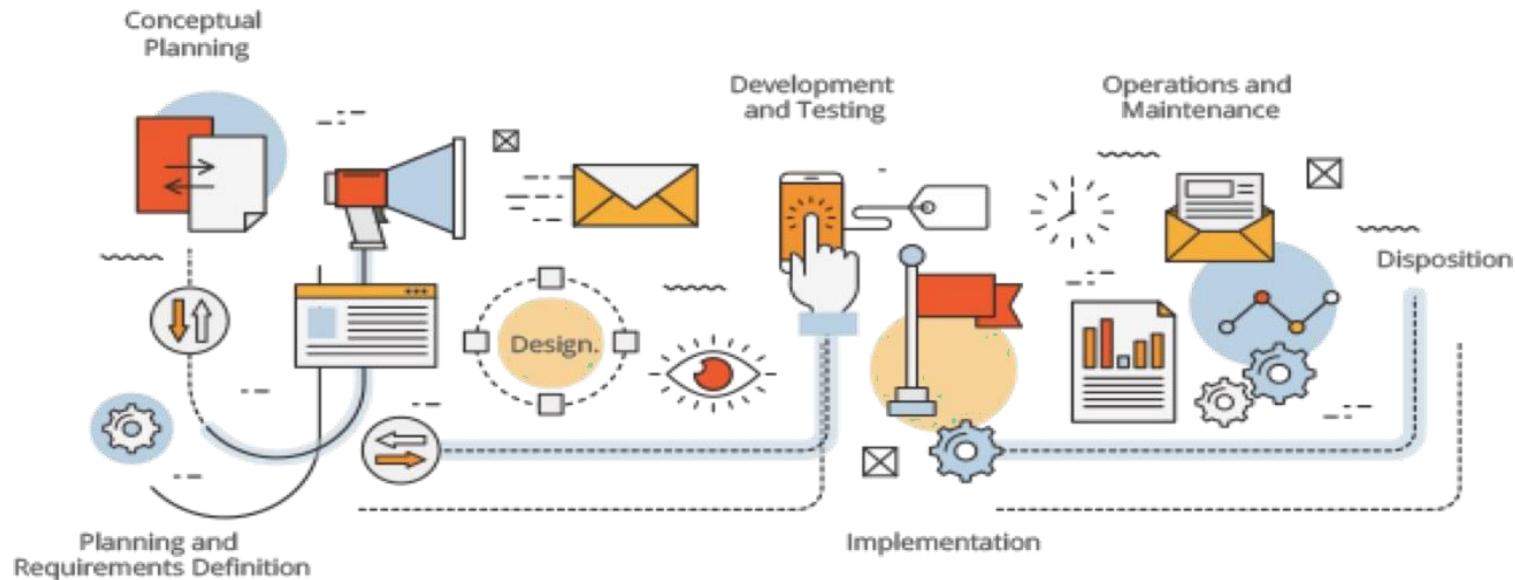
5. Regulated Industries: The V-Model is often preferred in industries with strict regulatory requirements, such as aerospace, defense, healthcare, and automotive sectors. These industries typically require thorough documentation and testing to ensure compliance with regulatory standards.

6. Longer Timeframes: Projects with longer timeframes and ample resources can accommodate the sequential nature of the V-Model more easily. Shorter, more iterative projects may find the V-Model too rigid and time-consuming.



