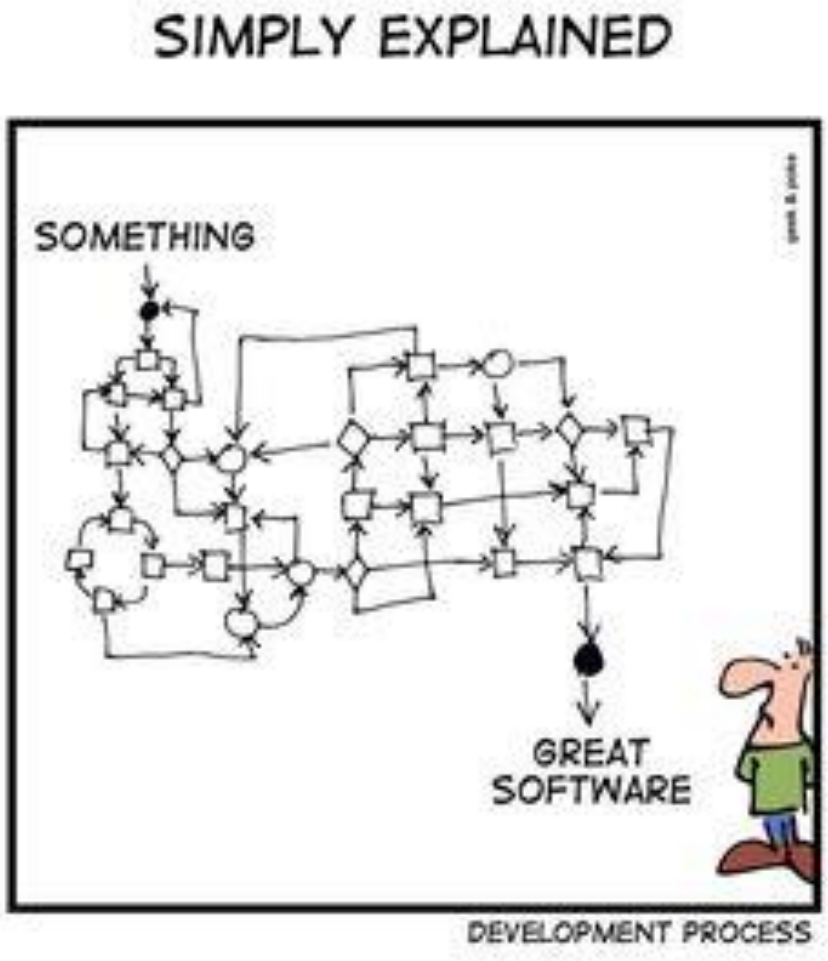
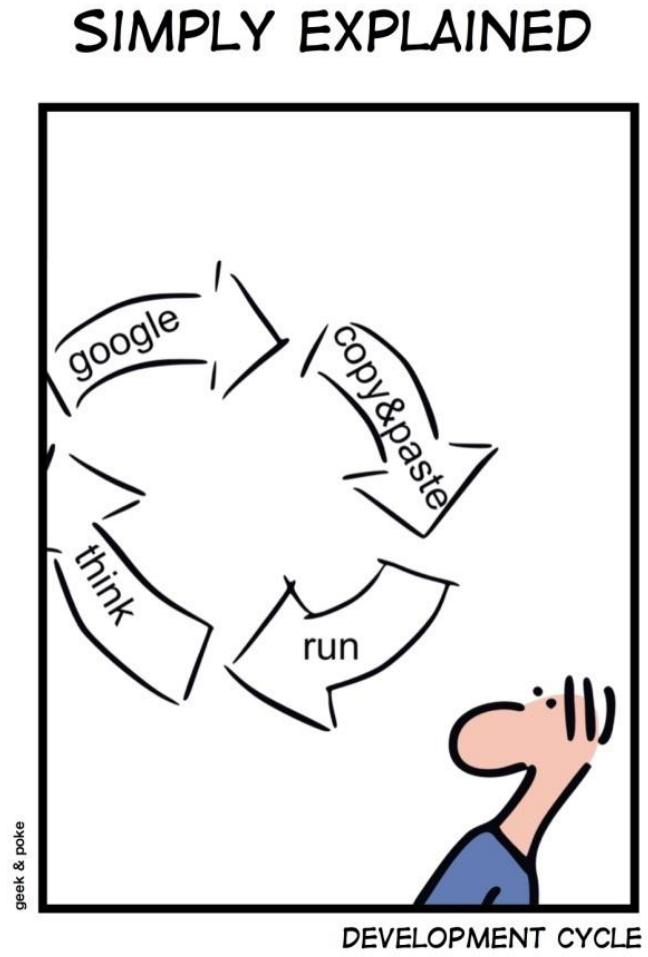
University
Mohammed VI
Polytechnic



Software Development Process

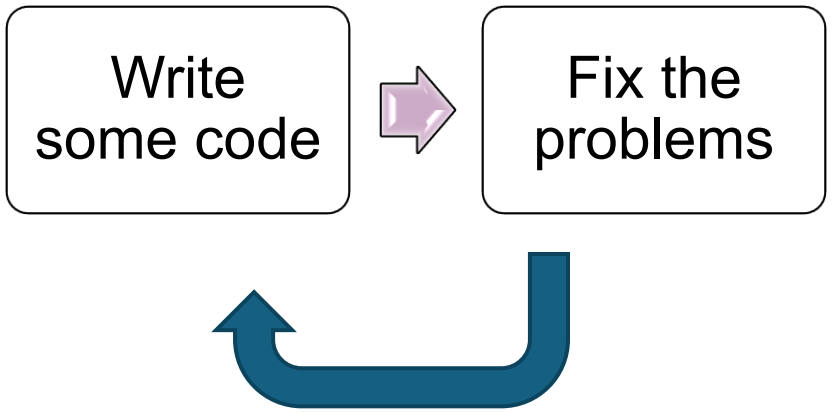
Pr. Imane Fouad

How do we develop small programs for internal operation/personal use?



The code & fix model

Th basic model used in the earliest days of software.

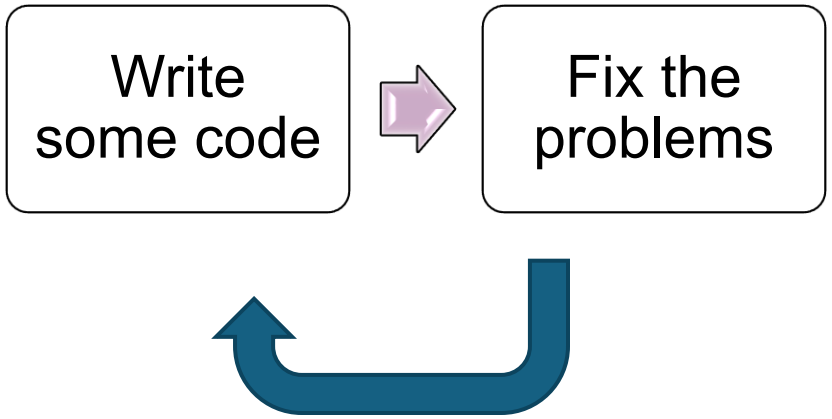


What is wrong with that?

The code & fix model

After a number of fixes, subsequent fixes may become very expensive.

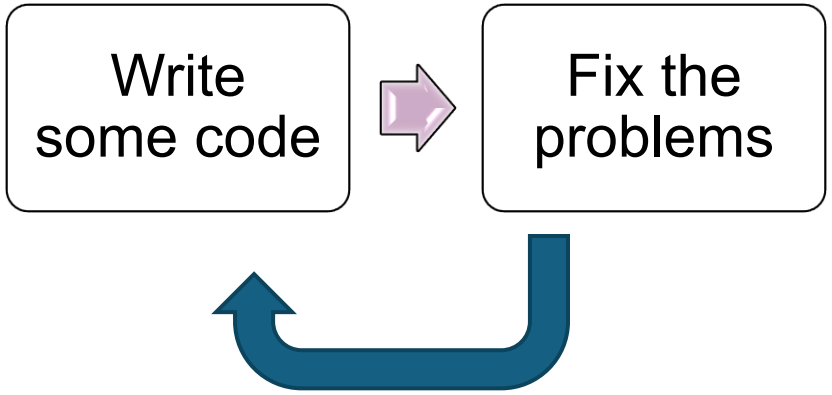
Frequently, the software becomes a poor match to users' needs that it is rejected or redeveloped.



The code & fix model

Code is expensive to fix because of poor preparation for testing and modification.

Lack of requirements analysis and design.



Motivation

Manage large projects effectively

Break down the project and assign tasks properly

Anticipate and handle risks

Reduce complexity

What is a software development process model ?

- A **software development process**, or software development methodology is a set of activities that lead to the creation of a software product.
- Methodologies are also known as the Software Development Life Cycle **(SDLC).**
- A methodology defines the steps in a project and how they are connected.
- It specifies how development activities are assigned to team members.
- Development methodologies depend heavily on the people managing the activities.

What is a software development process model ?

There are many different software processes but most involve these key activities:

Requirements specification – defining what the system should do.

Design and implementation – defining the organization of the system and implementing the system.

Verification & Validation – checking if it works correctly and if it does what the customer wants.

