EAST WEST UNIVERSITY

CSE 412

Software Engineering

Yasin Sazid
Lecturer
Department of CSE
East West University

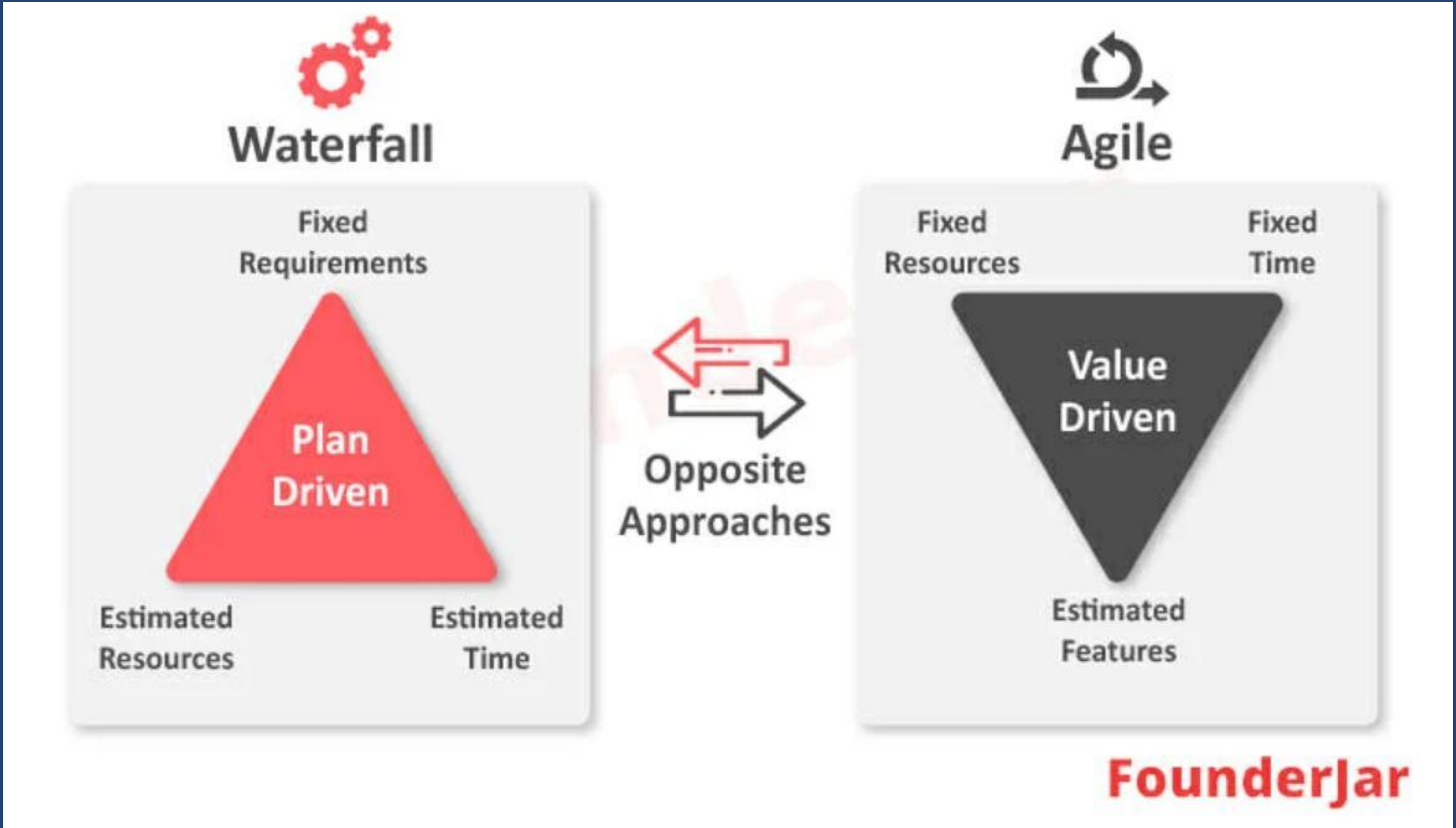
Topic 2: Software Processes & Process Models

Plan-Driven Development Process
vs
Agile Development Process

Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan

vs

Agile processes allow incremental planning and it is easier to change the process to reflect changing customer requirements.



Waterfall

Fixed Requirements

Plan Driven

Estimated Resources

Estimated Time

Opposite Approaches

Agile

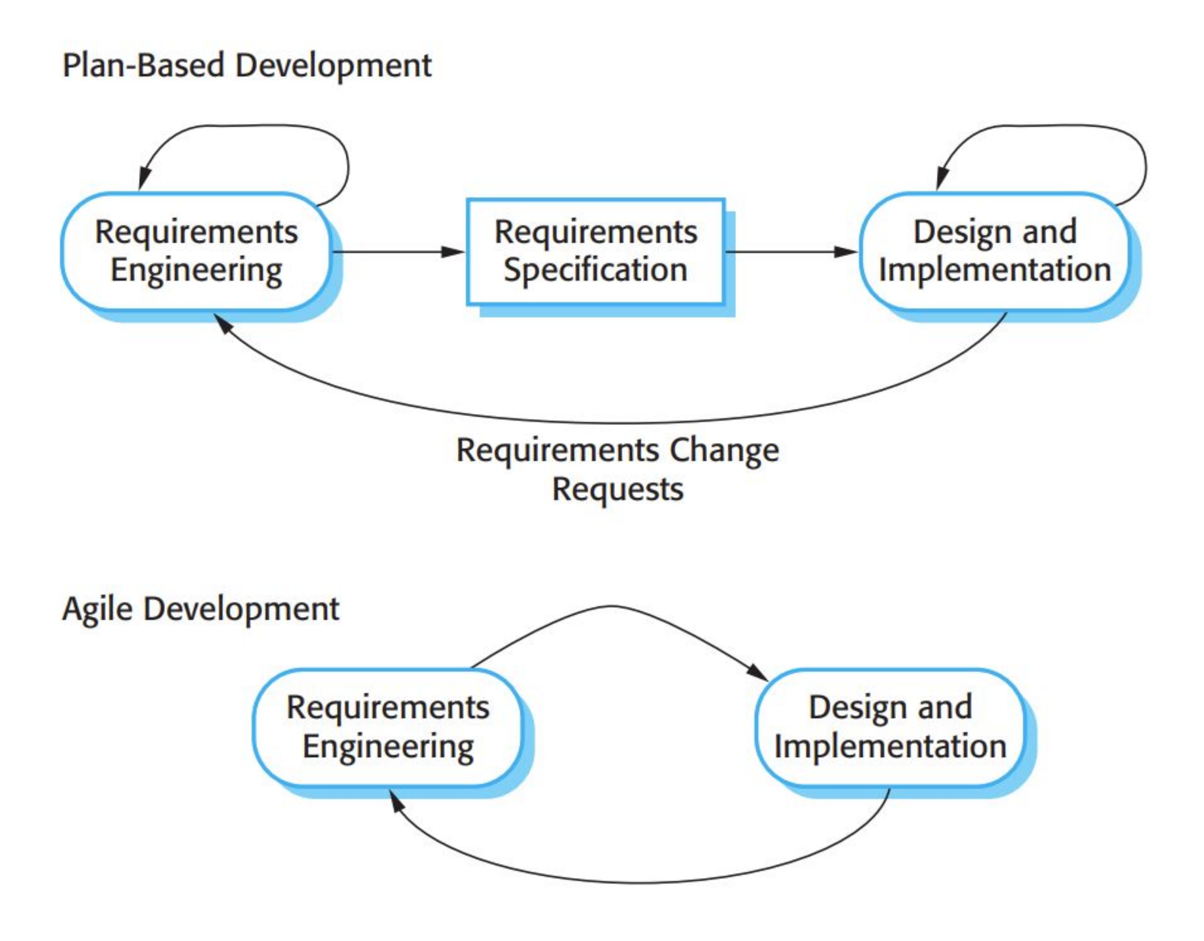
Fixed Resources

Fixed Time

Value Driven

Estimated Features

FounderJar



Plan-Based Development

Requirements Engineering

Requirements Specification

Design and Implementation

Requirements Change Requests

Agile Development

Requirements Engineering

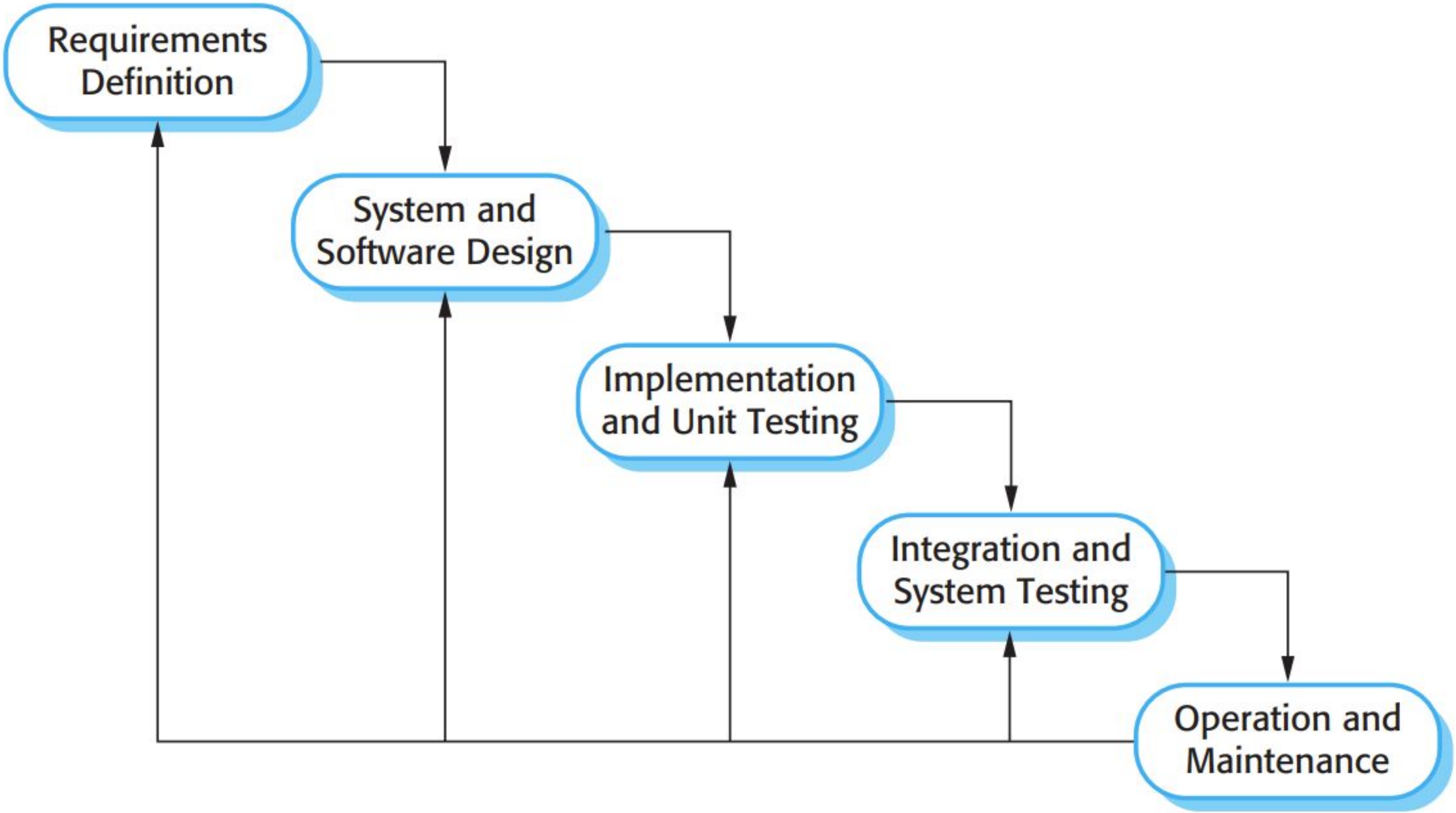
Design and Implementation

Some Common Process Models

Waterfall Model

Waterfall Model

- A linear and sequential software development approach.
- A phase cannot begin until the previous phase is completed.
- No overlapping of phases is allowed.
- A step-by-step flow, similar to water falling from a cliff—it cannot go back up.
- The output of one phase serves as the input for the next phase.
- Commonly used for small and well-defined projects with clear requirements.