Evolutionary Process Models

Prototyping Model



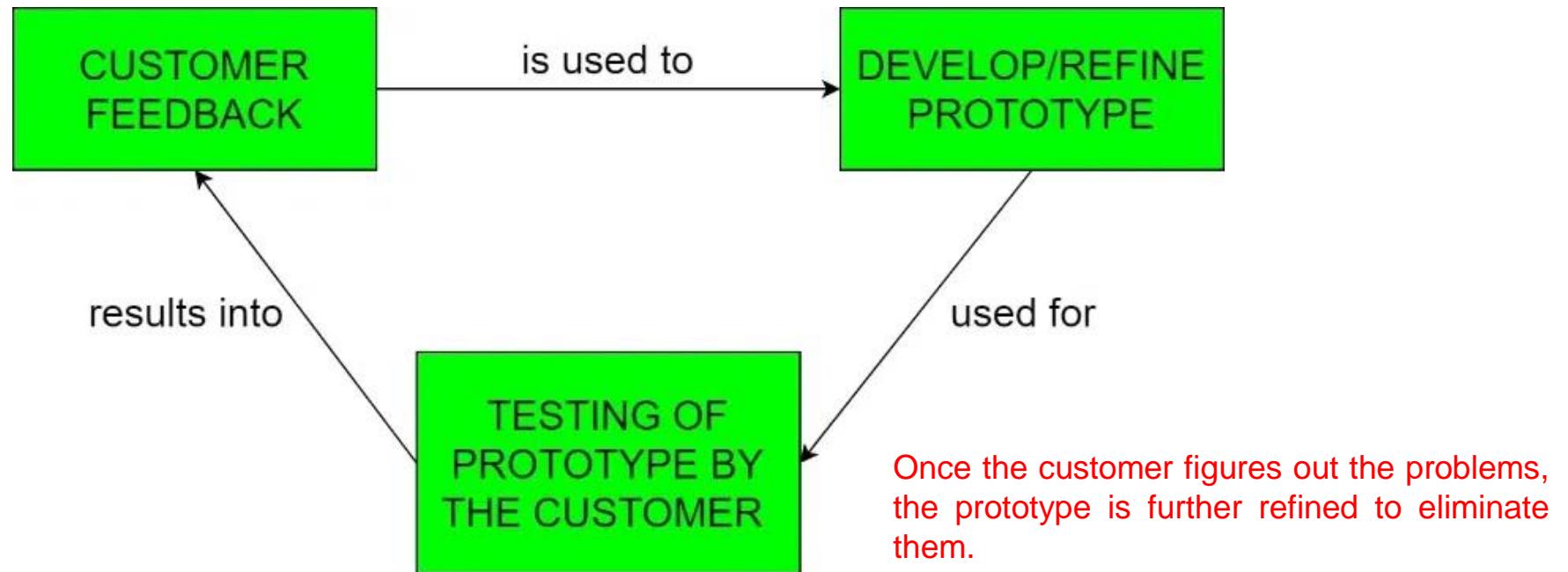
Software Prototyping Model

The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used **when the customers do not know the exact project requirements beforehand**. In this model, a prototype of the end product is first developed, tested, and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

Software prototyping

The process starts by interviewing the customers and developing the incomplete high-level paper model.

This document is used to build the initial prototype supporting only the basic functionality as desired by the customer