Traditional vs Agile Software Development Methodologies

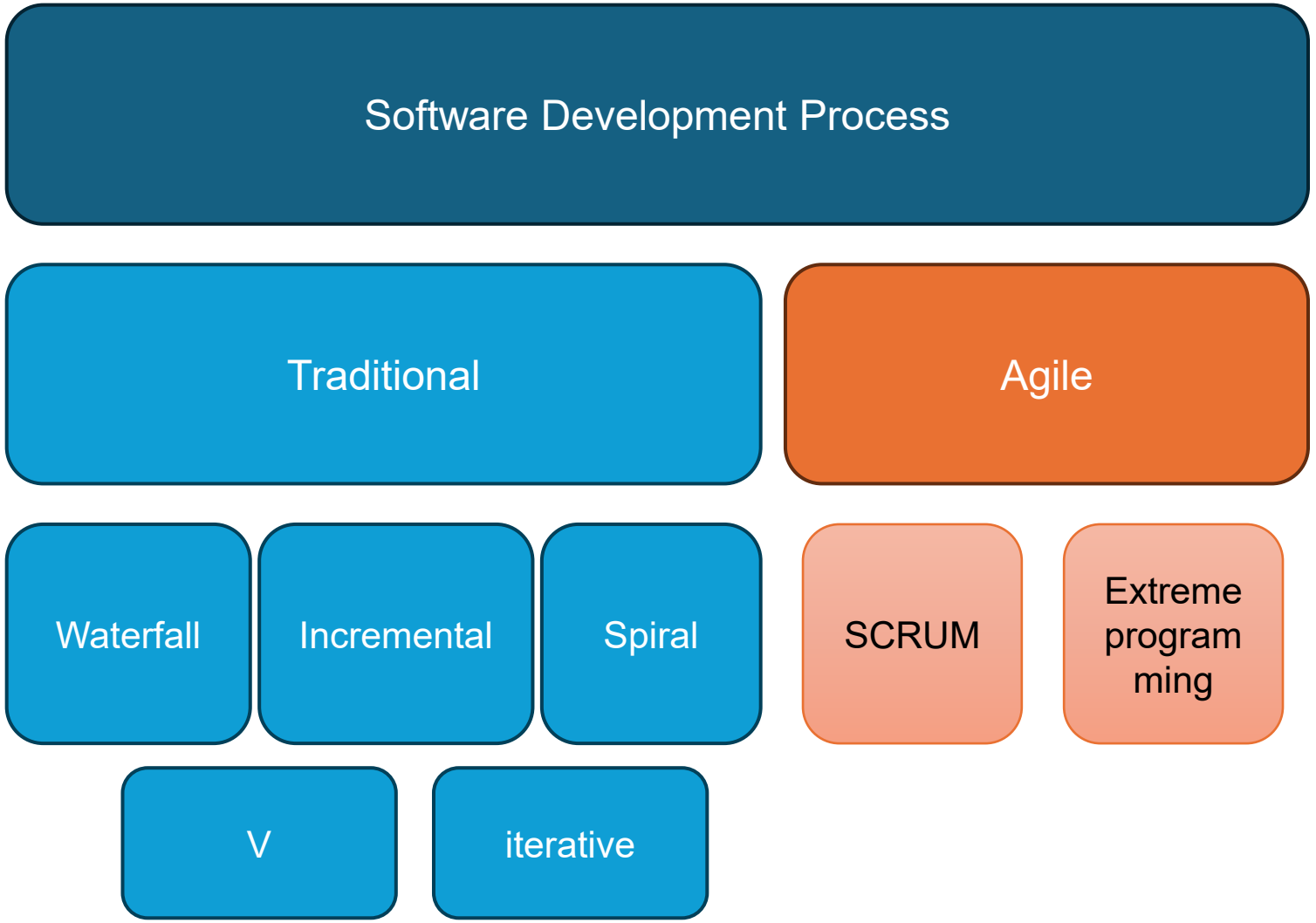
Traditional Methods

- Very clearly defined steps.
- Extensive documentation.
- Works well for **large projects**, especially government or enterprise projects.

Agile Methods

- Incremental and iterative models.
- Small and frequent deliveries.
- Focus more on coding and working software, less on documentation.
- Suitable for **small to medium-sized projects**.

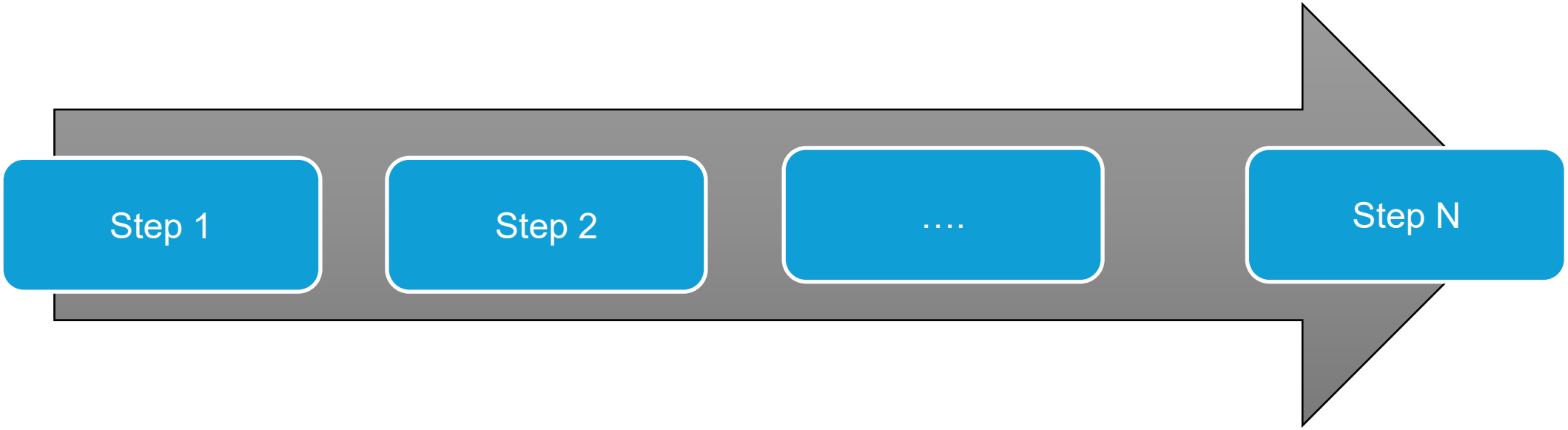
Traditional vs Agile Software Development Methodologies



Typology of Development Models

Sequential Model

- Work is done in a **strict order, step by step**.
- Each phase must be **completed before moving to the next**.



Incremental Model

- The product is built and delivered in **small parts (increments)**.
- Each increment adds **new functionality or features**

Iterative Model

- Development is done in **repeated cycles (iterations)**.
- Each iteration improves the product based on **feedback or evaluation**.

Typology of Development Models

- **Sequential:** Step by step, one phase at a time.
- **Incremental:** Built in small usable parts, adding functionality gradually.
- **Iterative:** Repeated cycles, improving the product each time.

When to Use a Methodology X?

Factors to consider:

- **Nature of the Project** – Is it simple, complex, or innovative?
- **Project Size** – Small, medium, or large team/project scope.
- **Nature of the Client** – Internal, external, government, or private.
- **Contract Requirements** – Strict regulations or flexible expectations.
- **Team Skills** – Experience and technical competence of the team.

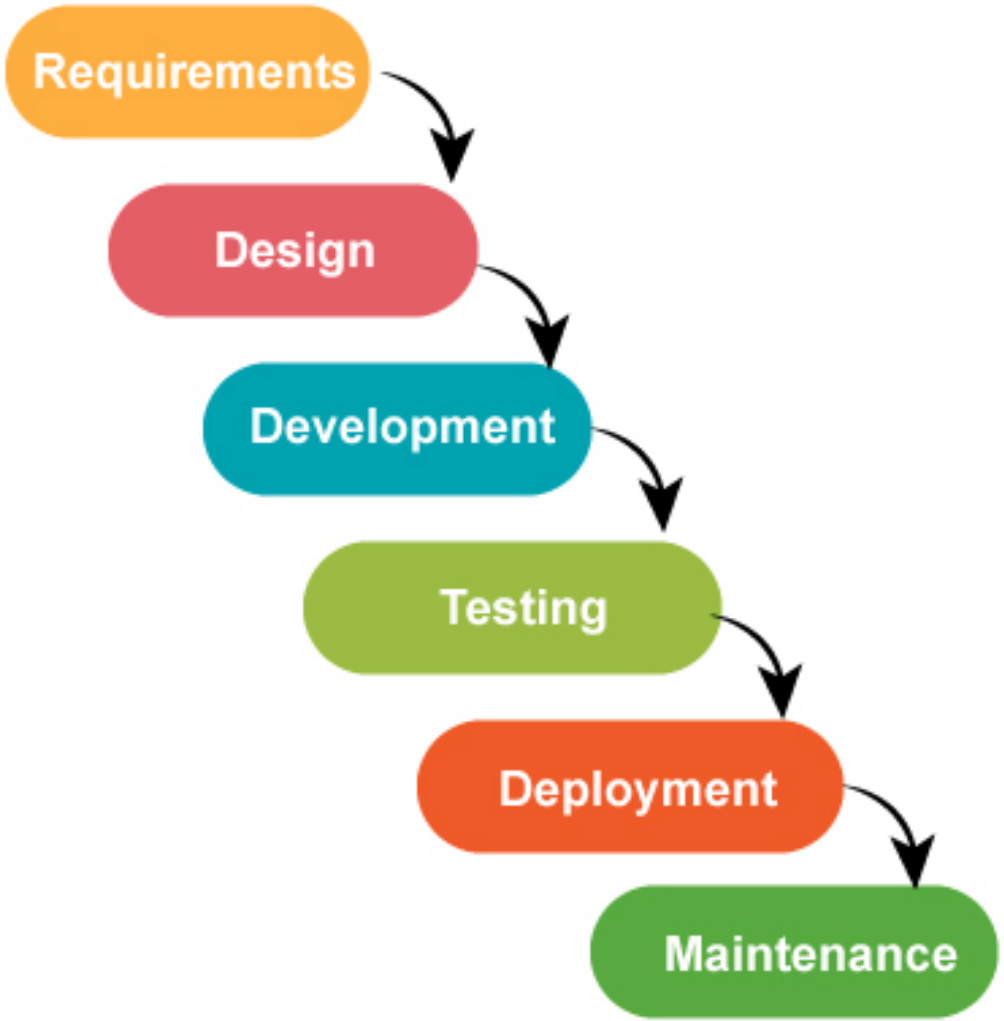
Traditional Software Development Methodologies

Waterfall Model

- One of the first models proposed, inspired by Royce (1970).
- Also called the **Linear Model**.
- Each phase produces a set of **deliverables** (documents, code, designs).
- A phase can only start **after the previous one is fully completed**.
- Considered the **academic model** of software development.