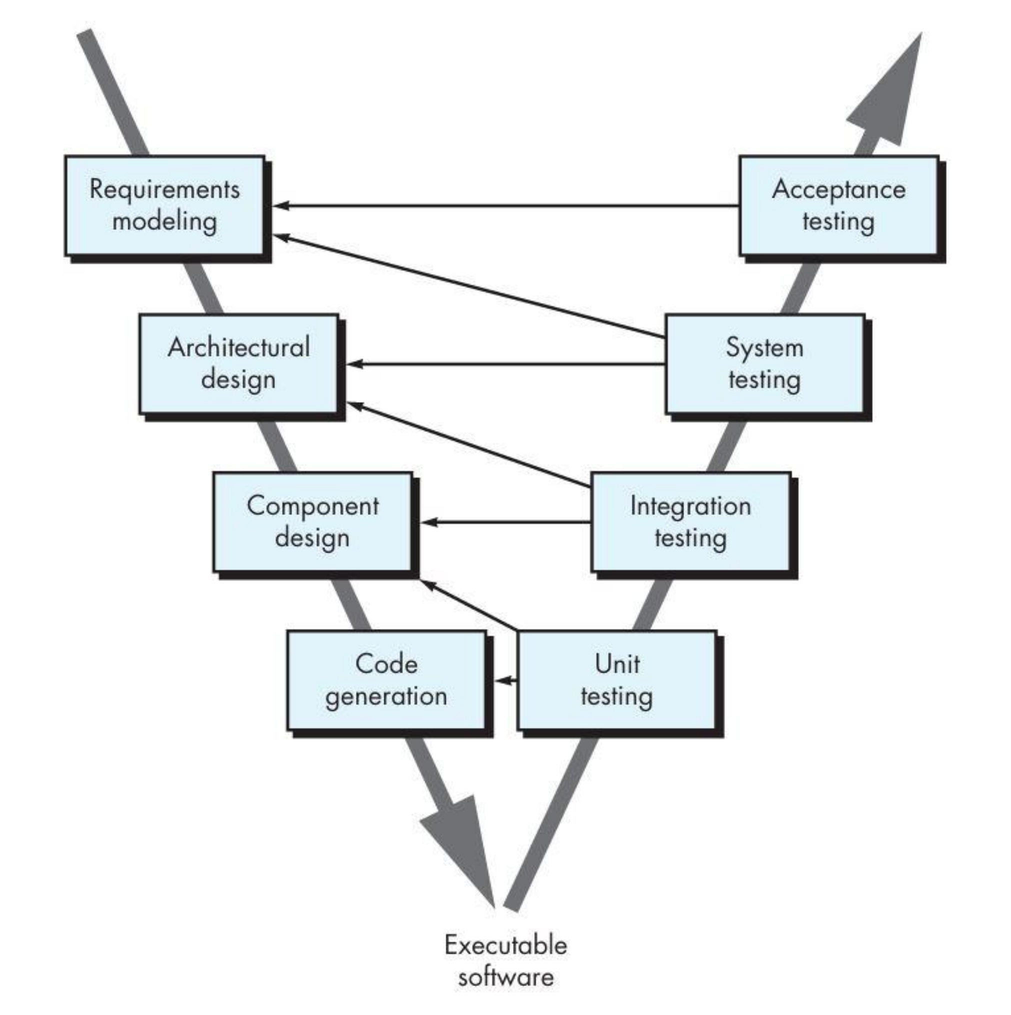

Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

Aspect	✅ Advantages	❌ Disadvantages
Simplicity	Easy to understand & implement.	Too rigid and inflexible for real-world applications.
Best for Small Projects	Works well for small & defined projects.	Not suitable for large or complex systems.
Management	Easy to manage with clear phases.	Changes are difficult to implement.
Structured Approach	Each phase is well-documented and follows a logical order.	No overlapping of phases, which slows down development.
Requirement Stability	End goal is determined early, avoiding scope creep.	Requires complete & accurate requirements upfront, which is often unrealistic.
Testing	Testing is done after development, reducing confusion.	Testing occurs late, making error detection costly.
Risk Management	Basic framework for other models	Risks are not assessed, leading to high uncertainty.
Time to Market	Predictable timeline due to its structured nature.	Working software is not available for a long time, delaying user feedback.

V Model (Verification and Validation Model)

V Model (Verification and Validation Model)

- A structured and disciplined SDLC approach.
- A sequential execution - each phase starts only after the previous phase ends.
- Testing is integrated with development - each development phase has a corresponding testing phase.
- The model forms a V-shape, where:
  - Left side → Represents development activities.
  - Right side → Represents testing activities.
  - Coding phase → Acts as the connection between both sides.
- Ensures early defect detection and better quality control.
- Works best for small to medium-sized projects with well-defined requirements.

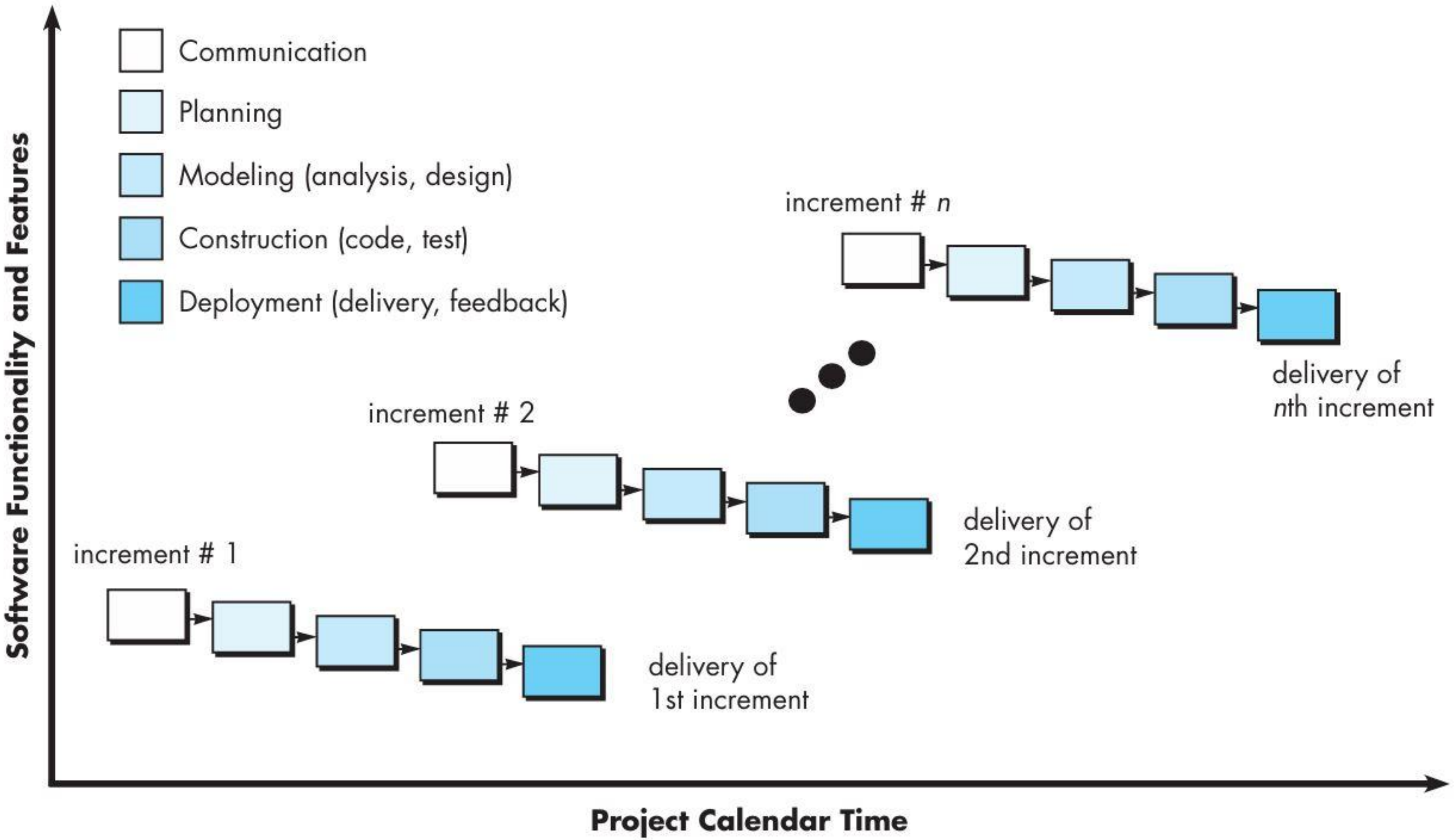

Requirements modeling
Acceptance testing
Architectural design
System testing
Component design
Integration testing
Code generation
Unit testing
Executable software

Aspect	✅ Advantages	❌ Disadvantages
Simplicity	Easy to use and understand.	Not suitable for complex projects.
Early Testing	Testing activities start before coding, ensuring better quality.	Changes in requirements are difficult to accommodate.
Discipline	Follows a well-structured phase-by-phase development and testing process.	High risk and uncertainty if requirements are not well-defined.
Defect Detection	Bugs are caught early, reducing cost and effort.	Not flexible for iterative or ongoing projects.
Best for Small & Medium Projects	Works well for small and medium-scale development.	Not suitable for large-scale, evolving projects.

Incremental Model

Incremental Model

- Divides the development process into multiple increments (modules/builds).
- Each increment follows the same SDLC phases (Requirement Analysis, Design, Implementation, Testing, Deployment).
- Development happens in small parts, making it suitable for complex projects.
- Instead of building the entire software at once, modules are delivered in stages, incorporating customer feedback.
- The process involves:
  - Developing an initial version with core functionalities.
  - Gathering user feedback and refining the system.
  - Adding more modules in each increment until the final system is complete.
- Each new release builds on the previous one, ensuring gradual improvement.
- Important functionalities are prioritized and developed in early iterations.