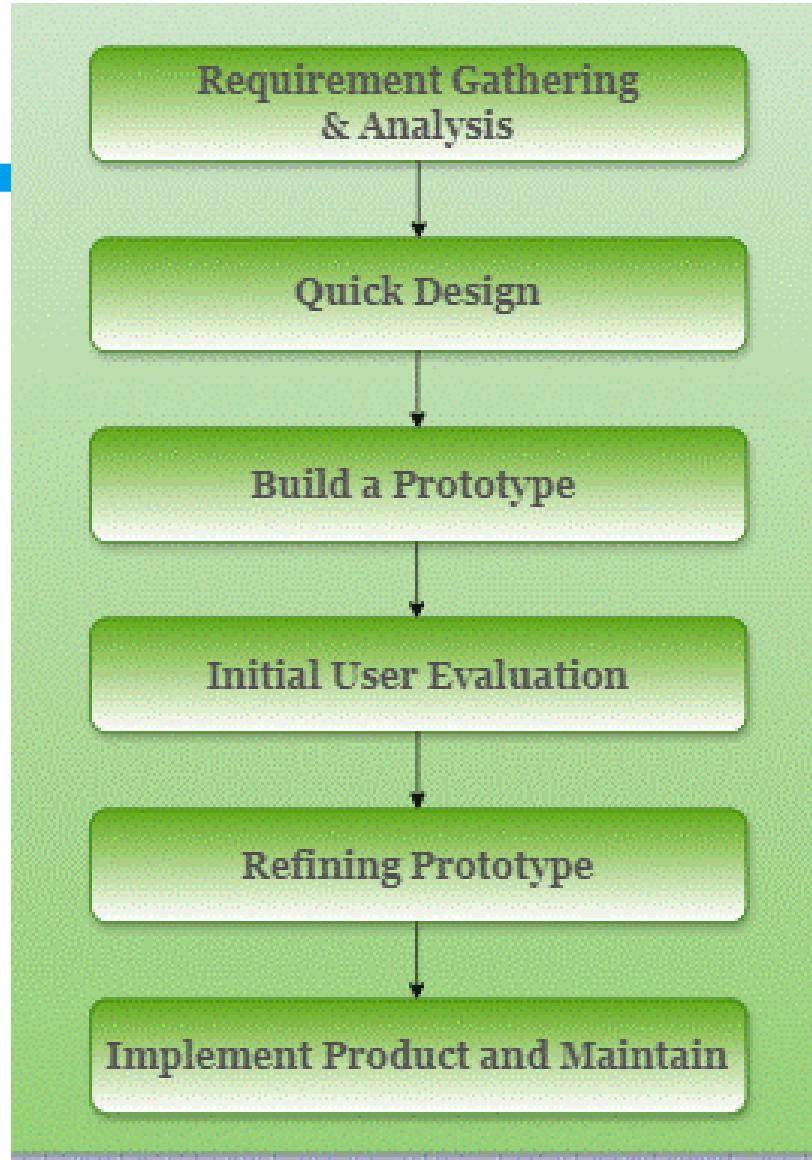


The process continues until the user approves the prototype and finds the working model to be satisfactory.