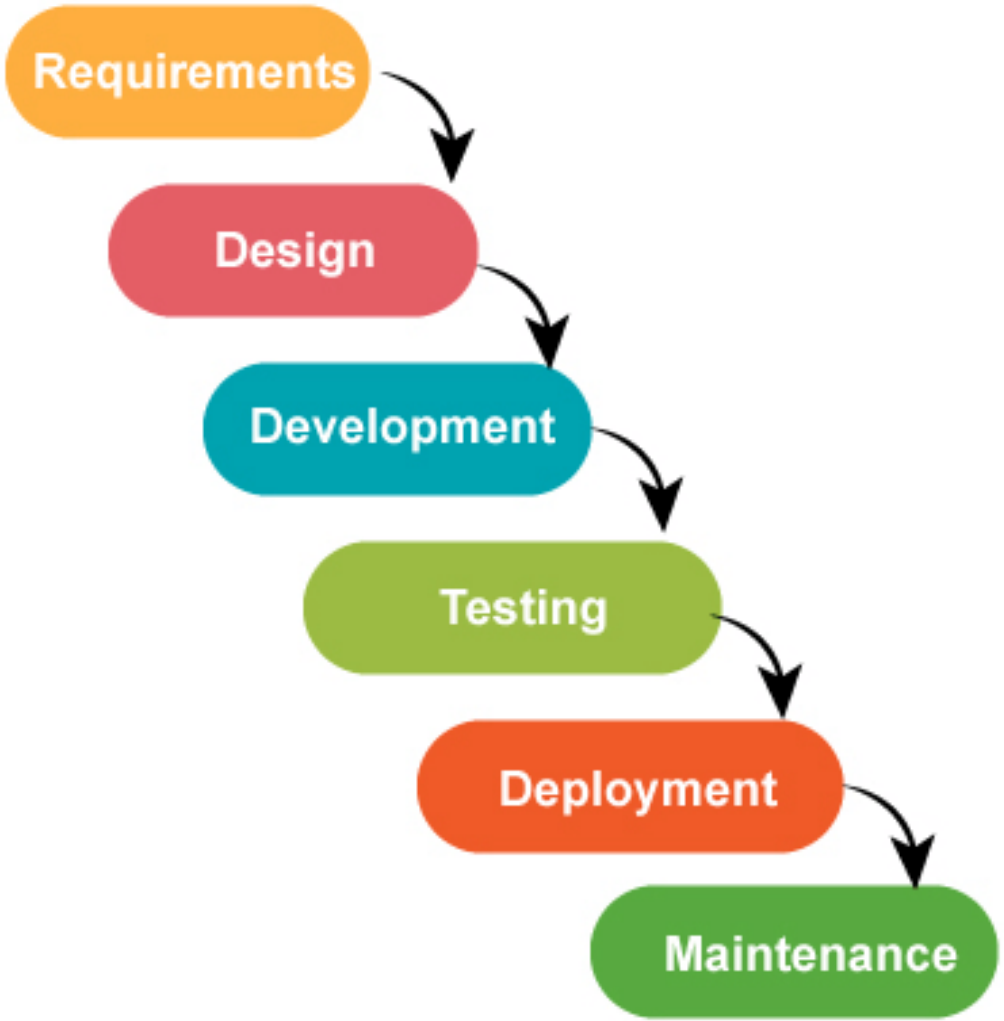
Waterfall Model

- **Requirements** – defines needed information, function, behavior, performance and interfaces.
- **Design** – data structures, software architecture, interface representations, algorithmic details.
- **Development** – source code, database, user documentation.



Waterfall Model

- **Testing** – verifies that the software meets requirements, identifies and fixes bugs through unit, integration, system, and acceptance testing.
- **Deployment** – delivers the software to the customer or end-users, involving installation, configuration, and initial use in the real environment.
- **Maintenance** – provides ongoing support after deployment, including corrective (bug fixes), adaptive (environment changes)



Waterfall Strengths

- Easy to understand, easy to use
- Provides structure to inexperienced staff
- Milestones are well understood
- Sets requirements stability
- Good for management control (plan, staff, track)
- Works well when quality is more important than cost or schedule

Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – inhibits flexibility
- Can give a false impression of progress
- Does not reflect problem-solving nature of software development – iterations of phases
- Integration is one big bang at the end
- Little opportunity for customer to preview the system (until it may be too late)

When to use the Waterfall Model?

- Requirements are very well known
- Product definition is stable
- Technology is understood
- New version of an existing product
- Porting an existing product to a new platform.

Quiz

V-Shaped SDLC Model

- A variant of the Waterfall that emphasizes **the verification** and **validation** of the product.

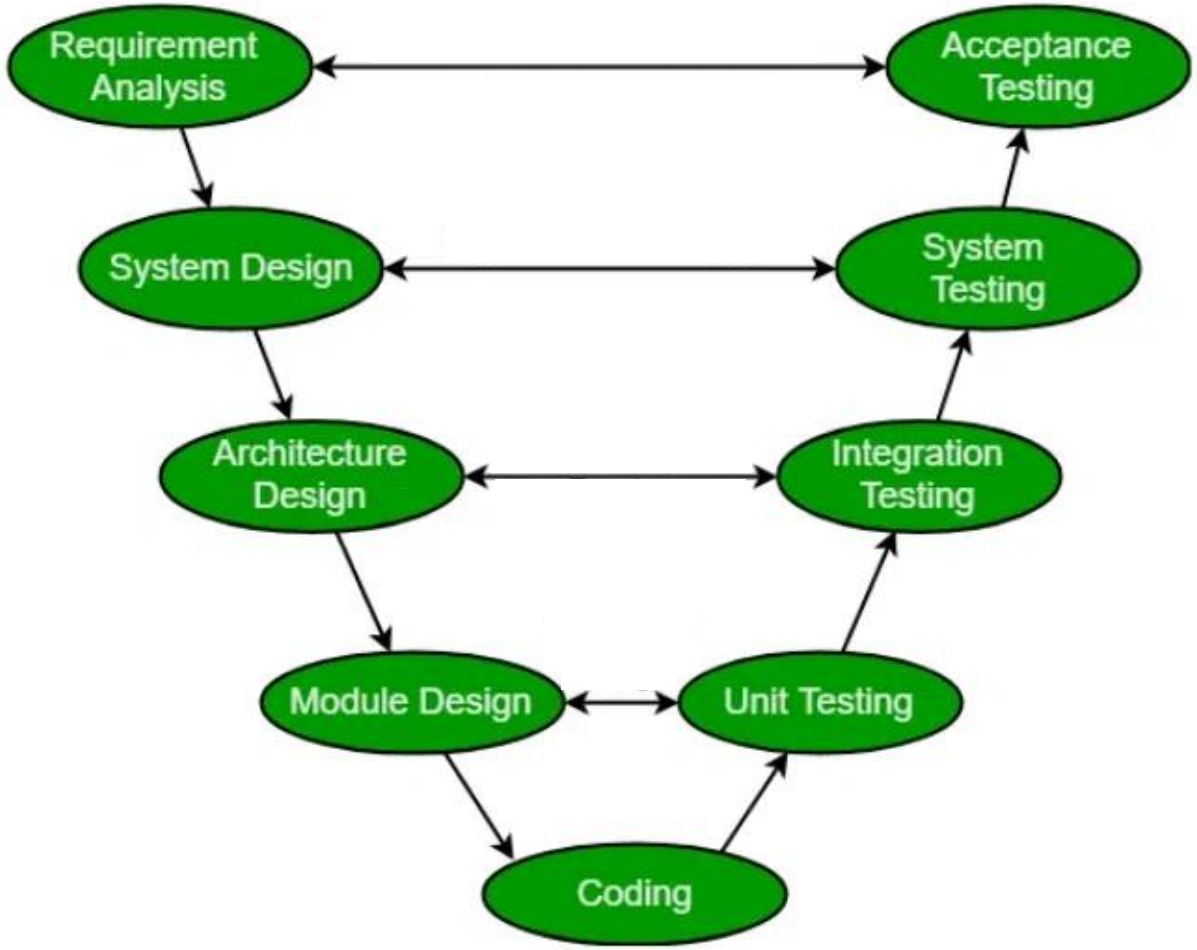
- **Testing** of the product is planned in **parallel** with a corresponding phase of development