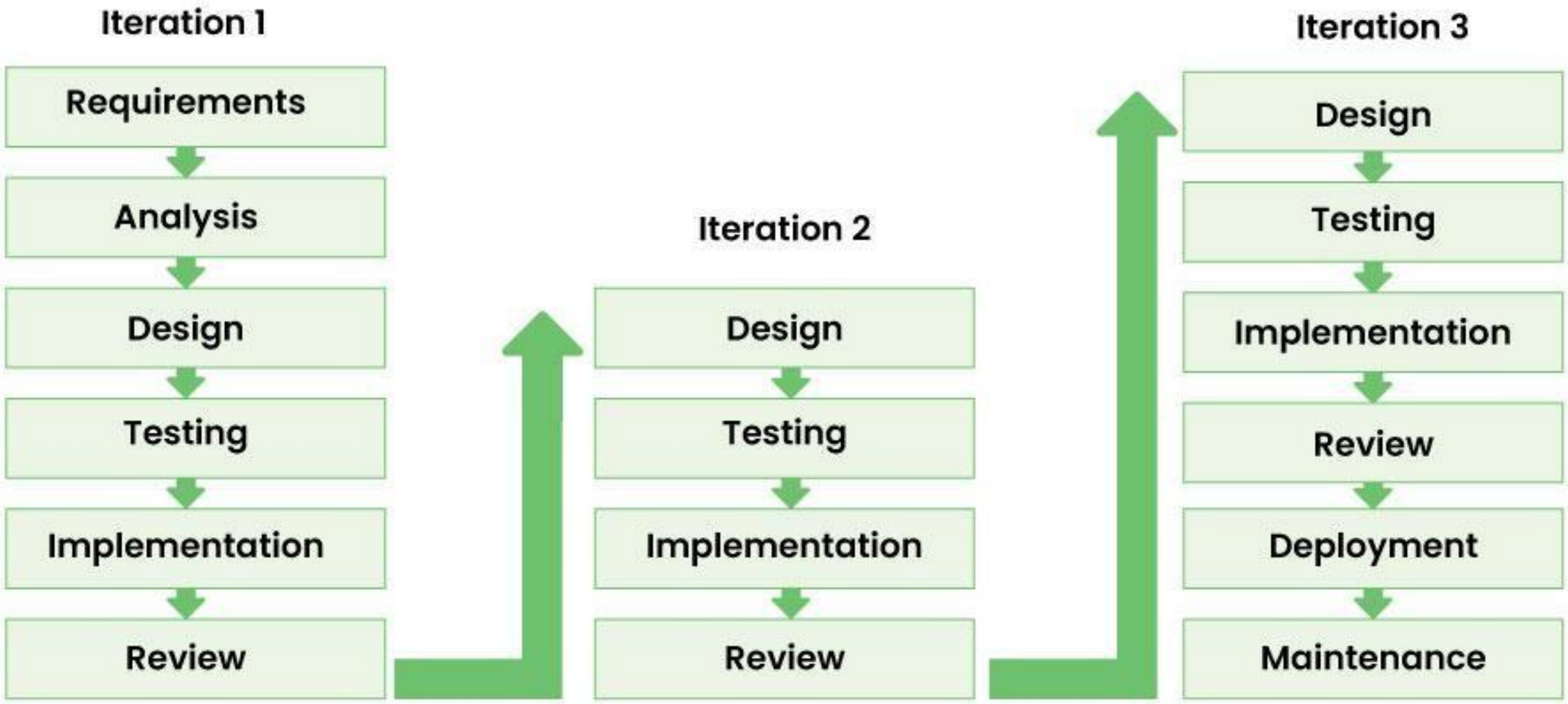

Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
delivery of
nth increment
increment # 2
delivery of
2nd increment
increment # 1
delivery of
1st increment
Software Functionality and Features
Project Calendar Time

Aspect	✅ Advantages	❌ Disadvantages
Development Approach	Critical modules are developed first, ensuring early functionality.	Requires complete clarity of overall software requirements before starting.
Early Working Software	Software is available early in the SDLC.	Needs continuous management and coordination.
Flexibility	Easy to modify scope and requirements.	Requires careful planning and design.
Customer Feedback	Users can review each module and suggest changes.	Late-stage changes may still impact overall system integration.
Progress Measurement	Project progress is easily trackable.	Total cost may be higher compared to simpler models.
Testing & Debugging	Easier to test and debug smaller iterations.	May lead to rework if early increments are not well-integrated.
Error Detection	Defects can be identified early.	Integration of multiple increments can be challenging.

Iterative Models

Evolutionary/Iterative Models

- The Iterative Model involves repeated cycles (iterations) of software development.
- Developed in small, manageable portions, allowing for frequent reviews & changes.
- Process:
  - Develop a version of the software with initial requirements.
  - Review the version and identify required changes.
  - Based on feedback, create a new version in the next iteration.
  - Repeat this process until the final product meets all requirements.
- Unlike the Waterfall Model, the Iterative Model allows for continuous improvement through each cycle.
- **Basic Concept**: Develop software in small parts (iterations), making adjustments after each cycle.
- **Goal**: Address the drawbacks of Waterfall by providing flexibility to adapt to changing requirements.


Iteration 1

Requirements

Analysis

Design

Testing

Implementation

Review

Iteration 2

Design

Testing

Implementation

Review

Iteration 3

Design

Testing

Implementation

Review

Deployment

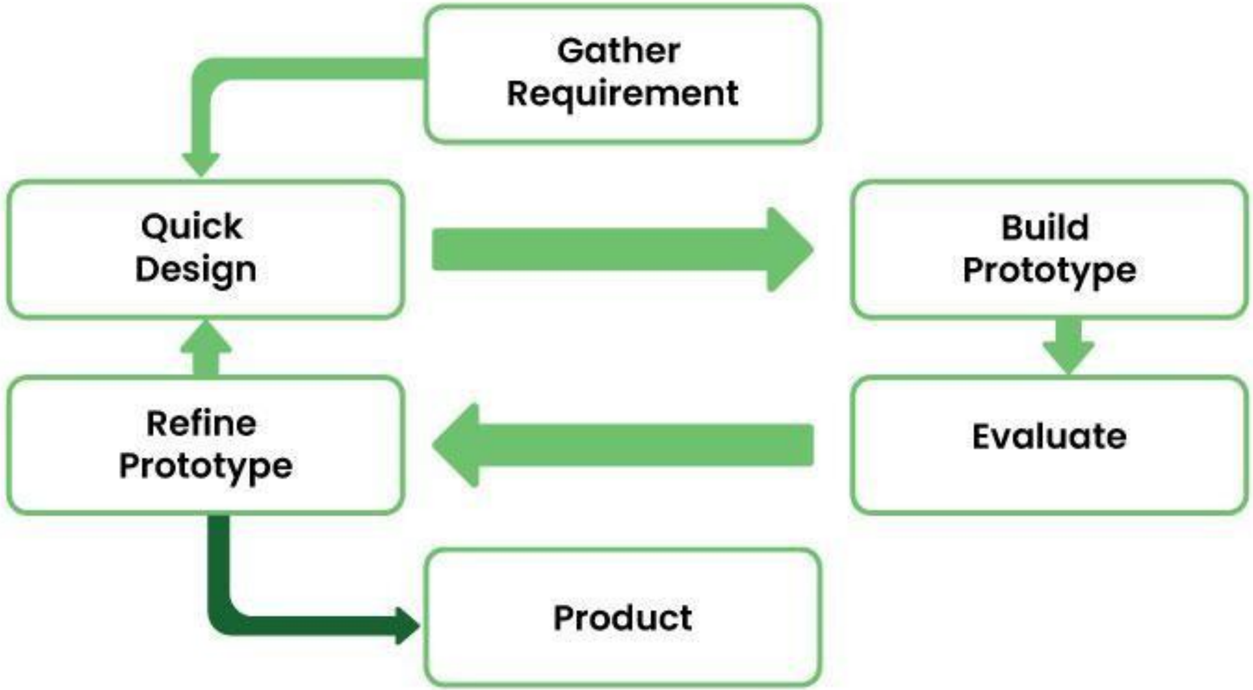
Maintenance

Aspect	✅ Advantages	❌ Disadvantages
Bug Detection	Errors and bugs are identified early in iterations.	Requires more resources due to repeated iterations.
Faster Development	Basic working software is available quickly.	Not suitable for small projects.
Testing & Debugging	Easier to test and debug in each iteration.	Frequent changes in requirements lead to increased effort.
User Feedback	Users provide feedback during development, improving final product quality.	Constant changes can increase project budget and timeline.
Flexibility	Adapts well to changing requirements.	Managing the entire process is complex.
Less Documentation	More focus on development, less time on documentation.	Hard to predict the exact completion date.
Risk Management	Risks are identified and resolved early.	Requires careful planning and coordination.

Prototyping Model is an Iterative Model

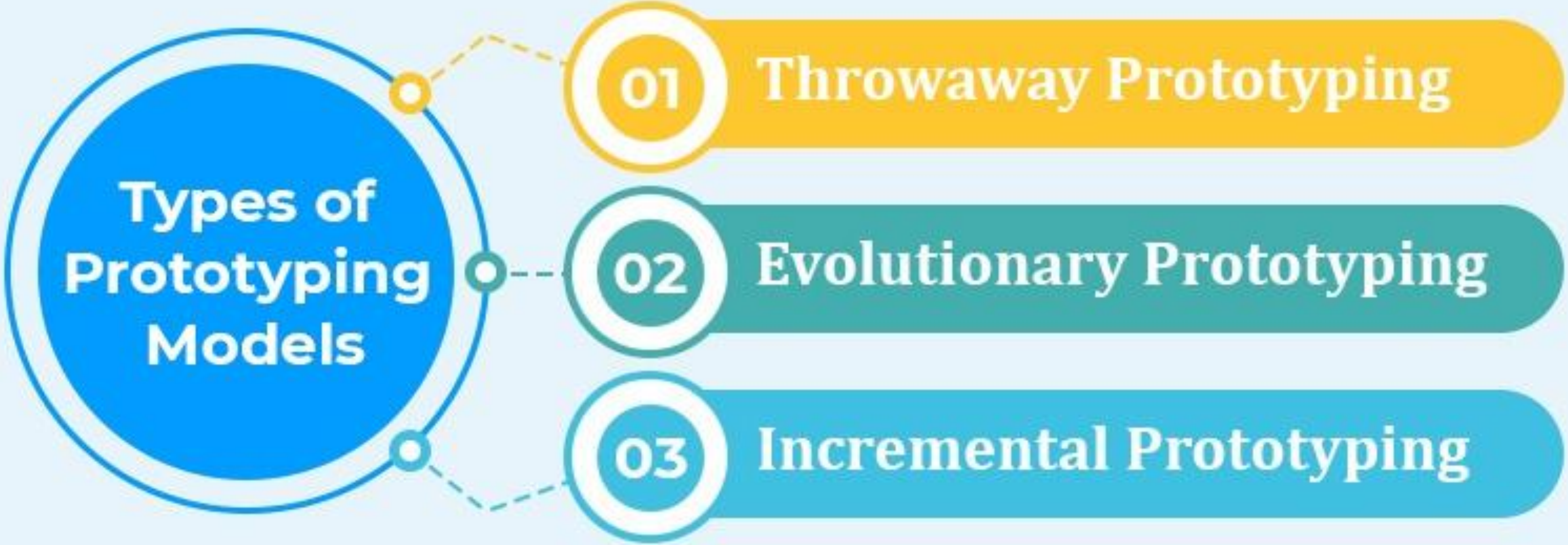
Prototyping Model

- Creating a prototype of the software application before the final product is developed.
- It is especially useful when the requirements are unclear or not well-defined.
- The prototype serves as a working model for understanding user needs, allowing for iterative refinement.
- Key Features:
  - A prototype is created based on the initial requirements.
  - The final product is developed based on feedback from the prototype.
  - Can be used with other SDLC models or as a standalone approach.
- Challenges:
  - If end-users are unsatisfied, a new version must be created, which can lead to increased costs and time.
  - Requires careful management of user expectations to avoid excessive iterations.
- Developed to overcome the limitations of the Waterfall Model, especially when dealing with uncertain or evolving requirements.