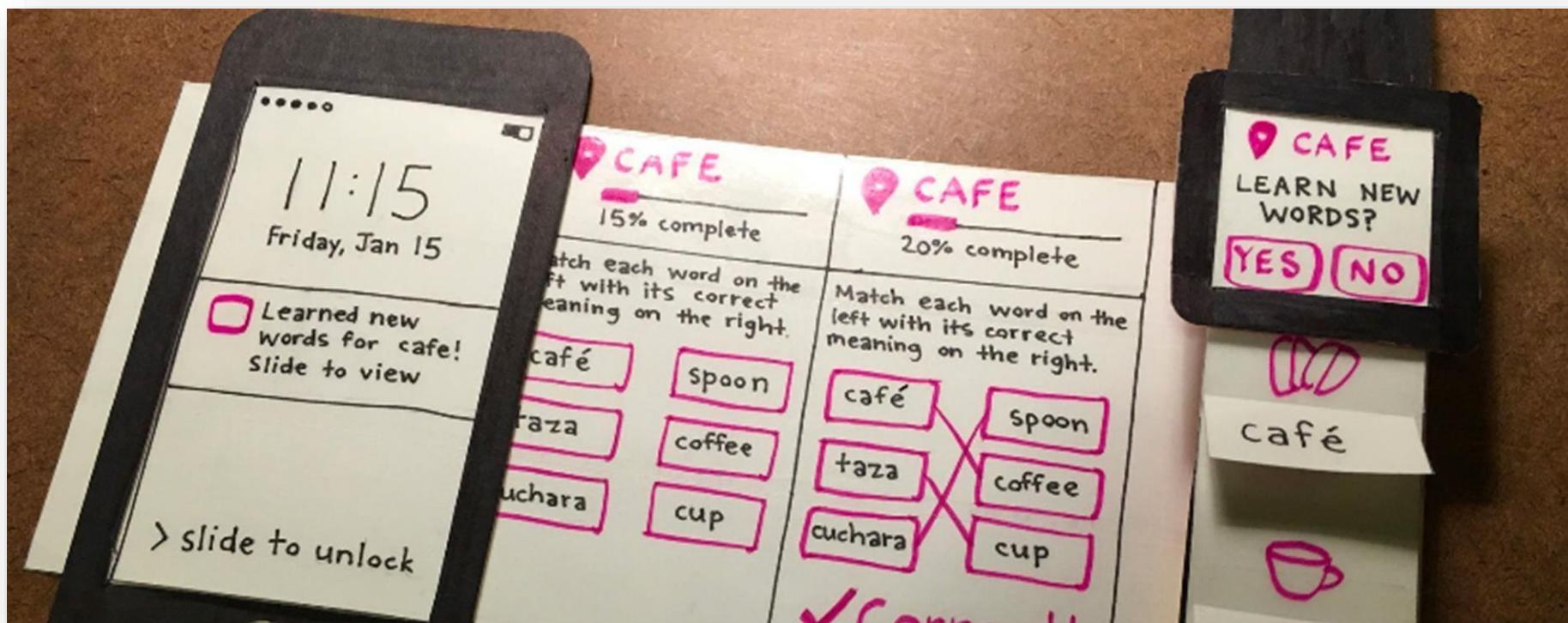
Key Features

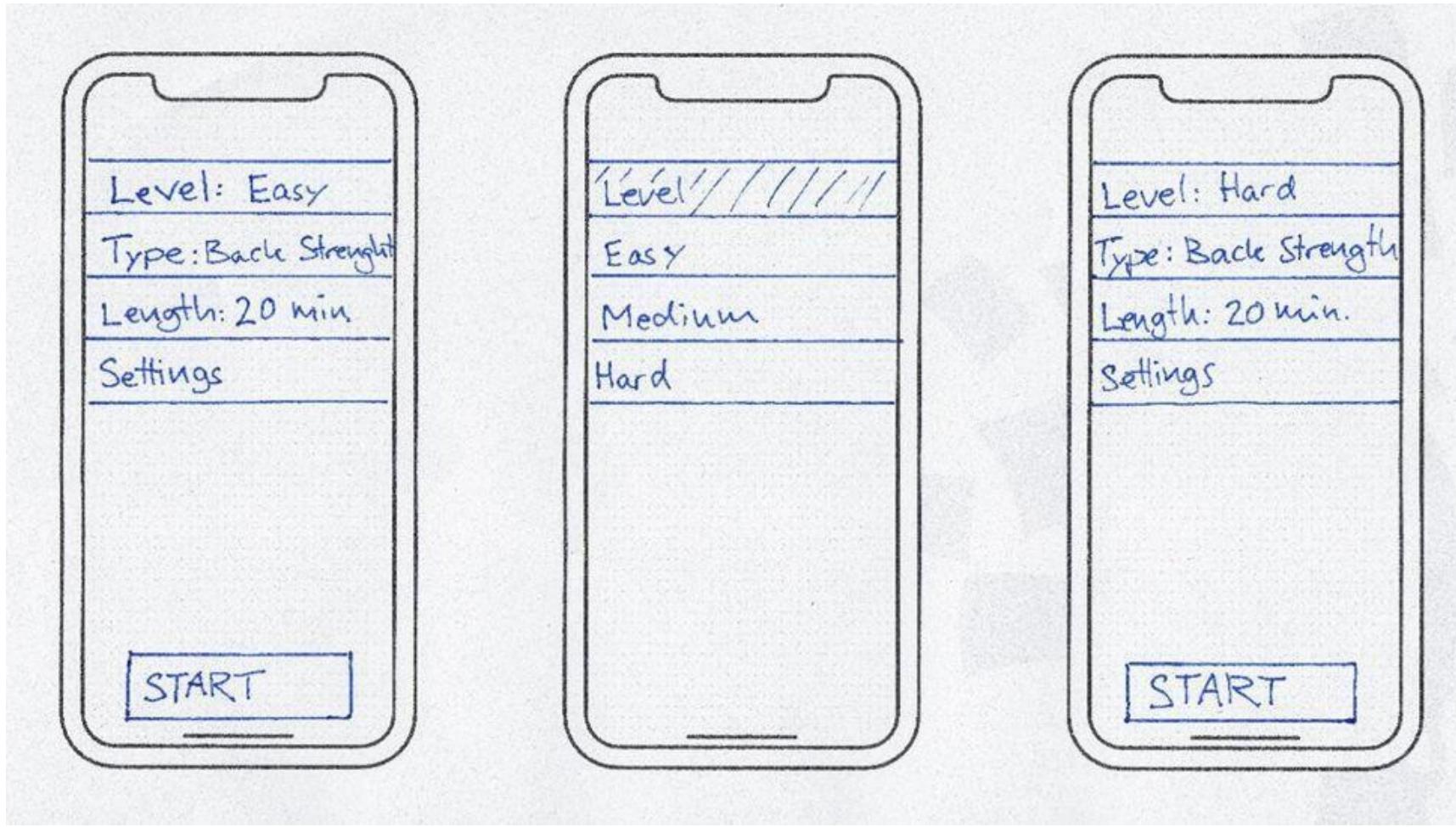
- **Early Visualization:** A prototype is created early to visualize the system's functionality or user interface.
- **User Feedback:** Users interact with the prototype and provide feedback, which helps refine the system.
- **Iterative Development:** The prototype is continuously refined until it meets user needs and can serve as a basis for the final system.
- **Focus on High-Risk Areas:** Prototyping focuses on developing and testing the most critical or uncertain parts of the system first.