Verification and Validation

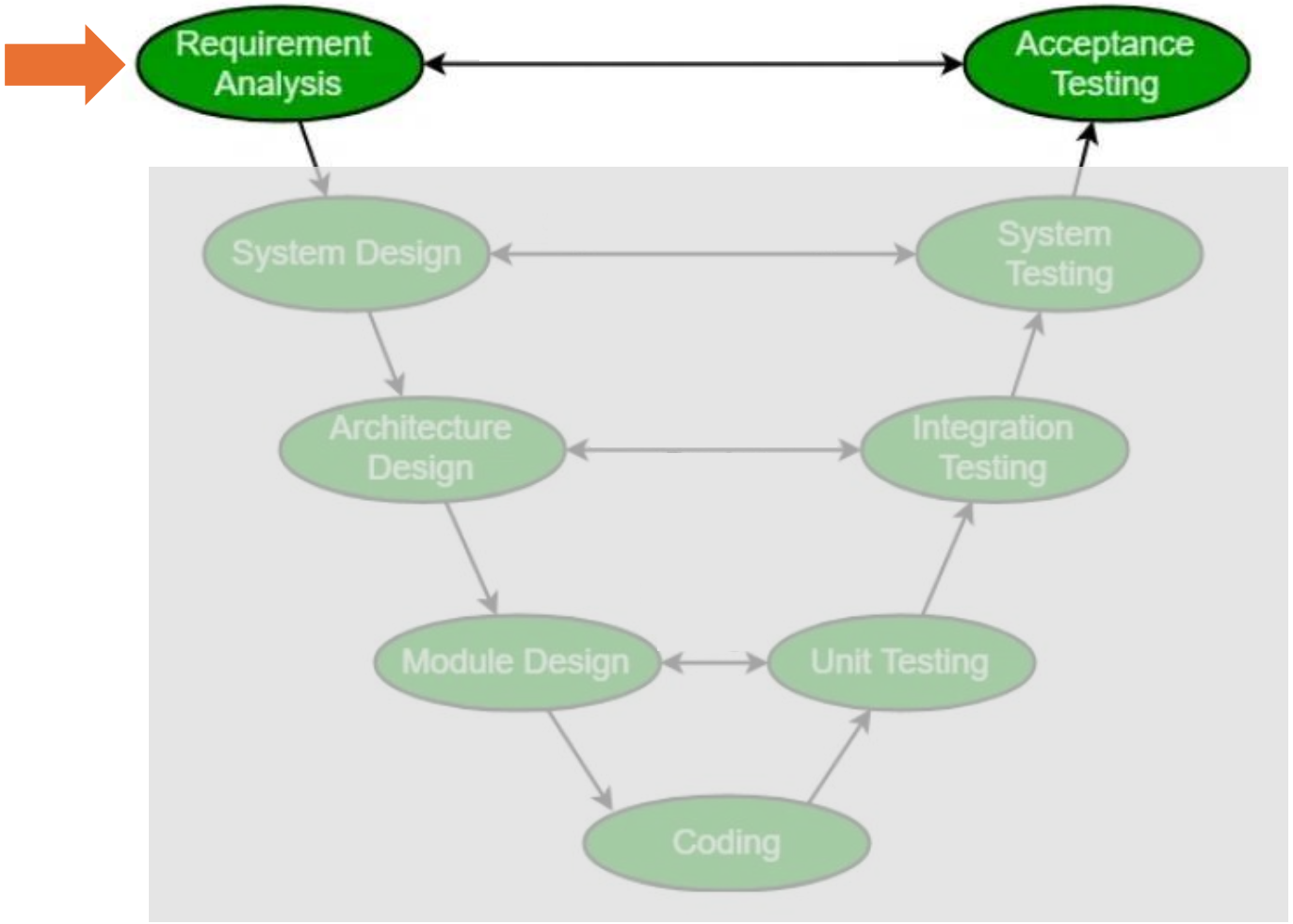
- **Verification:** Are we building the product right?
  Focus => Process, documents, design

- **Validation:** Are we building the right product?
  Focus => End product, user needs

V-Shaped Steps



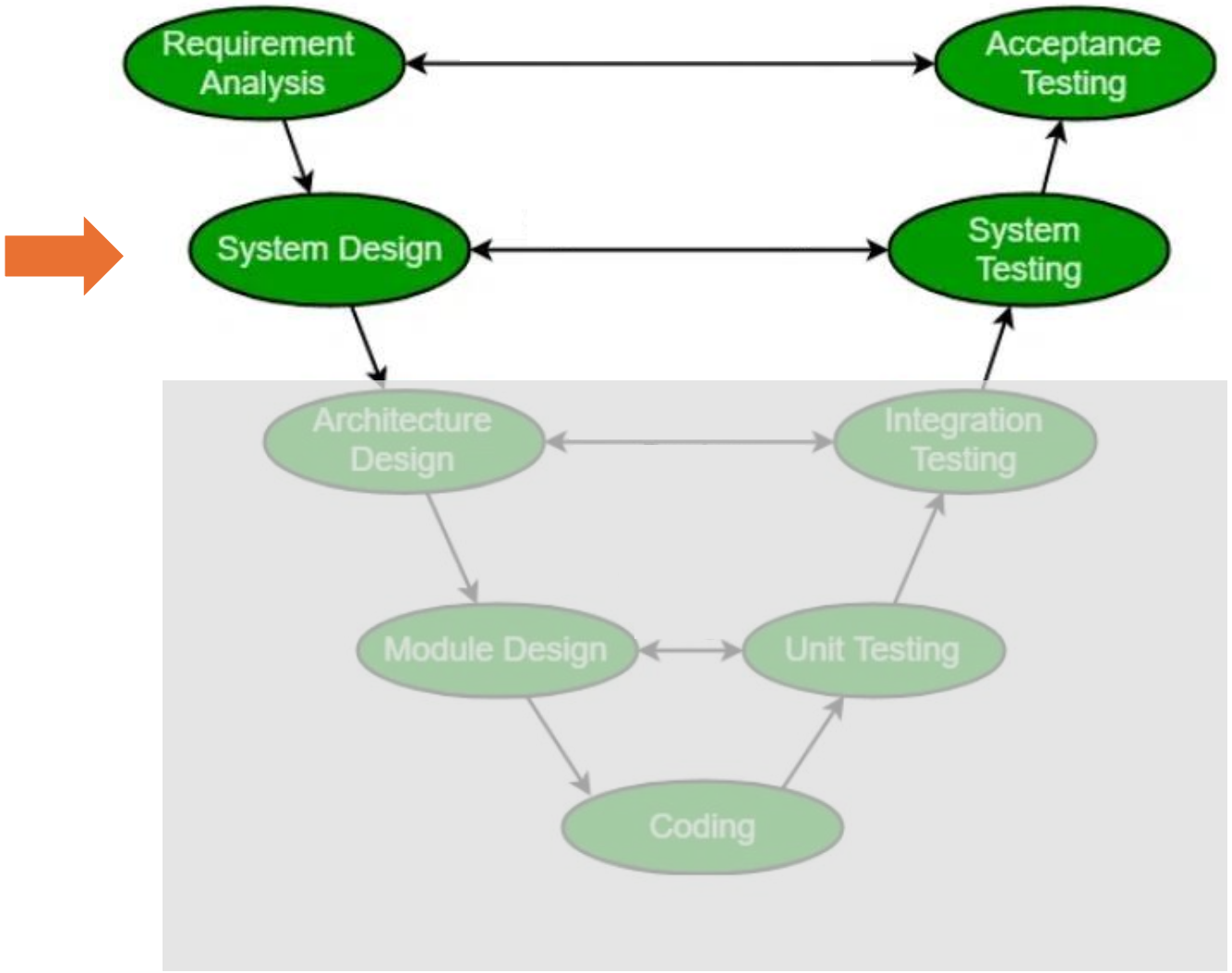
V-Shaped Steps - Requirements Analysis



Requirements Analysis : Gather all the user needs and document them clearly

Acceptance Testing : Check if the **final software meets the requirements specified**.

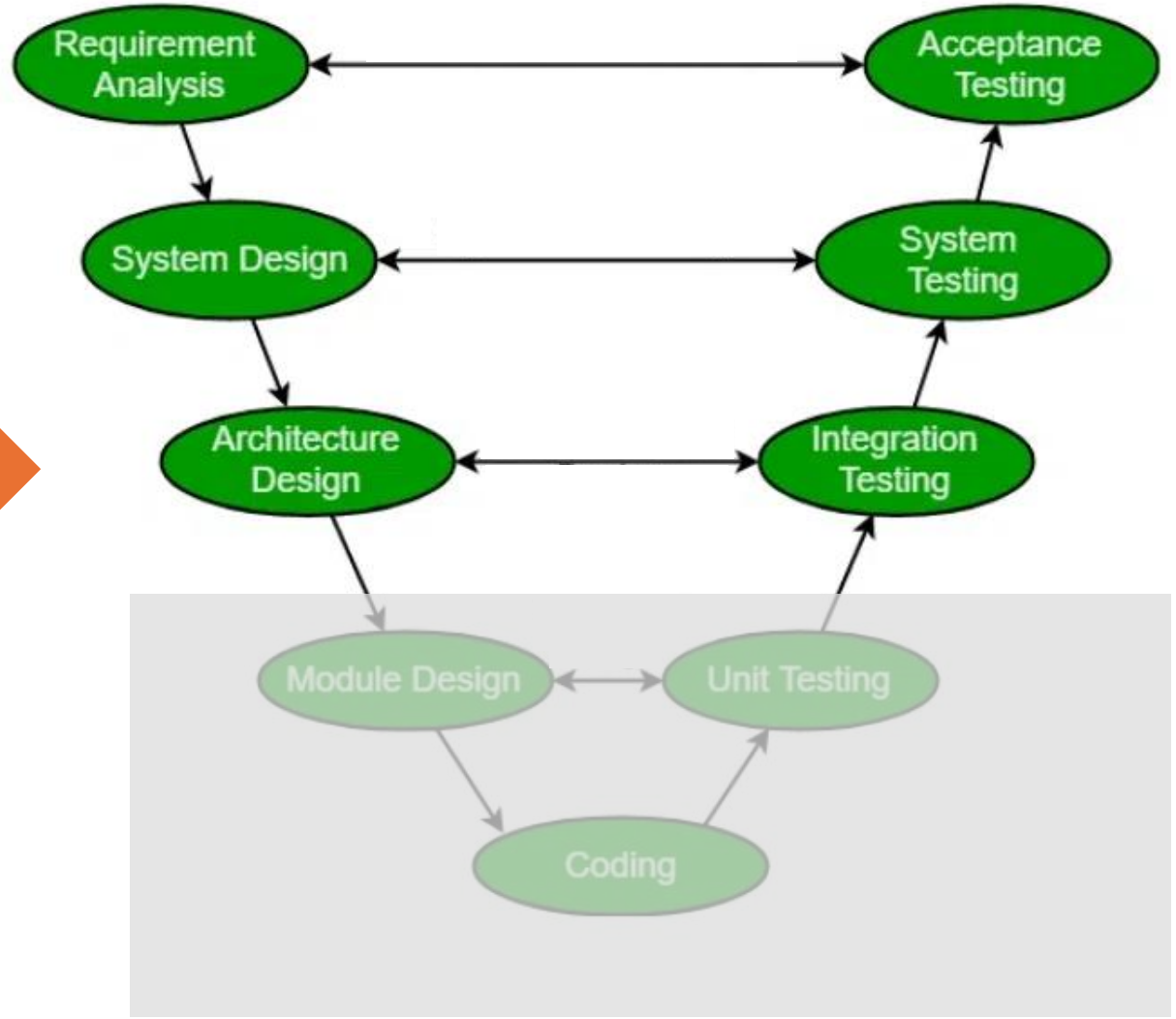
V-Shaped Steps - Requirements Analysis



System Design: Define overall system architecture.