Gather Requirement
Quick Design
Build Prototype
Evaluate
Refine Prototype
Product

Aspect	✅ Advantages	❌ Disadvantages
User Interaction	Best suited for applications requiring **human-machine interaction**.	Customers may focus on minor fixes instead of allowing a full system redesign.
Handling Unclear Requirements	Useful when only **general objectives** are known but details (input, processing, output) are unclear.	The first version may include many compromises.
Testing Feasibility	Helps developers evaluate **algorithm performance** and **OS adaptability**.	Developers may make shortcuts just to get a prototype working, leading to poor long-term quality.
Early User Feedback	Users can review and suggest changes early in development.	Too many iterations can delay final product delivery.
Risk Reduction	Helps identify potential risks early.	Not suitable for projects requiring strict documentation and planning.

Types of Prototyping



Types of Prototyping Models

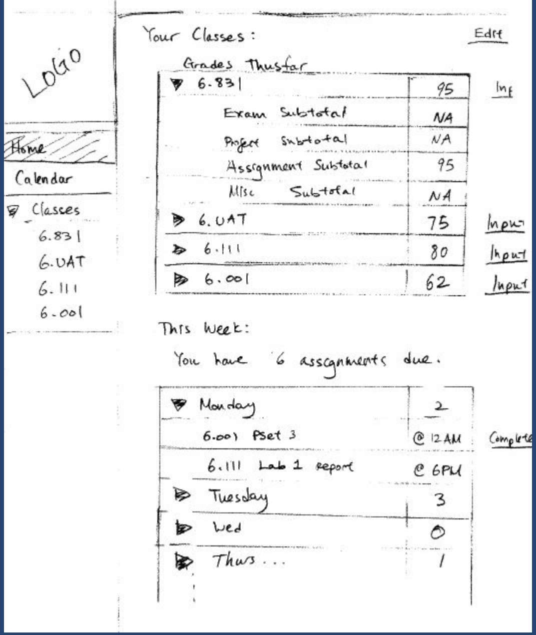
01 Throwaway Prototyping

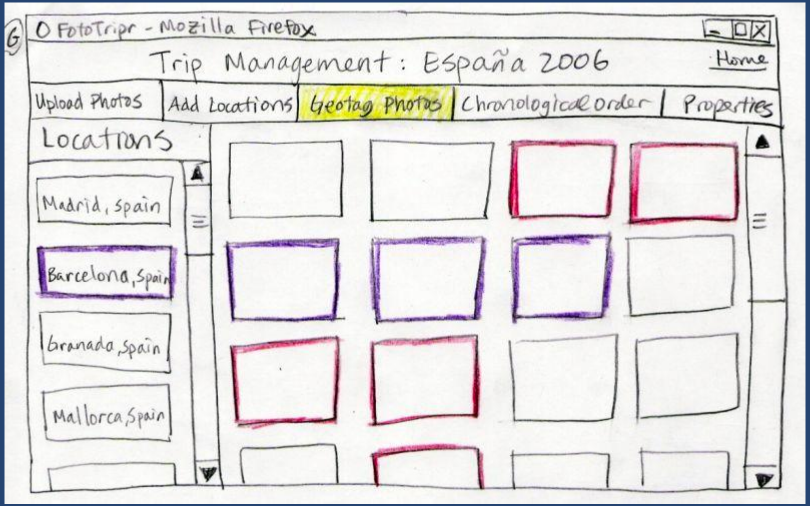
02 Evolutionary Prototyping

03 Incremental Prototyping

Rapid Prototyping Techniques

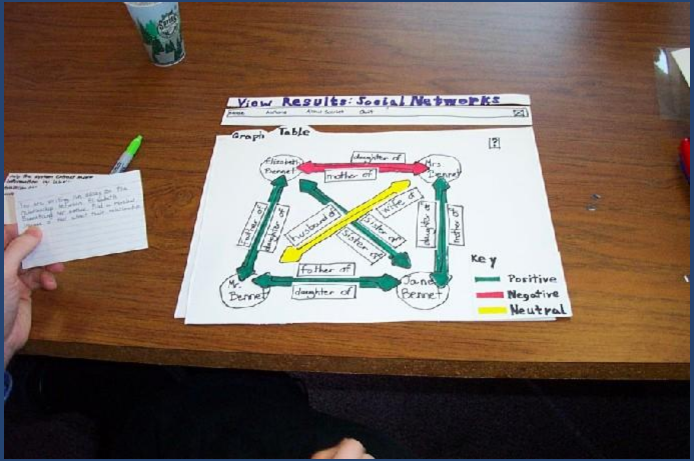
Sketches

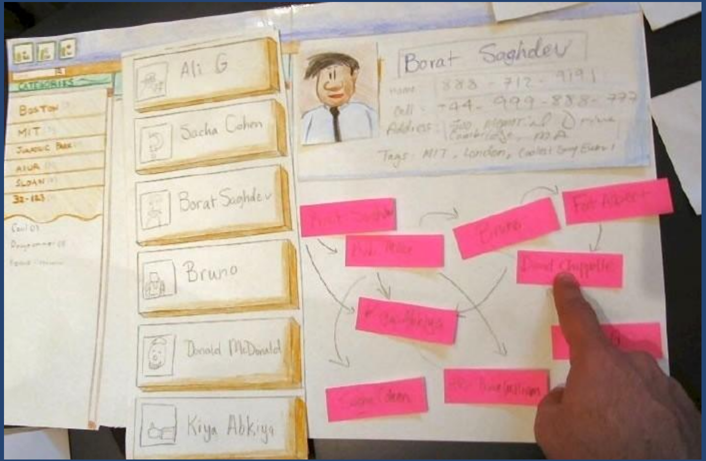




Rapid Prototyping Techniques

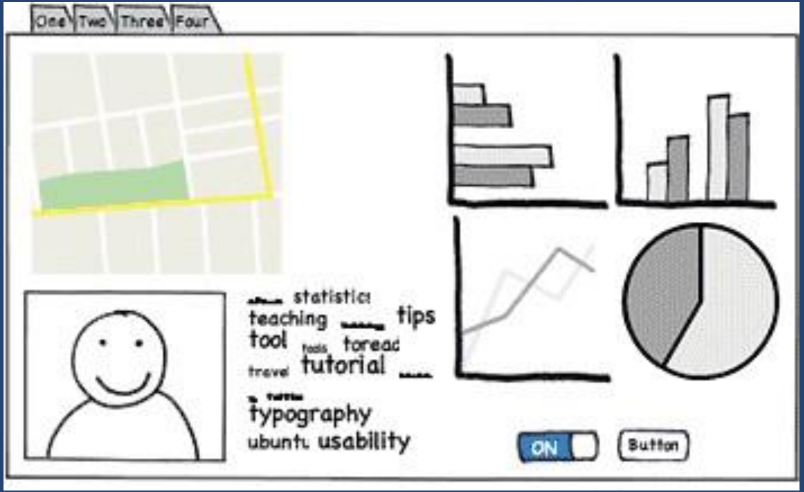
Paper Prototypes

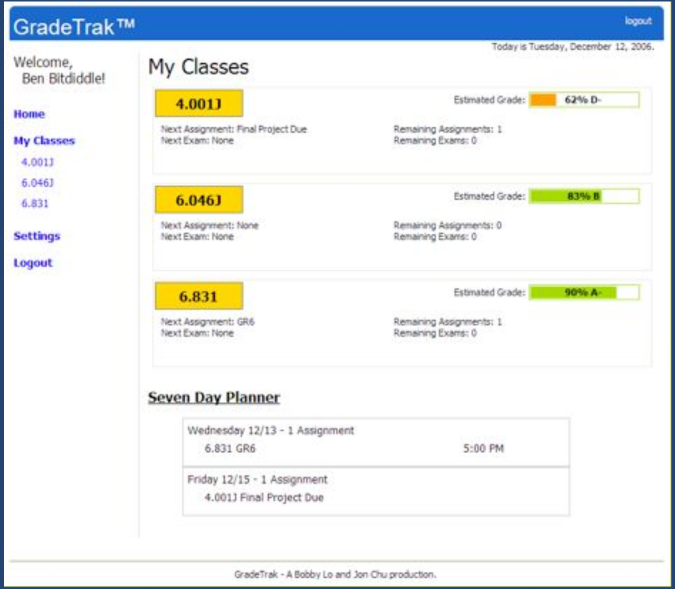




Rapid Prototyping Techniques

Wireframes and Digital Mockups





Today is Tuesday, December 12, 2006.

Welcome,
Ben Bitdiddle!

My Classes

4.001J
Estimated Grade: 62% D-
Next Assignment: Final Project Due
Next Exam: None
Remaining Assignments: 1
Remaining Exams: 0

6.046J
Estimated Grade: 83% B
Next Assignment: None
Next Exam: None
Remaining Assignments: 0
Remaining Exams: 0

6.831
Estimated Grade: 90% A-
Next Assignment: GR6
Next Exam: None
Remaining Assignments: 1
Remaining Exams: 0

Seven Day Planner

Wednesday 12/13 - 1 Assignment
6.831 GR6 5:00 PM

Friday 12/15 - 1 Assignment
4.001J Final Project Due

GradeTrak - A Bobby Lo and Jon Chu production.