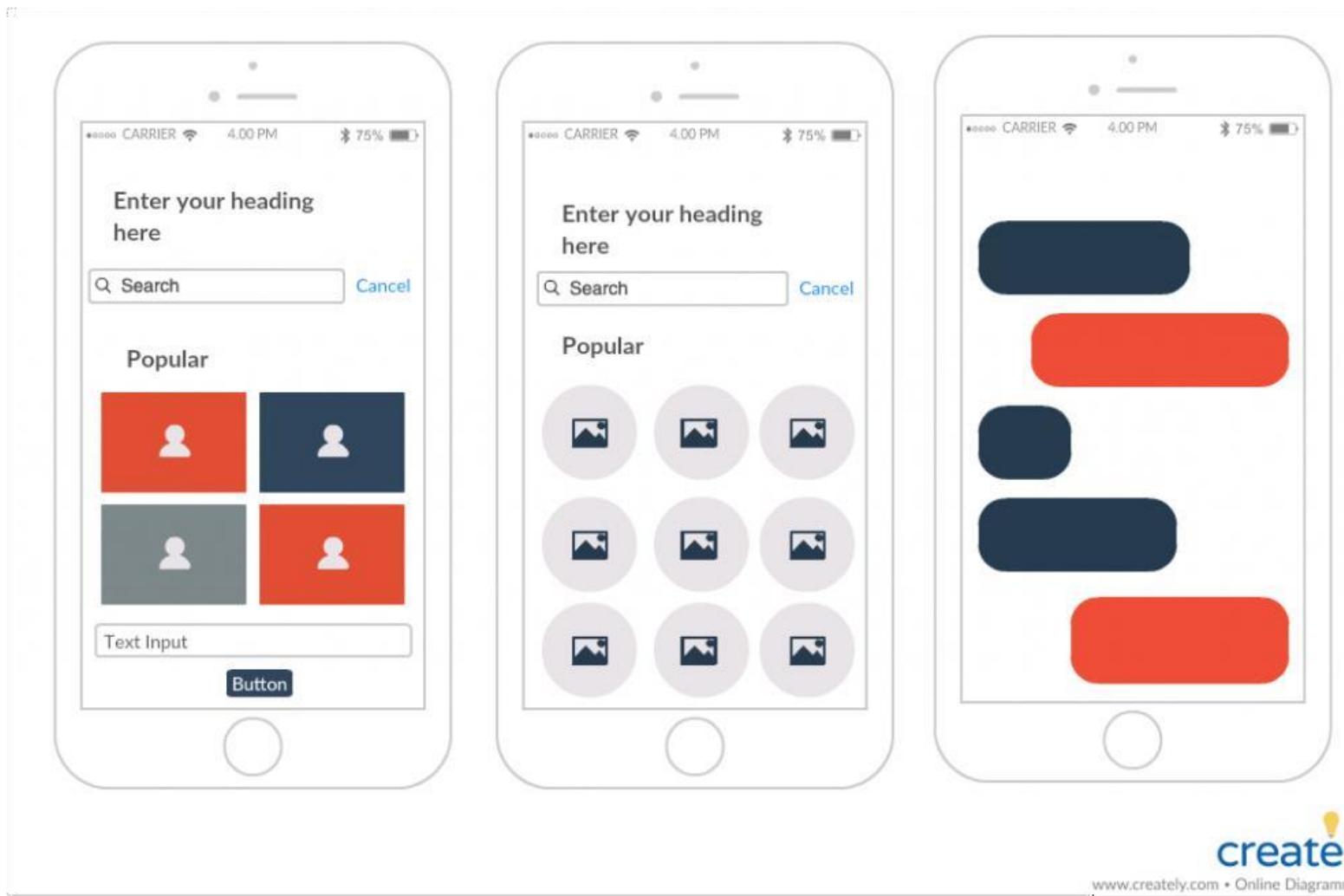
Prototype Model



Examples



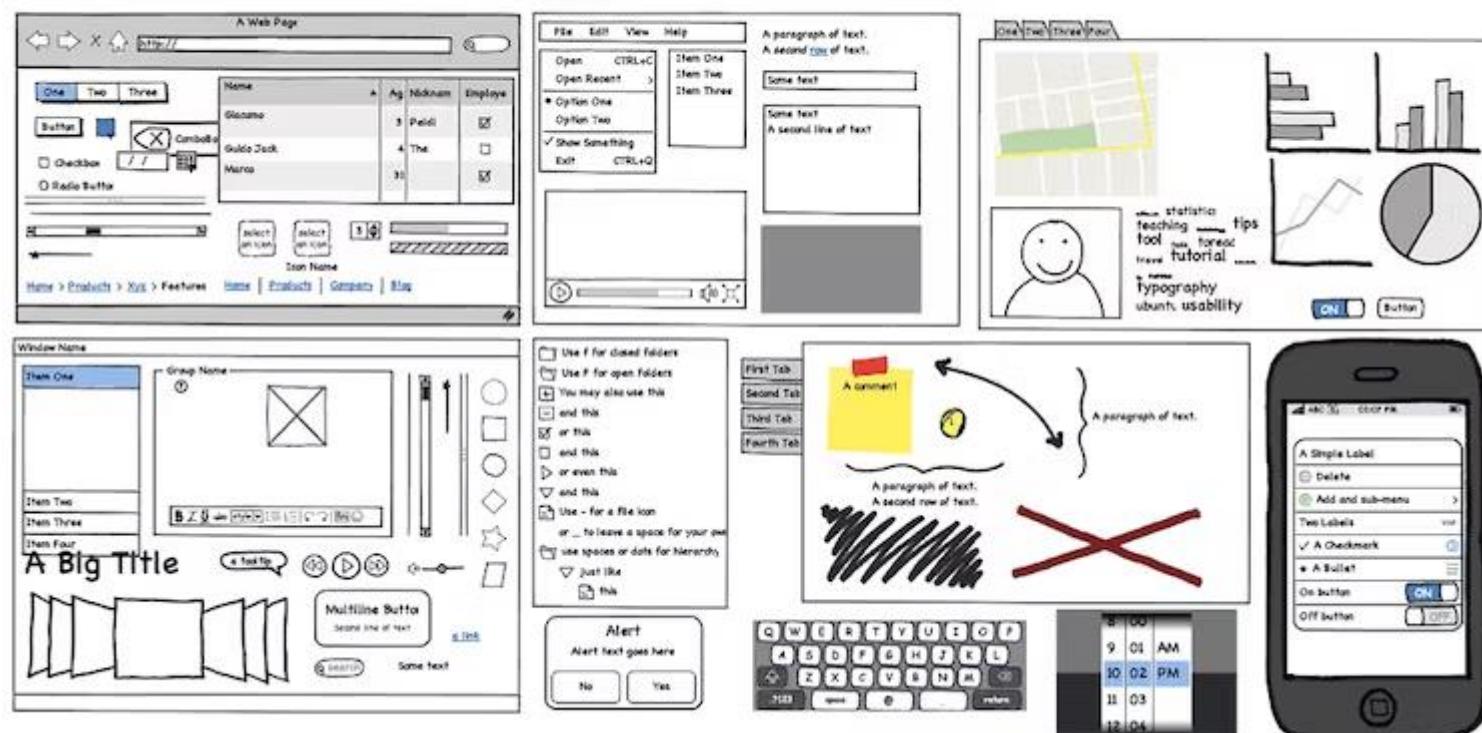




What is Balsamiq used for?

Balsamiq Cloud is a web-based user interface design tool for creating wireframes (sometimes called mockups or low-fidelity prototypes). You can use it to generate digital sketches of your idea or concept for an application or website, and to facilitate discussion and understanding before any code is written.