System Testing: test the entire system as a whole against the requirements

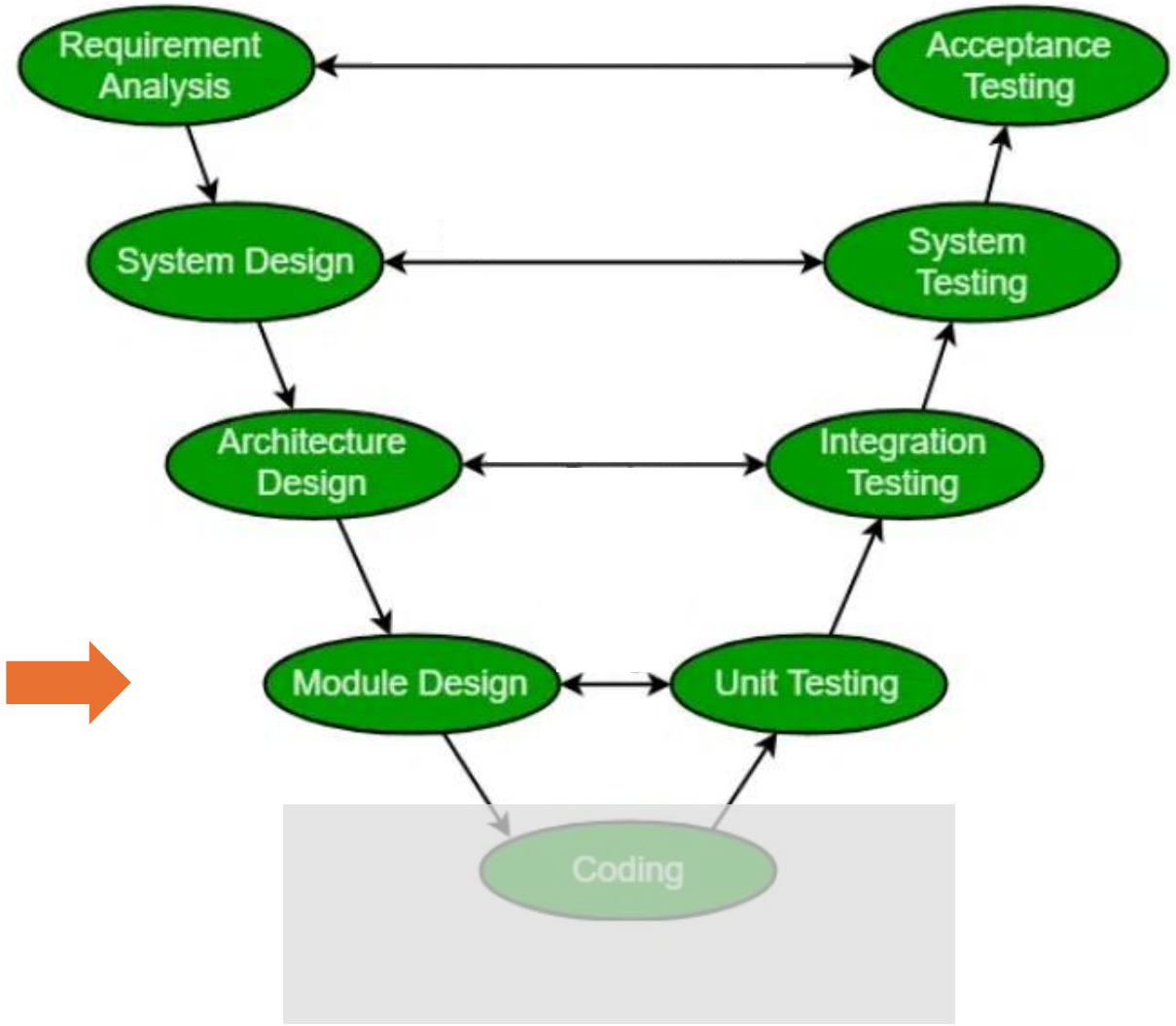
V-Shaped Steps - Requirements Analysis



Architecture or High-Level Design – defines how software functions fulfill the design

Integration and Testing check that modules interconnect correctly

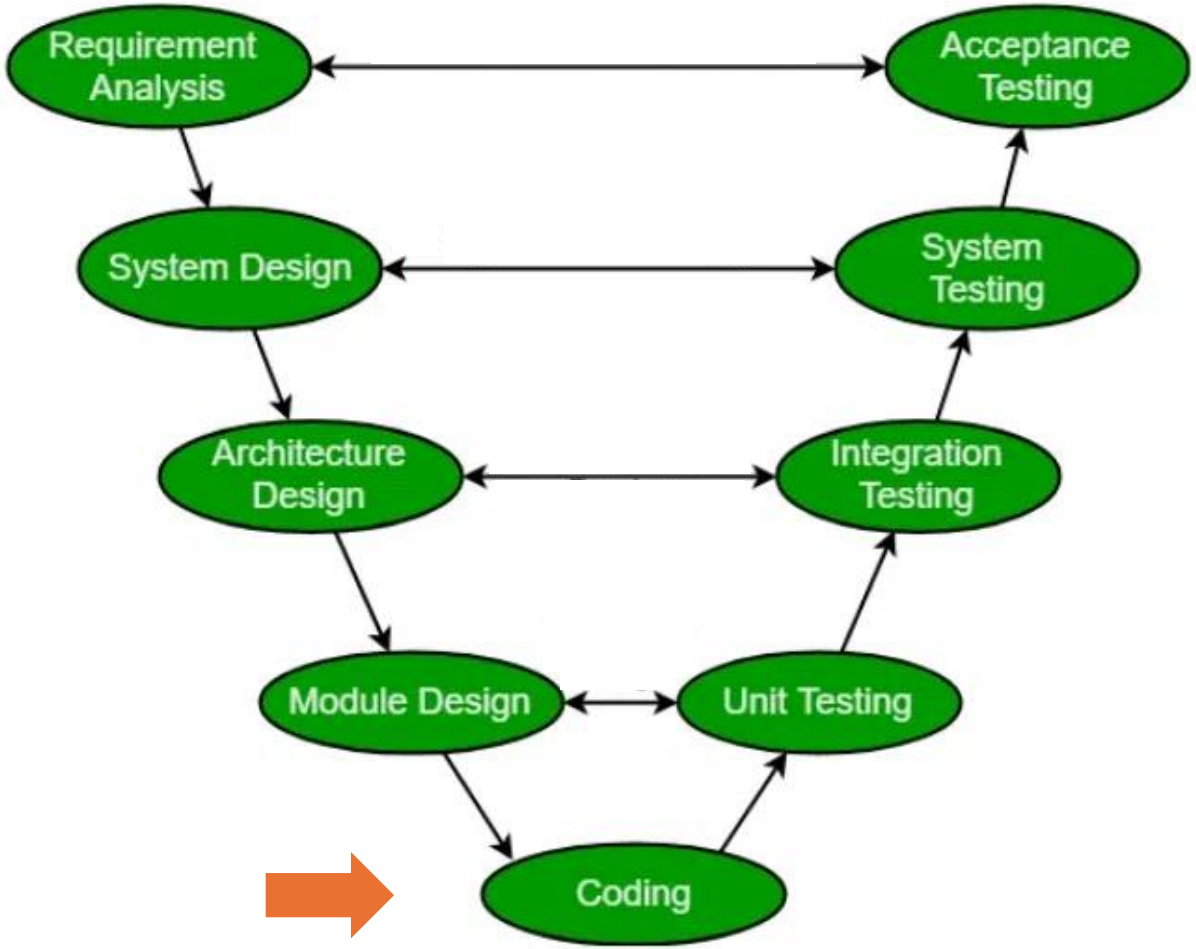
V-Shaped Steps - Requirements Analysis



Module Design – develop algorithms for each architectural component

Unit testing – check that each module acts as expected

V-Shaped Steps - Requirements Analysis



V-Shaped Strengths

- Emphasize planning for **verification and validation** of the product in early stages of product development
- **Each deliverable must be testable**
- Project management can **track progress by milestones**
- **Easy to use**

V-Shaped Weaknesses

- Does not handle **iterations** or phases
- Does not easily handle **dynamic changes in requirements**
- Does not contain **risk analysis** activities