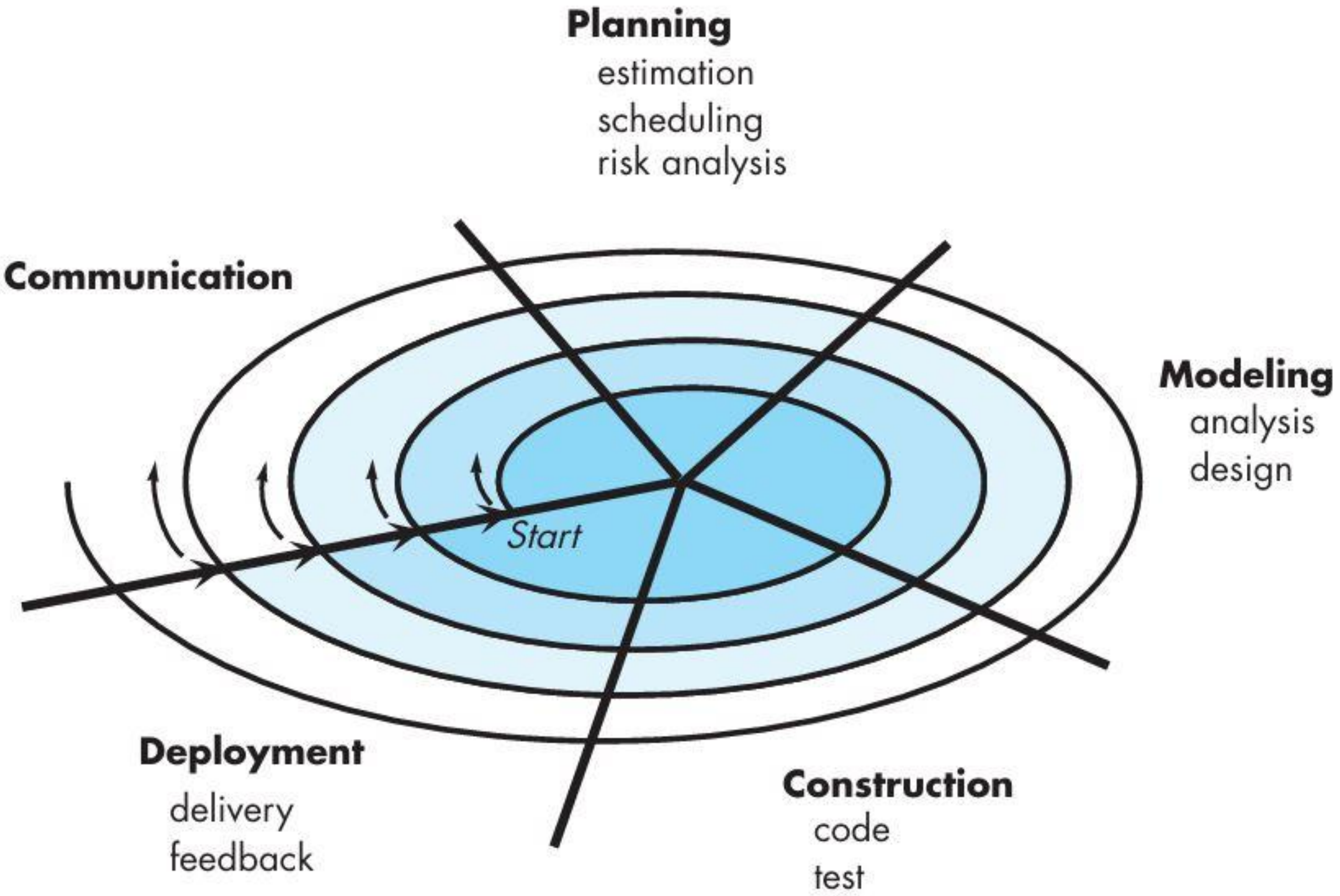
Rapid Prototyping Techniques

- Throwaway Code
  - Quick low-quality code is written just to demonstrate functionality.
  - Used in software engineering to test algorithms or interactions.
- 3D Prototyping (For Hardware & VR Systems)
  - Used for hardware, IoT, or game development.

Spiral Model is also an Iterative Model

Spiral Model

- The Spiral Model combines features of both Iterative and Waterfall Models.
- Ideal for large and complex projects that have evolving requirements.
- Named Spiral because its process resembles a spiral:
  - Curved lines start from a central point and make multiple loops around it.
  - The number of loops depends on the project's size and changing requirements.
  - Each loop is a phase of the software development process.
- **Iterations**: The project goes through multiple loops (iterations), with each loop creating a more complete version of the software.
- **Key Feature: Risk Handling** – Risks are identified in each phase and resolved using prototyping.



Planning
estimation
scheduling
risk analysis

Communication

Modeling
analysis
design

Start

Deployment
delivery
feedback

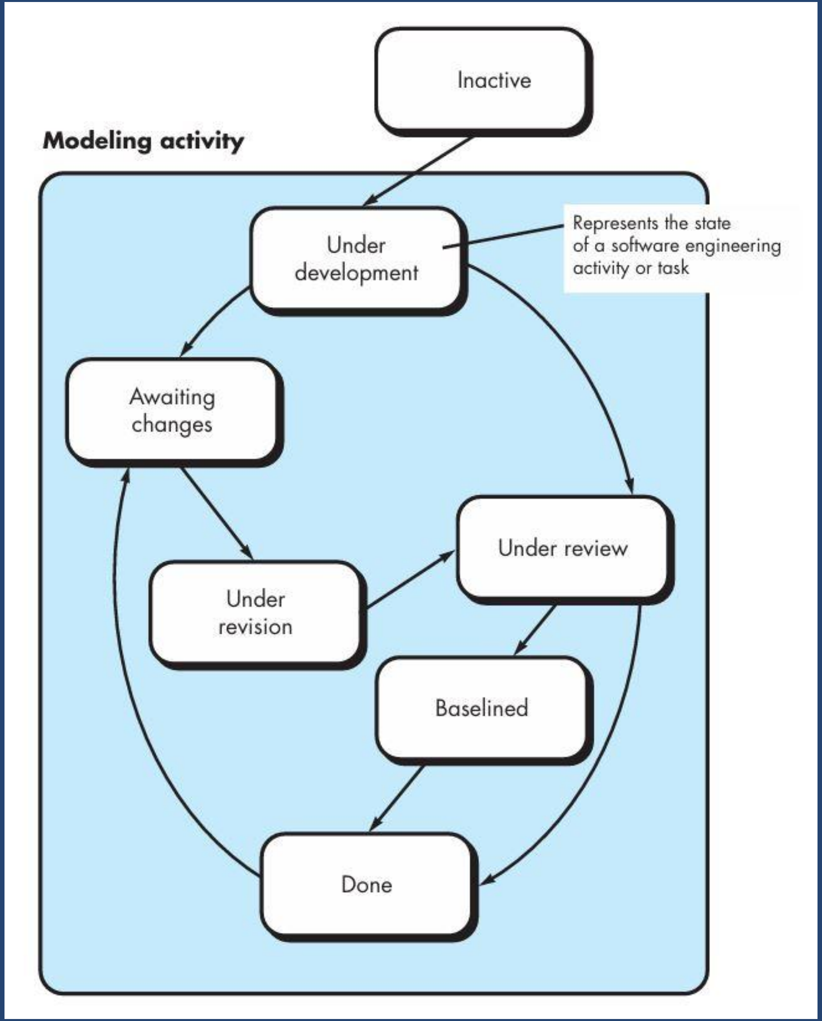
Construction
code
test

Aspect	✅ Advantages	❌ Disadvantages
Flexibility	Additional functionality and changes can be made at later stages.	Highly complex model, making it difficult to manage.
Best for Large Projects	Suitable for large and complex projects.	Not suitable for small projects.
Cost Estimation	Easier to estimate project costs.	High development cost compared to other models.
Risk Management	Risk analysis is performed at each phase, reducing uncertainties.	Success heavily depends on accurate risk analysis.
Early Visualization	Customers can see a working version of the software in early stages.	Requires experienced professionals for evaluation and decision-making.
Customer Satisfaction	Continuous feedback ensures the final product meets customer expectations.	Requires extensive documentation, increasing overhead.

Concurrent Model

Concurrent Model

- The Concurrent Model allows different phases of the software development process to happen in parallel instead of sequentially.
- Each activity (e.g., analysis, design, coding, testing) is represented as a state, which can transition between:
    - Under development
    - Under review
    - Awaiting changes
    - Approved
- At any given time:
    - **Requirements Engineering** might be ongoing for some modules while others are complete.
    - **Design & Prototyping** might be happening for one part, while testing occurs for another.
    - **Implementation** might begin before finalizing all requirements, allowing parallel work.


Inactive
Modeling activity
Under development
Represents the state of a software engineering activity or task
Awaiting changes
Under review
Under revision
Baselined
Done

Concurrent Model

Aspect	✅ Advantages	❌ Disadvantages
Flexibility	Allows **requirement changes** during development.	Needs strong project coordination.
Parallel Development	Faster progress as multiple teams work parallelly.	Requires careful integration planning.
Risk Management	Early risk identification and mitigation.	Syncing parallel tasks can be difficult.
Efficient Resource Utilization	Multiple teams work at the same time.	High communication overhead.

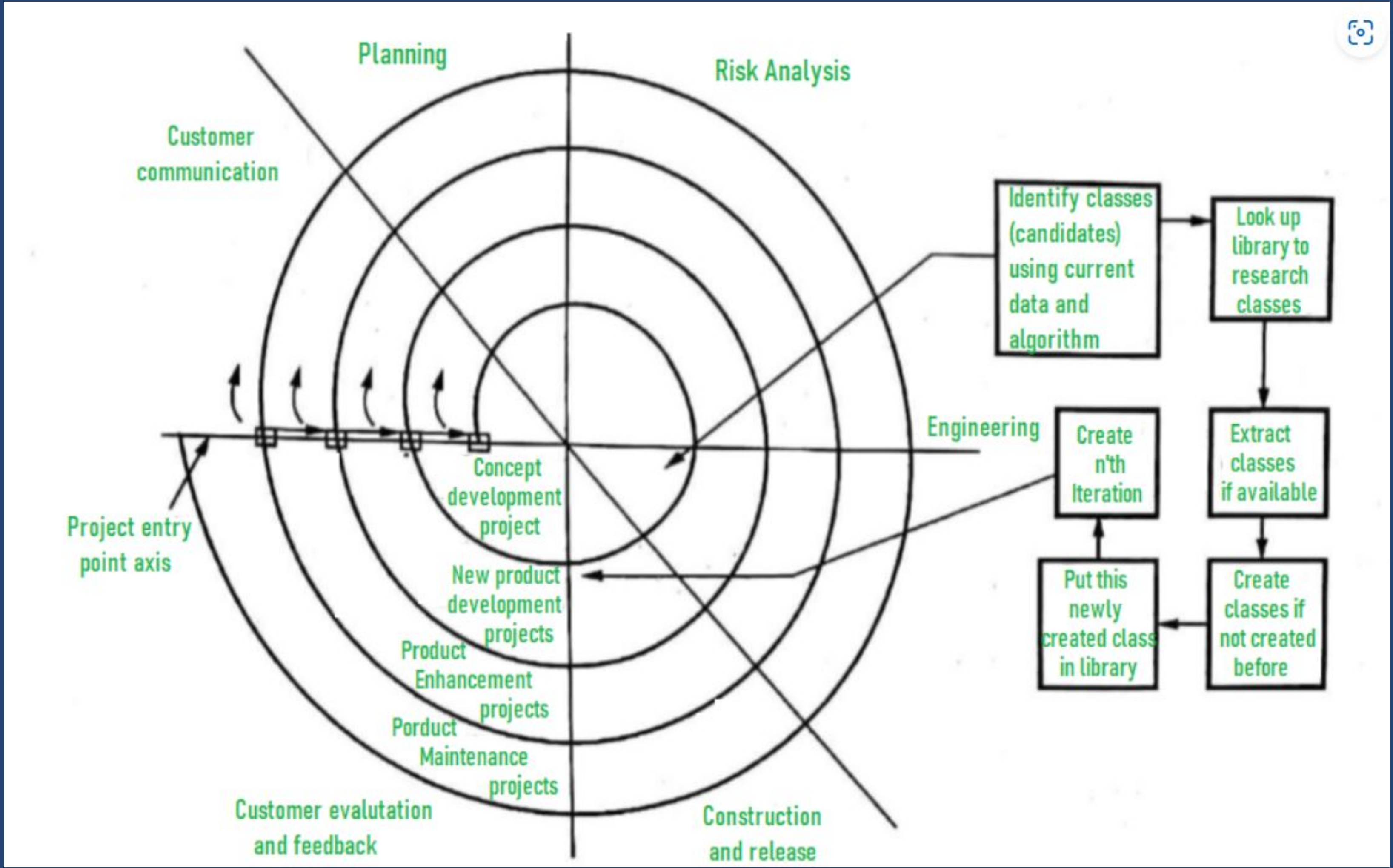
When to Use the Concurrent Model?