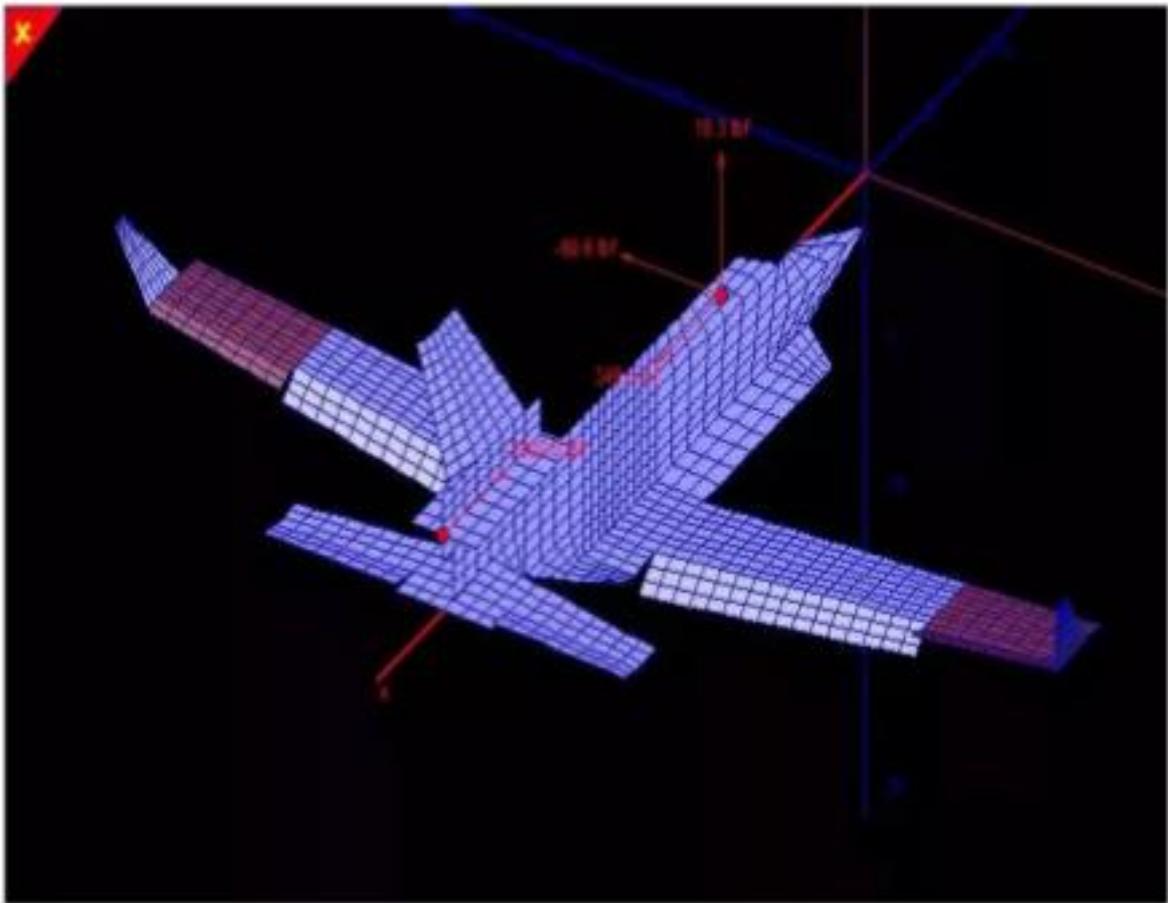
Components for Web, Application, and Mobile Interface Design



EXAMPLES -

- **Digital Prototype** -A digital prototype allows product developers to create a virtual model of the product which enables them to see how the individual components will work together and how the product will look once it's completed.

That is, it lets the developers virtually explore the complete product before it's actually built.



Throwaway prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
- It may be impossible to tune the system to meet non functional requirements;
- Prototypes are normally undocumented;
- The prototype structure is usually degraded through rapid change;
- The prototype probably will not meet normal organizational quality standards.

Advantages

- ▶ This model is flexible in design.
- ▶ It is easy to detect errors.
- ▶ We can find missing functionality easily.
- ▶ There is scope of refinement, it means new requirements can be easily accommodated.
- ▶ It can be reused by the developer for more complicated projects in the future.
- ▶ It ensures a greater level of customer satisfaction and comfort.
- ▶ It is ideal for online system.
- ▶ It helps developers and users both understand the system better.
- ▶ Integration requirements are very well understood and deployment channels are decided at a very early stage.
- ▶ It can actively involve users in the development phase.

Limitations

- **Scope Creep:**
 - Users may keep requesting changes, leading to uncontrolled growth in the scope of the project.
 - Customers sometimes demand the actual product to be delivered soon after seeing an early prototype.
- **High Costs and Time:**

Iterative refinement can increase costs and extend timelines.
- **Over-Reliance on the Prototype:**

Users might mistake the prototype for the final product, leading to unrealistic expectations.
- **Limited Scalability:**

Quick-built prototypes may not be scalable or well-structured for full system development.

When to use prototyping

- ▶ Prototype model should be used when the desired system needs to have a lot of **interaction with the end users**.
- ▶ Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- ▶ Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Tools

- proto.io
- moqups
- invisionapp
- mybalsamiq
- pixate
- marvelapp
- fluid UI
- flinto
- popapp.in

Task: Waterfall Model for Non-Software Projects

- Task: Apply the Waterfall model to a non-software project (e.g., planning a university event, host a qawali night).
- Details: Break the project into phases (Planning, Design, Execution, Testing, Delivery) and document each step.