When to use the V-Shaped Model

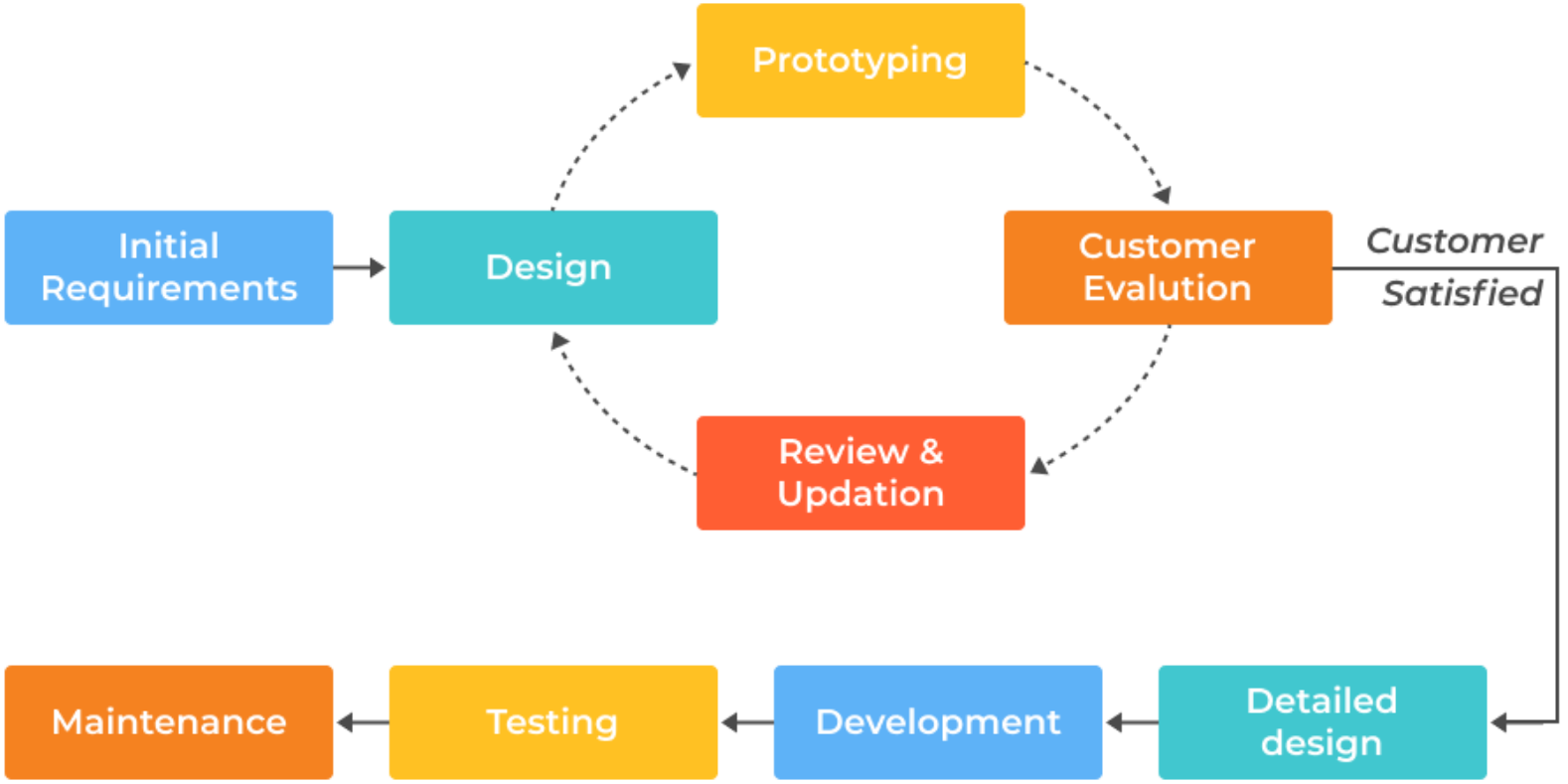
- Excellent choice for **systems requiring high quality**
- **All requirements are known** up-front
- **Solution and technology are known**

Quiz

Prototyping Model

- **Developers build a prototype** during the requirements phase
- Prototype is **evaluated by end users**
- Users give **corrective feedback**
- Developers further **refine the prototype**
- When the **user is satisfied**, the prototype code is brought up to the standards needed for a final product.

Prototyping Model



Prototyping Strengths

- Customers can **“see” the system requirements** as they are being gathered
- Developers **learn from customers**
- A more **accurate end product**
- **Unexpected** requirements accommodated
- Allows for **flexible design**
- Steady, **visible signs** of progress produced
- Interaction with the prototype stimulates awareness of **additional needed functionality**

Prototyping Weaknesses

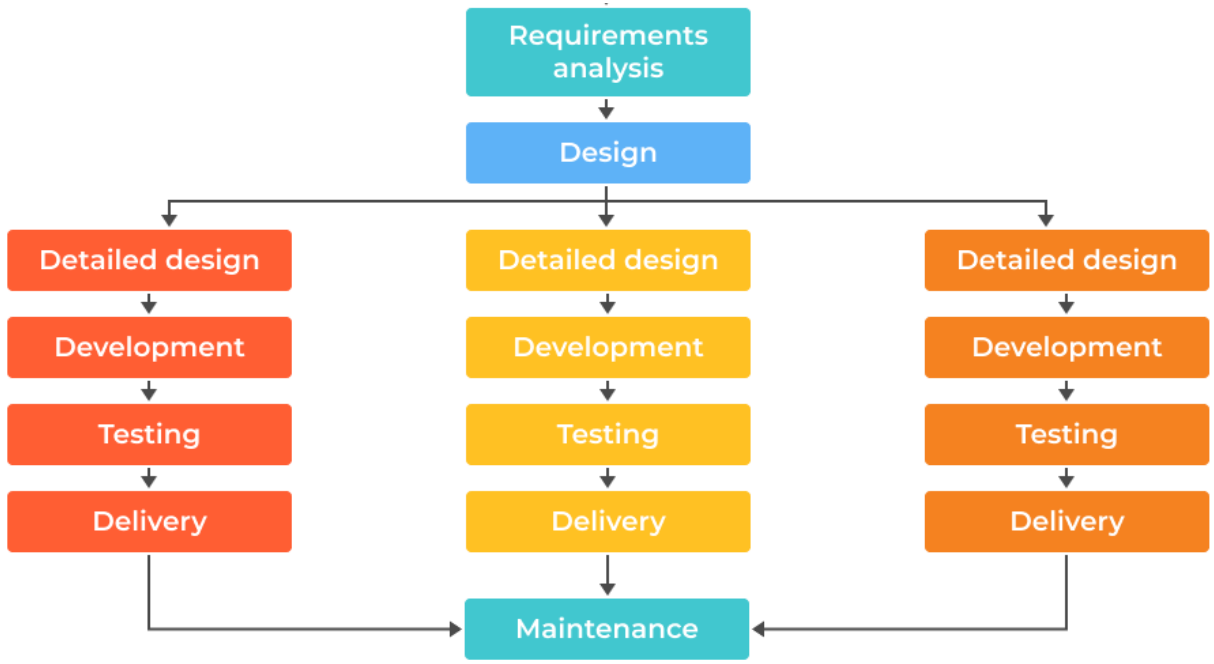
- Requires **constant client involvement**, which isn't always possible.
- **Slower** process, takes more time for developers.
- Frequent changes **can disturb the rhythm of the team.**
- **Poor documentation**, requirements keep changing.
- Can be **costly** in time and money.
- Prototyping tools may be **expensive**.

When to use Prototyping

- Requirements are unstable or have to be clarified
- For very small projects involving very few people
- New, original development

Quiz

Incremental SDLC Model



- Construct a partial implementation of a total system
- Then slowly add increased functionality
- The incremental model prioritizes requirements of the system.
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.

Incremental Model Strengths

- Develop high-risk or major functions first
- Each release delivers an operational product
- Customer can respond to each build
- Uses “divide and conquer” breakdown of tasks
- Lowers initial delivery cost
- Initial product delivery is faster
- Customers get important functionality early
- Risk of changing requirements is reduced

Incremental Model Weaknesses

- Requires good planning and design
- Requires early definition of the final system to allow the definition of increments
- Can be **costly**

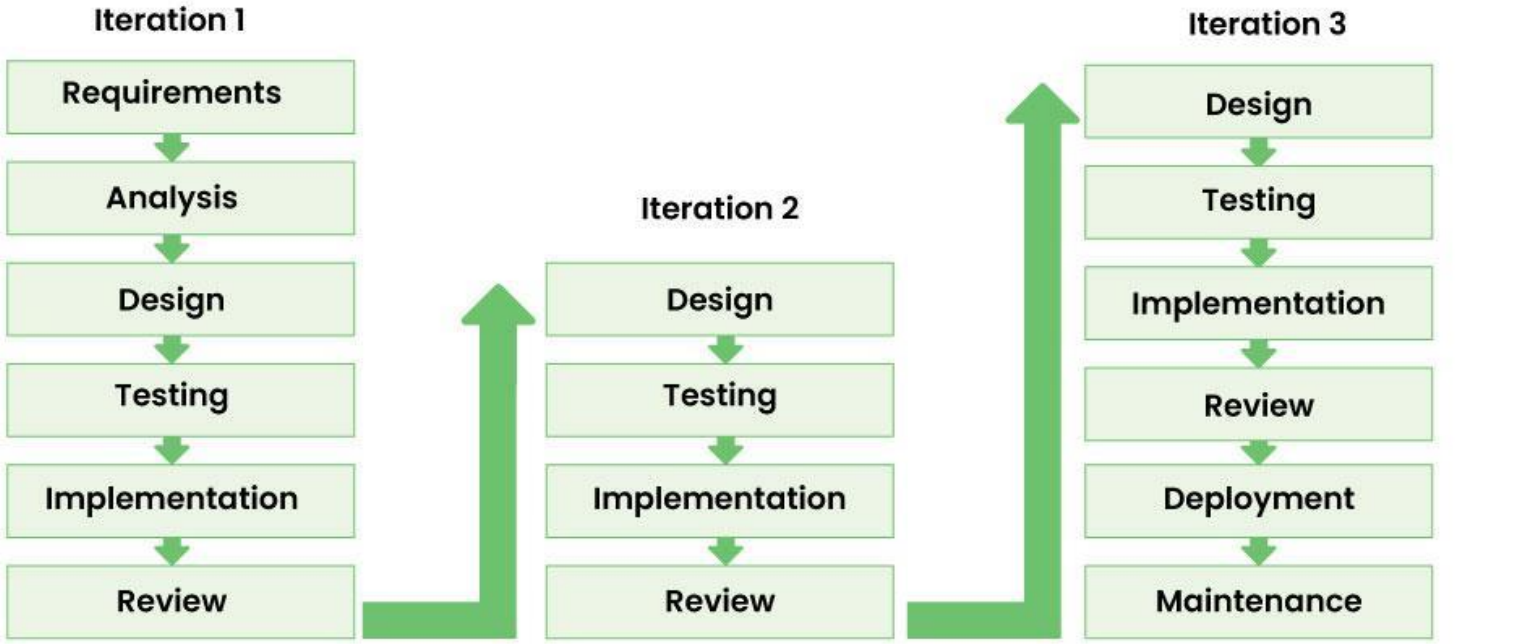
When to use the Incremental Model

- When most specifications are known in advance and will only undergo minor changes
- When a functional product is needed **quickly**
- For **long-duration** projects
- For projects involving new technologies

Quiz

Iterative model

- Software is developed in **small cycles (iterations)**.
- The product improves with **feedback and refinement**.
- Focus: **evolution over time**, not delivering everything at once.