- Large-scale, complex systems
  - Aerospace
  - Defense
  - Banking
- Agile environments, where teams develop and release updates continuously
- Projects requiring frequent changes and quick iterations

Component Assembly Model

Component Assembly Model

- The Component Assembly Model is based on reusing existing software components instead of building everything from scratch.
- It follows the principle of “develop once, reuse multiple times.”
- Similar to how hardware engineers assemble circuits using pre-built chips, this model assembles software systems using pre-made components (modules, libraries, APIs, frameworks).


Planning
Risk Analysis
Customer communication
Engineering
Project entry point axis
Concept development project
New product development projects
Product Enhancement projects
Porduct Maintenance projects
Customer evalutation and feedback
Construction and release
Identify classes (candidates) using current data and algorithm
Look up library to research classes
Extract classes if available
Create classes if not created before
Put this newly created class in library
Create n'th Iteration

Component Assembly Model

Aspect	✅ Advantages	❌ Disadvantages
Development Speed	Faster development by reusing components.	Hard to find the perfect component match.
Cost Savings	Reduces cost as components are pre-built.	Licensing fees may apply for third-party components.
Software Quality	Components are tested and reliable.	Compatibility issues when integrating multiple components.
Maintenance	Easier updates by replacing/upgrading components.	Risk of dependency on third-party providers.
Flexibility	Modular design allows easier modifications.	Less control over third-party components' internals.

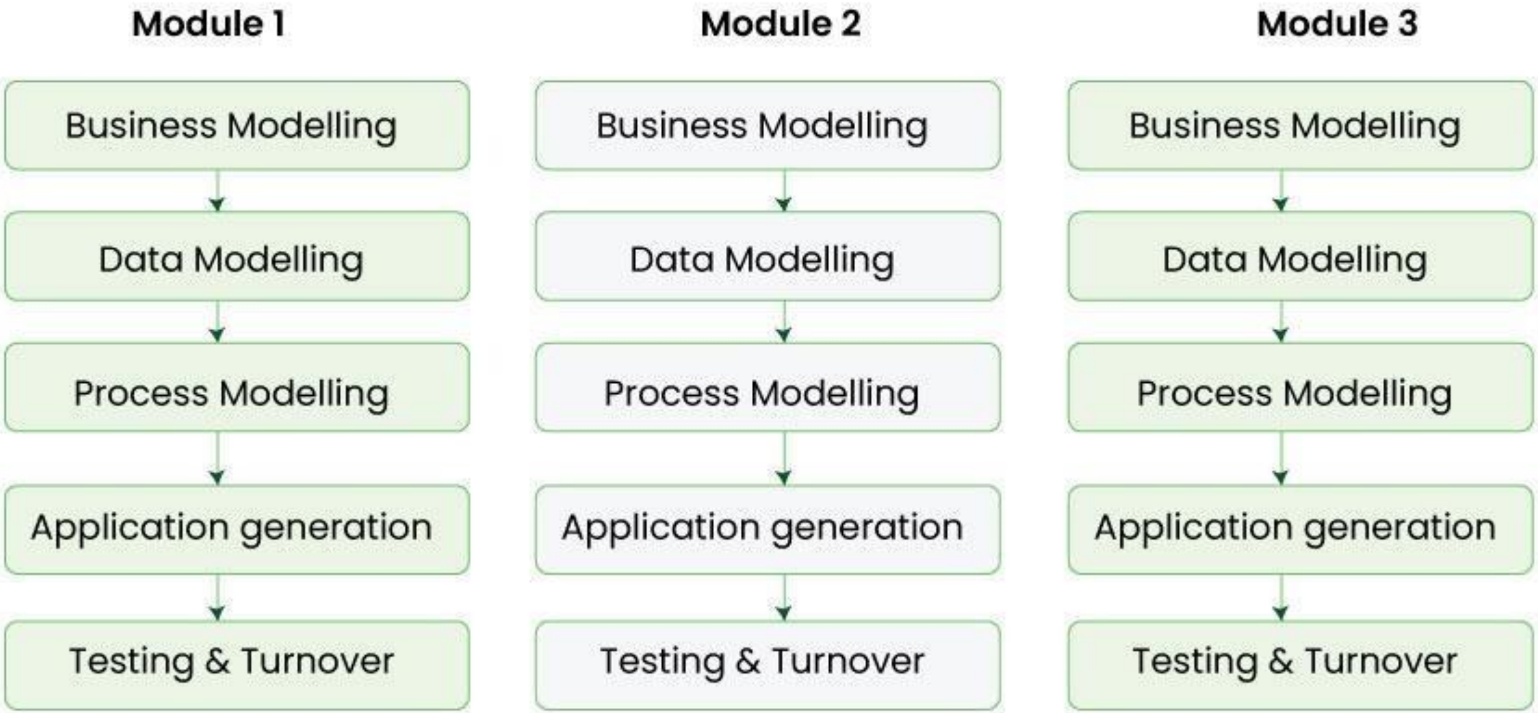
When to Use the Component Assembly Model?

- When there are existing reusable components that fit the project’s needs.
- When fast development is required (e.g., startups, rapid prototyping).
- For large, scalable systems where modularity is important.
- In app development, where frameworks and APIs are widely available.

Rapid Application Development (RAD) Model

Rapid Application Development (RAD) Model

- A fast-paced software development approach.
- Similar to Incremental and Waterfall Models but focuses on speed & flexibility.
- Best suited for small projects; large projects are broken into smaller parts and completed iteratively.
- Key Features:
    - Fast Development – Projects are completed quickly.
    - Predefined Requirements – All requirements are gathered before development begins.
    - Low Error Rate – Rapid development reduces errors.
    - Reuse of Code & Components – Maximizes efficiency by reusing tools and processes.
- Ensures timely delivery while maintaining quality and flexibility.



Module 1

Business Modelling

Data Modelling

Process Modelling

Application generation

Testing & Turnover

Module 2

Business Modelling

Data Modelling

Process Modelling

Application generation

Testing & Turnover

Module 3

Business Modelling

Data Modelling

Process Modelling

Application generation

Testing & Turnover

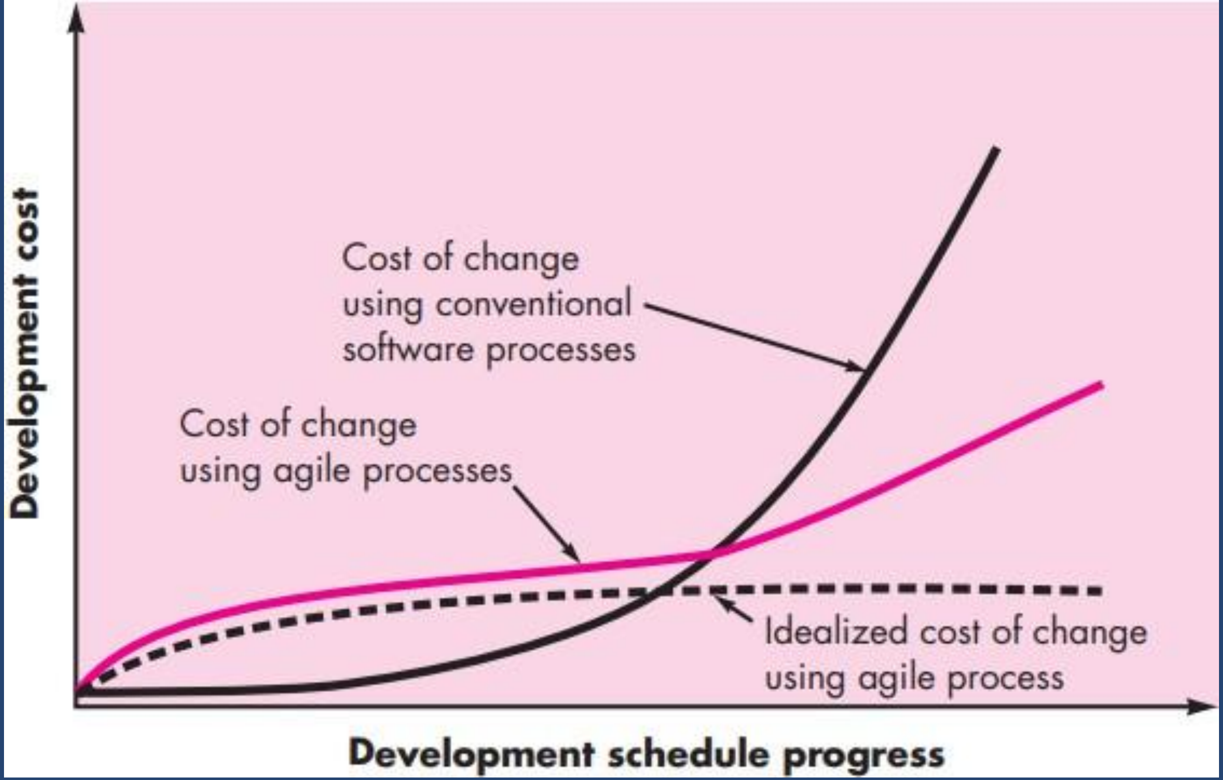
Aspect	✅ Advantages	❌ Disadvantages
Development Speed	Reduces development time significantly.	Requires highly skilled devs and designers.
Reusability	Components are reused, reducing redundancy.	Not suitable for complex, long-term projects.
Flexibility	Easy to modify and update during development.	Difficult to manage due to rapid iterations.
Portability	Uses high-level abstraction and intermediate code, making it easy to transfer.	Automated code generation tools can be expensive.
Quality & Defects	Prototype-based approach minimizes defects.	Requires constant client feedback.
Efficiency	Achieves high productivity with fewer people.	Works only for component-based and scalable systems.
Cost Effectiveness	Reduces overall project costs.	Not ideal for projects with undefined or frequently changing requirements.
Best for Small Projects	Works well for small, well-defined applications.	Unsuitable for large-scale enterprise systems.

Agile Models

WHAT IS AGILITY?