Iterative model

Iterative Model Strengths

- Early delivery of a **working version**.
- **Customer feedback** is included regularly.
- Easier to **adapt to changing requirements**.
- Errors can be detected **early**.
- Reduces overall **risk of failure**.

Iterative Model Weaknesses

- Needs **good planning and design** upfront.
- Frequent changes can make it **costly**.
- Requires **constant involvement** of the client.
- Hard to estimate **final cost and timeline**.
- Not ideal for **small teams with limited resources**.

When to use the Iterative Model

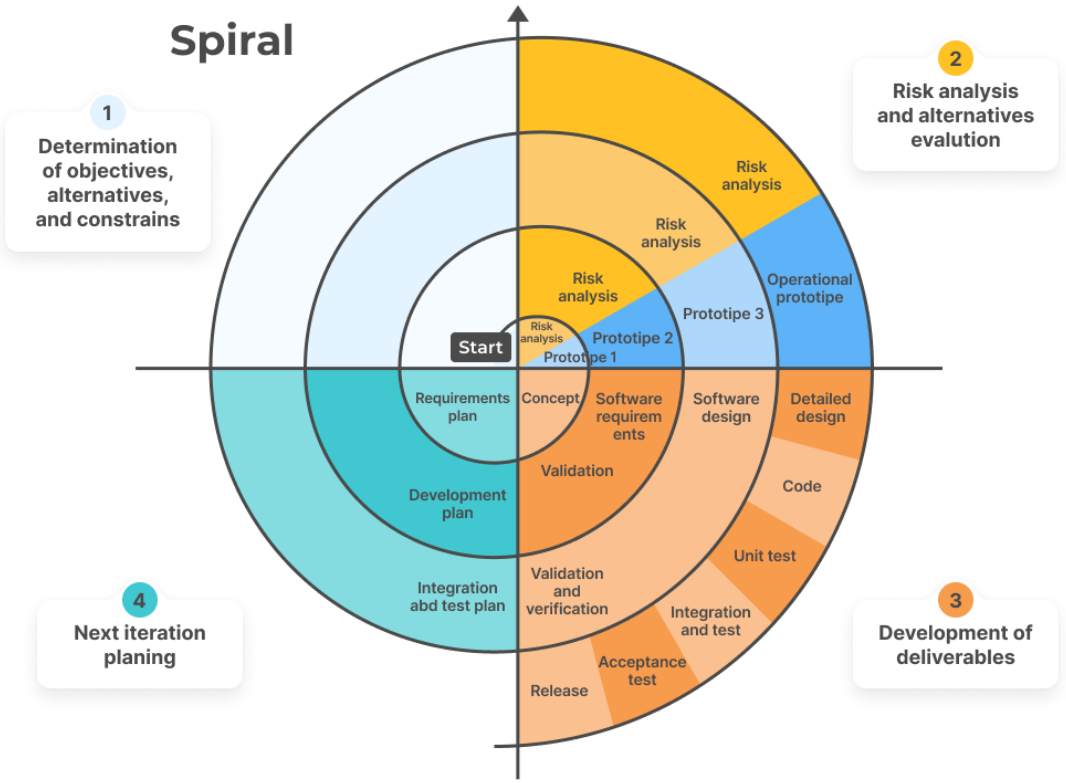
- Projects with **unclear or evolving requirements**.
- When **early feedback** is critical.
- For **medium to large projects**.
- When **new technology** is being explored.

Quiz

Spiral SDLC Model

- Developed by **Barry Boehm in 1986.**
- It’s a risk-driven software development process.
- Combination of Waterfall, Iterative, and Prototyping models.
- Software is developed in a series of incremental releases as per each spiral.
- Example: Gaming industry.

Spiral SDLC Model



Spiral model divided into the four part:

1. Determination of Objectives
2. Risk Analysis and Evaluation
3. Development
4. Next Iteration Planning

Each cycle involves the same sequence of steps as the waterfall process model

Spiral Quadrant
1- Determine objectives, alternatives and constraints

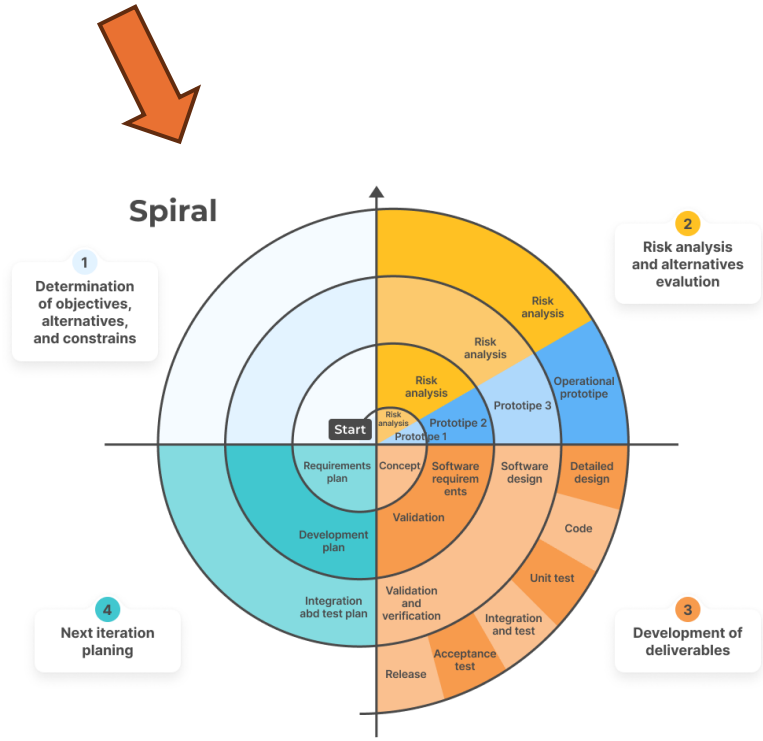
Purpose: Define objectives, alternatives, and constraints for the iteration.

Activities:

Identify requirements for this phase.
Define goals of the increment.
Decide deliverables, schedules, and resources.

Output:

Requirement Specification (for this cycle).
Planning Document.



Spiral Quadrant
2- Risk Analysis

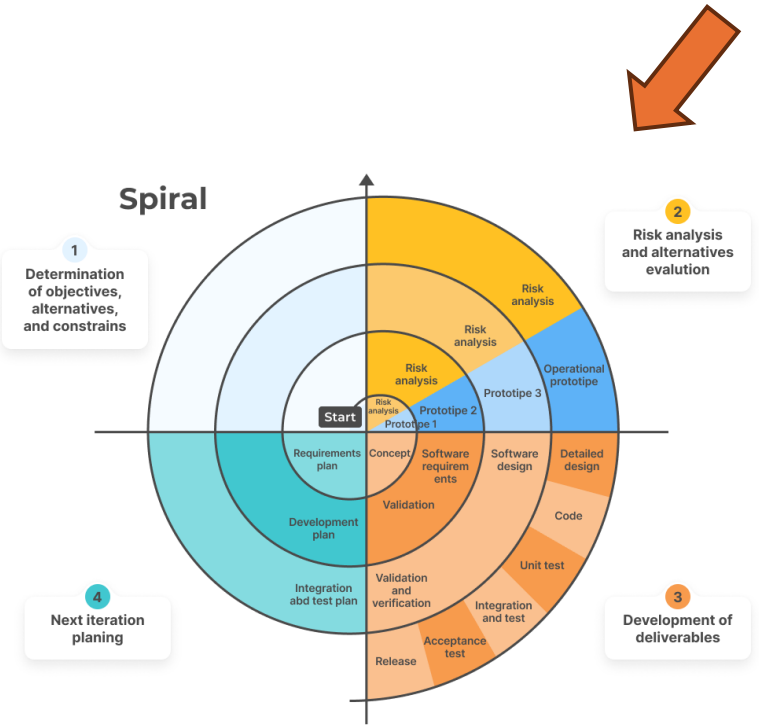
Purpose: Identify, analyze, and reduce risks.

Activities:

Identify possible risks (technical, cost, schedule, usability).
Explore alternatives to mitigate risks.
Build prototypes if needed to reduce uncertainty.

Output:

Risk Analysis Report.
Prototype (if used to mitigate risk).