- Response to the changing requirements appropriately
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Organizing a team so that it is in control of the work performed
- Rapid, incremental delivery of software

AGILITY AND THE COST OF CHANGE



WHY IS AGILITY NECESSARY?

- It is difficult to predict in advance which software requirements will persist and which will change. It is equally difficult to predict how customer priorities will change as the project proceeds.
- For many types of software, design and construction are interleaved. It is difficult to predict how much design is necessary before construction is used to prove the design.
- Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.

AGILE PROCESS

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with an emphasis on construction activities
- Delivers multiple software increments
- Adapts as changes occur

AGILITY PRINCIPLES

- Satisfy the customer through early and continuous delivery.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily.

AGILITY PRINCIPLES

- Build projects around motivated individuals. Give them the environment and support needed and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. Sponsors, developers, and users should be able to maintain a constant pace indefinitely.

AGILITY PRINCIPLES

- Continuous attention to technical excellence and good design enhances agility.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

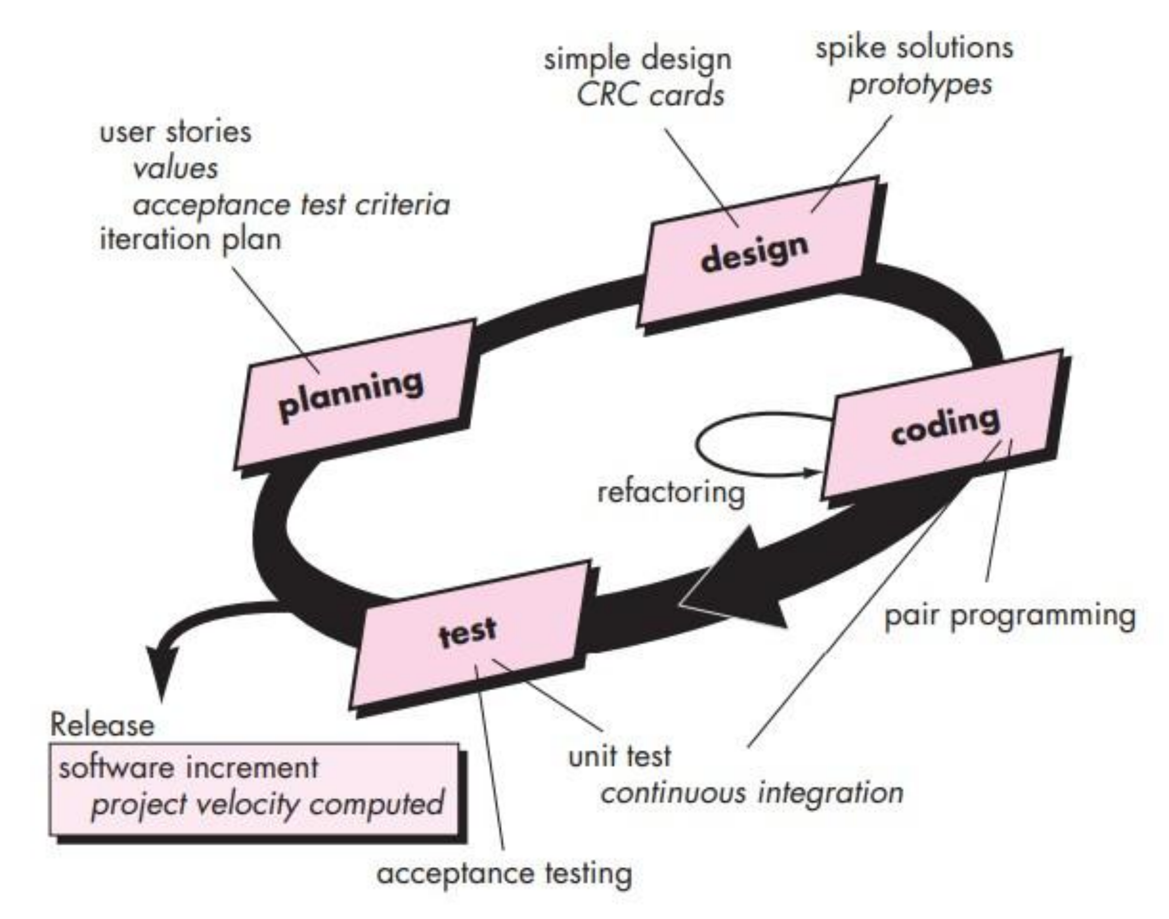
HUMAN FACTOR

- Competence
- Common Focus
- Collaboration
- Decision-making ability
- Fuzzy-problem solving ability
- Mutual Trust and Respect
- Self-organization

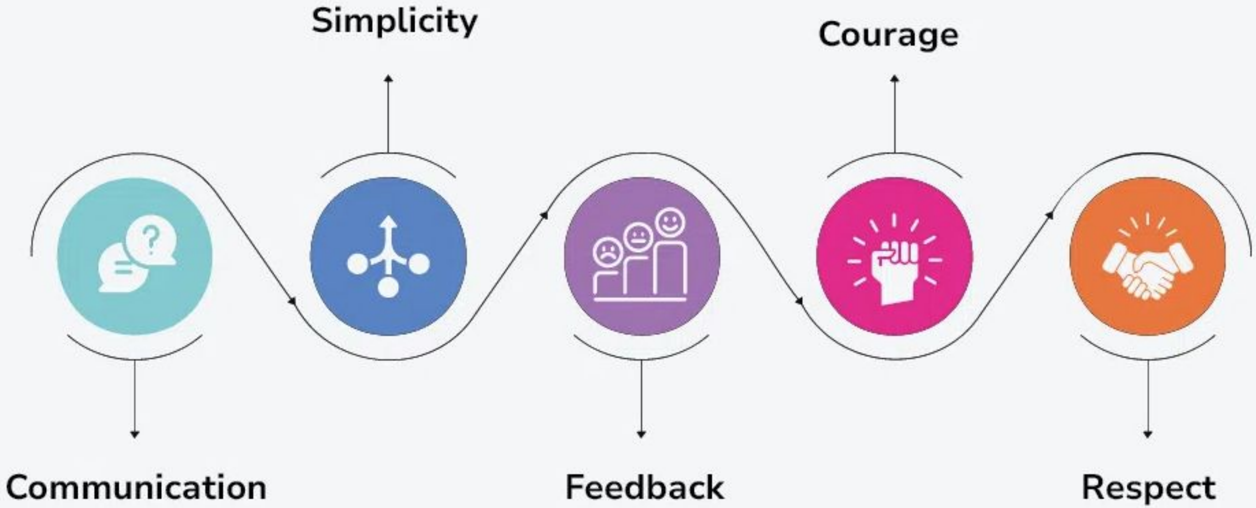
Extreme Programming (XP)

Extreme Programming (XP)

- XP is an agile framework that emphasizes high-quality software through **frequent releases**, **continuous feedback**, and **technical excellence**.
- Best for small to medium-sized teams with rapidly changing requirements
- Key Engineering Practices
    - **Test-Driven Development (TDD)** – Write tests before writing code.
    - **Pair Programming** – Two developers work together on the same code.
    - **Continuous Integration (CI)** – Merge code frequently to avoid integration issues.
    - **Simple Design** – Keep design as simple as possible.
    - **Refactoring** – Continuously improve existing code.
    - **Collective Code Ownership** – Anyone can improve the codebase.
    - **Sustainable Pace (40-hr work week)** – Avoid burnout by having a balanced workload.
    - **On-Site Customer** – The client or a representative is always available for feedback.


simple design
CRC cards
spike solutions
prototypes
user stories
values
acceptance test criteria
iteration plan
design
planning
coding
refactoring
pair programming
test
Release
software increment
project velocity computed
unit test
continuous integration
acceptance testing

Values of Extreme Programming (XP)



Communication

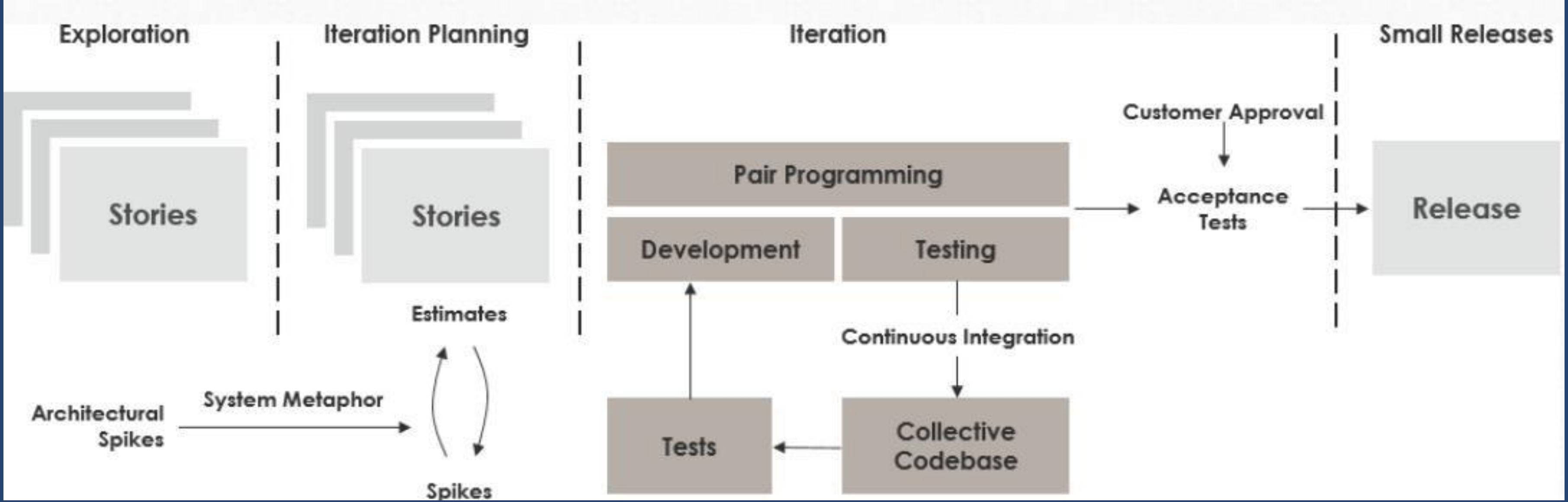
Simplicity

Feedback

Courage

Respect

XP SUMMARY



Exploration
Iteration Planning
Iteration
Small Releases
Stories
Stories
Customer Approval
Pair Programming
Acceptance Tests
Release
Development
Testing
Estimates
Continuous Integration
Architectural Spikes
System Metaphor
Tests
Collective Codebase
Spikes

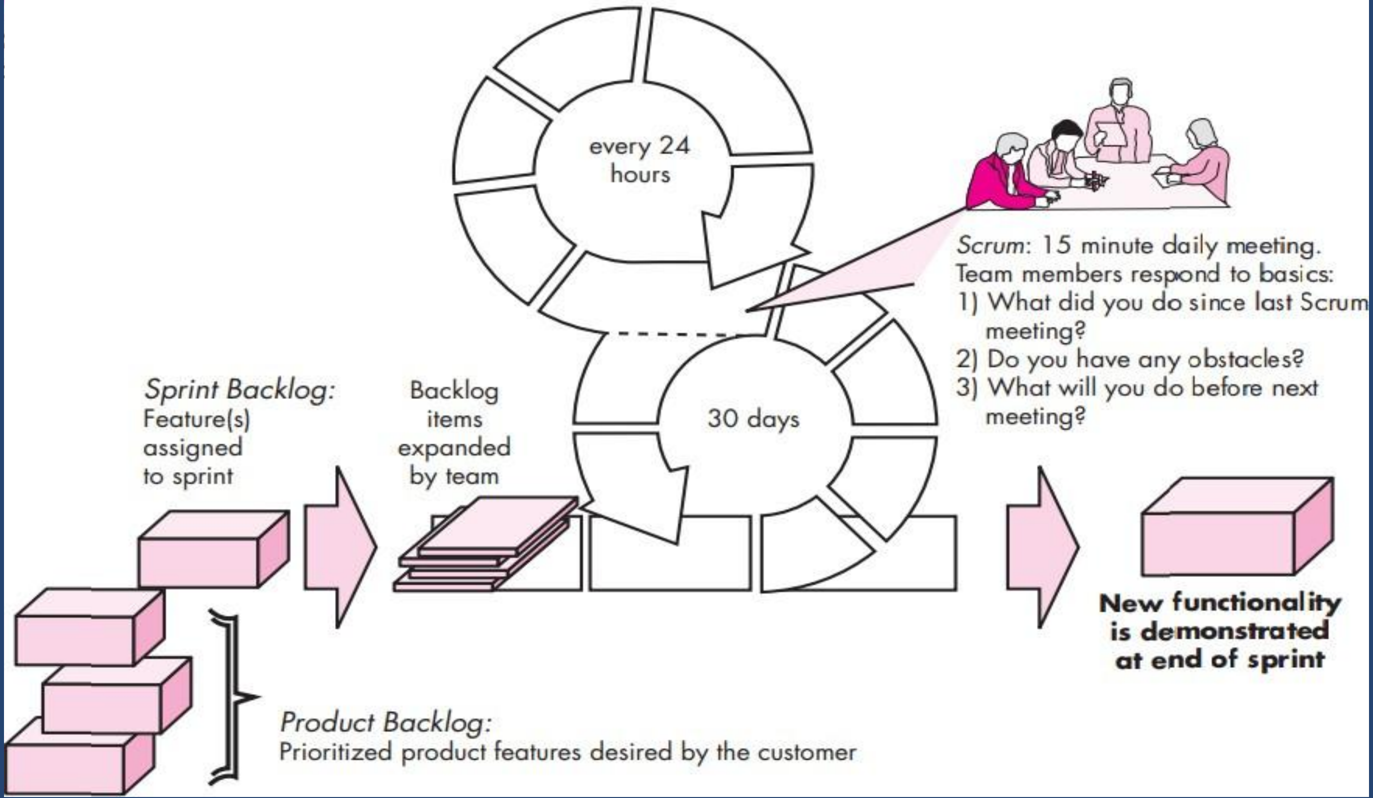
XP Advantages vs Challenges

- Advantages:
  - High customer involvement ensures better product alignment.
  - Improves software quality through continuous testing and refactoring.
  - Faster adaptation to changing requirements.
- Challenges:
  - Requires highly skilled and disciplined developers.
  - Pair programming can be costly.
  - Difficult to scale for large teams.

SCRUM

SCRUM

- Scrum is an agile framework that organizes work in short cycles (**sprints**), emphasizing team collaboration, roles, and structured meetings.
- Key Elements
  - Roles
    - Product Owner (PO) – Defines and prioritizes the product backlog.
    - Scrum Master – Facilitates the process and removes impediments.
    - Development Team – Self-organizing team that delivers the product.
  - Artifacts
    - Product Backlog – List of features/tasks for development.
    - Sprint Backlog – Selected tasks for a sprint.
    - Increment – Working product after each sprint.
  - Events
    - Sprint Planning – Define what will be worked on in the sprint.
    - Daily Standup – Short 15-minute meeting for progress updates.
    - Sprint Review – Present the completed work.
    - Sprint Retrospective – Reflect on what went well and what can improve.



every 24 hours

30 days

Scrum: 15 minute daily meeting.
Team members respond to basics:

1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?

Sprint Backlog:
Feature(s) assigned to sprint

Backlog items expanded by team

New functionality is demonstrated at end of sprint

Product Backlog:
Prioritized product features desired by the customer

SCRUM Advantages vs Challenges

- Advantages:
  - Increases productivity and faster delivery of features.
  - Improves collaboration and transparency.
  - Enhances adaptability to changing requirements.
- Challenges:
  - Requires dedicated commitment from the team.
  - Can be difficult to implement in rigid organizational structures.
  - Risk of scope creep if backlog is not well managed.

XP vs Scrum

Feature	Extreme Programming (XP)	Scrum
Focus	Engineering practices	Project management
Iteration Length	1-2 weeks	2-4 weeks
Team Size	Small to medium teams	Medium to large teams
Customer Involvement	High (on-site customer)	Indirect (via product owner)
Key Practice	Pair programming, TDD	Daily stand-ups, Sprint planning

XP vs Scrum Roles

Role	Extreme Programming (XP)	Scrum
Customer	On-Site Customer (actively involved daily)	Product Owner (manages backlog, represents stakeholders)
Coach	Guides the team on XP practices	Scrum Master (ensures Scrum process, removes impediments)
Tracker	Monitors project progress and team performance	Not a defined role in Scrum
Developer	Writes code, follows TDD, pair programming, and collective ownership	Development Team (self-organizing, cross-functional)
Tester	No separate tester role (developers test using TDD; some teams may include testers)	No separate tester role (testing is done within the Dev Team)

When to Use XP vs Scrum?

- **Use XP if:**
  - You need **strong engineering practices** (TDD, refactoring, pair programming).
  - Customer involvement is **very high**.
  - Requirements change **frequently**.
- **Use Scrum if:**
  - You need a **structured project management approach**.
  - The team is **larger** and cross-functional.
  - Business stakeholders drive priorities.

Advantages of Agile Models

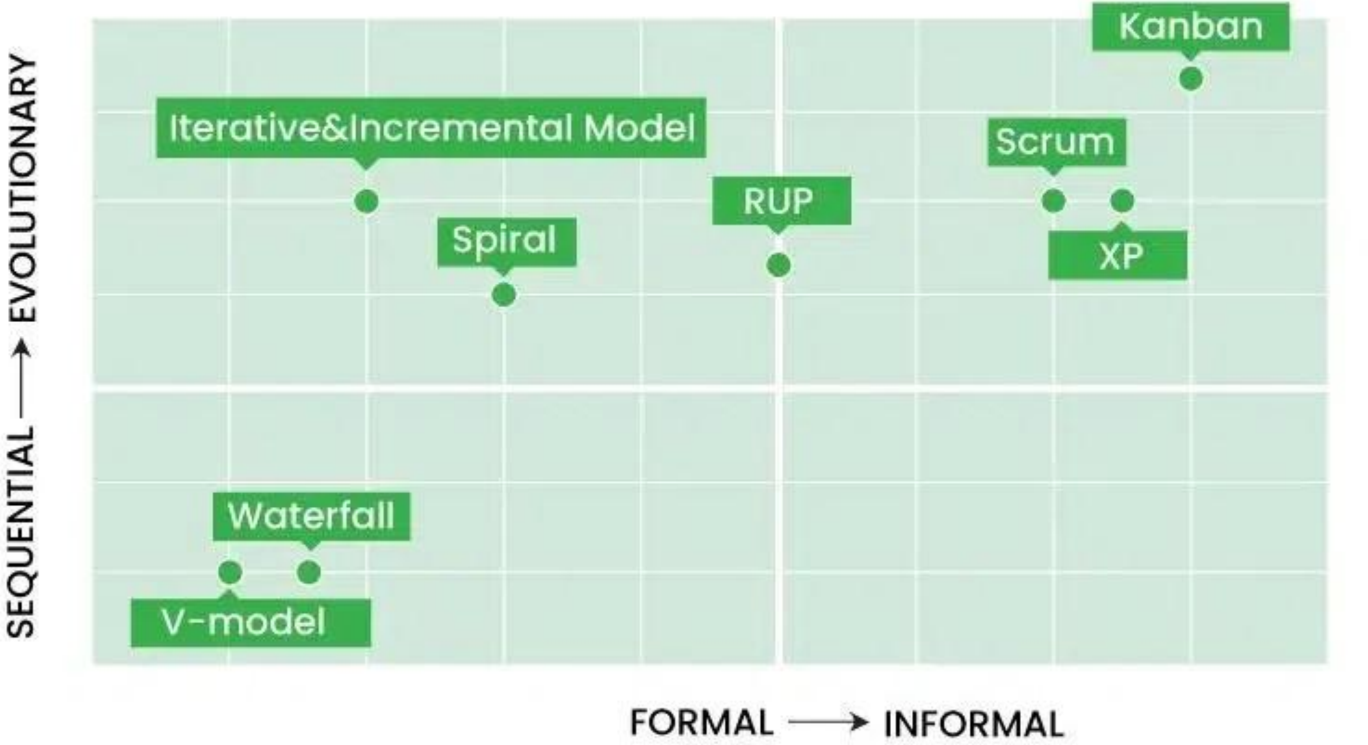
- **Flexibility and Adaptability** – Agile allows for changes based on customer feedback, evolving requirements, and market conditions.
- **Customer Satisfaction** – Continuous involvement of customers ensures that the final product meets their needs.
- **Faster Delivery** – Agile delivers working software in short iterations, allowing faster time-to-market.
- **Improved Quality** – Frequent testing and iterative improvements lead to higher software quality.
- **Better Risk Management** – Early issue detection minimizes project risks.
- **Enhanced Team Collaboration** – Daily stand-ups, retrospectives, and feedback loops improve communication.
- **Continuous Improvement** – Teams reflect and refine their processes regularly for better efficiency.

Disadvantages of Agile Models

- **Lack of Predictability** – Changing requirements can make project timelines and costs difficult to estimate.
- **Scope Creep** – Constant changes may lead to uncontrolled growth in project scope.
- **Requires Skilled Team Members** – Agile relies on self-organizing teams, requiring experienced and disciplined developers.
- **Less Documentation** – Agile prioritizes working software over comprehensive documentation, which can cause issues in large projects.
- **Not Suitable for All Projects** – Large-scale projects with strict regulatory or security requirements may not fit well with Agile.
- **Higher Customer Involvement Required** – Frequent customer input is needed, which may not always be feasible.
- **Integration Challenges** – Continuous changes may lead to integration issues, requiring more testing and refactoring.

IMPORTANT Reading Assignments

- Top 8 Software Development Life Cycle (SDLC) Models used in Industry
- What is Extreme Programming (XP)?
- What is Scrum?


Kanban
Iterative&Incremental Model
Scrum
RUP
Spiral
XP
Waterfall
V-model
SEQUENTIAL ⟶ EVOLUTIONARY
FORMAL ⟶ INFORMAL

THANK YOU