Spiral Quadrant
3- Development

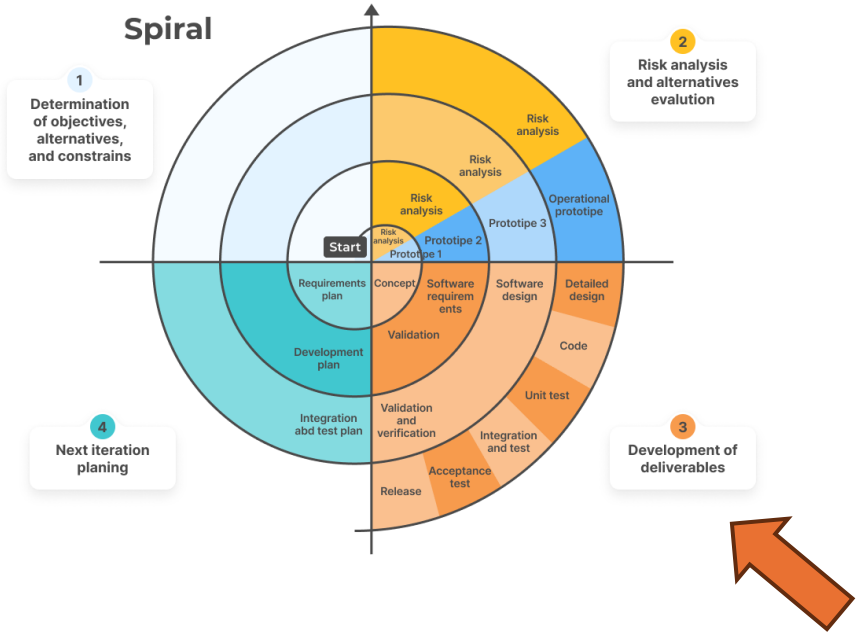
Purpose: Develop and test the increment.

Activities:

Detailed design of modules.
Coding and implementation.
Unit and integration testing.

Output:

software increment.



Spiral Quadrant
4- Plan next phase

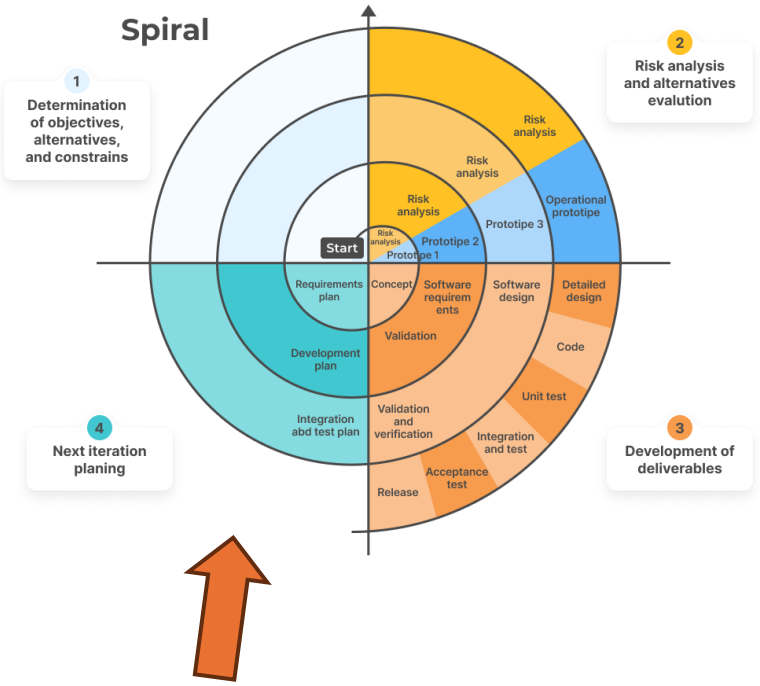
Purpose: Get feedback and validate progress with the customer.

Activities:

Customer reviews increment.
Collect feedback on usability, performance, functionality.
Approve or suggest changes for next cycle.

Output:

User Feedback.
Updated requirements for next loop.



Spiral Model Strengths

- Provides early indication of insurmountable risks, without much cost
- Users see the system early because of rapid prototyping tools
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Early and frequent feedback from users

Spiral Model Weaknesses

- Time spent for evaluating risks too large for small or low-risk projects
- Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive
- The model is complex
- Risk assessment expertise is required
- Spiral may continue indefinitely
- Developers must be reassigned during non-development phase activities
- May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration

When to use Spiral Model

- When creation of a prototype is needed
- When costs and risk evaluation is important
- For medium to high-risk projects
- Users are unsure of their needs
- Requirements are complex
- When the project involves research and investigation

The spiral model – Top risk items

Table 4. A prioritized top-ten list of software risk items.

Risk item	Risk management techniques
1. Personnel shortfalls	Staffing with top talent, job matching; teambuilding; morale building; cross-training; pre-scheduling key people
2. Unrealistic schedules and budgets	Detailed, multisource cost and schedule estimation; design to cost; incremental development; software reuse; requirements scrubbing
3. Developing the wrong software functions	Organization analysis; mission analysis; ops-concept formulation; user surveys; prototyping; early users' manuals
4. Developing the wrong user interface	Task analysis; prototyping; scenarios; user characterization (functionality, style, workload)
5. Gold plating	Requirements scrubbing; prototyping; cost-benefit analysis; design to cost
6. Continuing stream of requirement changes	High change threshold; information hiding; incremental development (defer changes to later increments)
7. Shortfalls in externally furnished components	Benchmarking; inspections; reference checking; compatibility analysis
8. Shortfalls in externally performed tasks	Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; teambuilding
9. Real-time performance shortfalls	Simulation; benchmarking; modeling; prototyping; instrumentation; tuning
10. Straining computer-science capabilities	Technical analysis; cost-benefit analysis; prototyping; reference checking

Barry W. Boehm, A Spiral Model of Software Development and Enhancement, IEEE Computer 21(5), pp. 61-72, 1988.

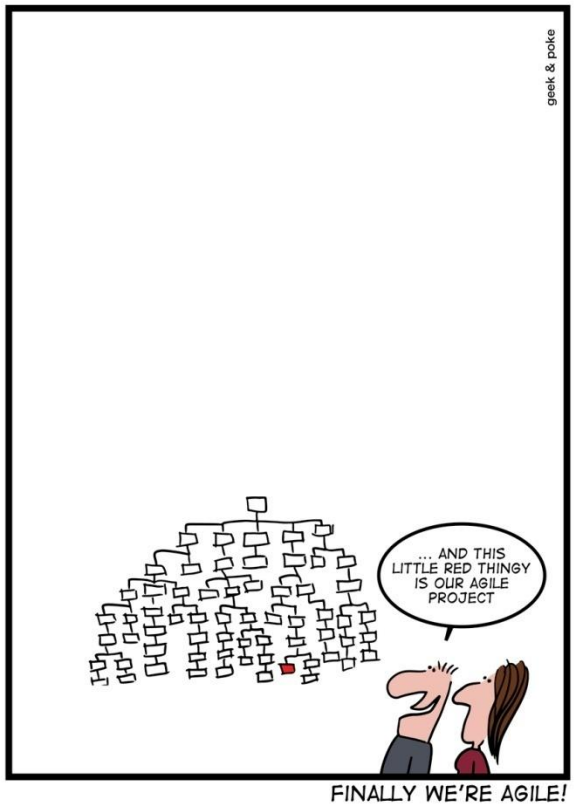
Quiz

Traditional Software Development Methodologies

Discussion

Agile Software Development Methodologies

Agile methods



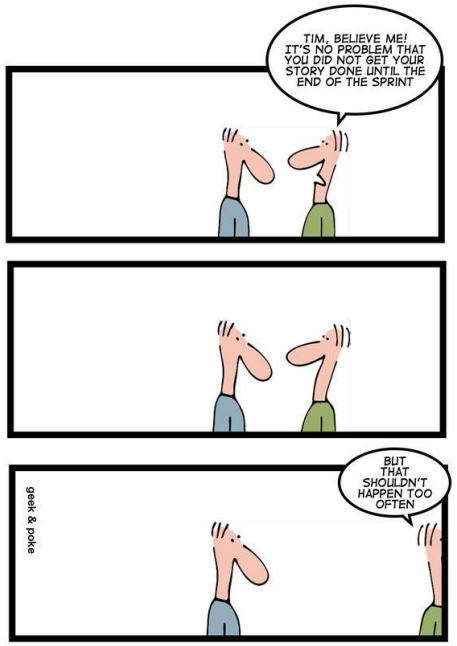
Dissatisfaction with the overheads involved in conventional processes led to the creation of **agile methods**.

The aim of agile methods is to **reduce overheads** in the software process (e.g. by limiting documentation) and to be able to **respond quickly to changing requirements** without excessive rework.

Agile – Historical Background

- In the **mid-1990s**, project management experts wanted **new ways** to handle software development.
- These new approaches were **“lighter”** → less documentation, less bureaucracy.
- Targeted **small to medium projects** with **smaller teams**.
- Focus on **quick adaptation** to changes in requirements.
- Ensured **frequent deliveries** instead of waiting for the final product.
- These approaches became known as **“Agile Methods.”**

Agile manifesto



AGILE MANIFESTO

We have come to value:

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

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

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

Principle 1 – Individuals & Interactions over Processes & Tools

- **People are the key to success**.
- Even senior experts will fail if they don’t **work as a team**.
- A good team player is more valuable than a brilliant but isolated programmer.
- **Too many tools** can be just as harmful as not having enough.
- Start small, invest little at the beginning.
- **Building the team** is more important than building the environment.

Principle 2 – Working Software over Comprehensive Documentation

- Code **without any documentation** is a disaster.
- But **too much documentation** can be worse than none.
- Hard to produce and keep in sync with the code.
- Documents often become **“formal lies”** (outdated or inaccurate).
- **The code never lies**, it shows the real state of the system.
- Always produce documentation that is **short, useful, and up-to-date**

Principle 3 – Customer Collaboration over Contract Negotiation

- Impossible to describe the **entire software** from the start.
- Successful projects involve the client **frequently and regularly**.
- The client should have **direct contact** with the development team

Principle 4 – Responding to Change over Following a Plan

- Software cannot be **planned too far ahead**.
- Everything changes: technology, environment, and especially **user needs**.
- Traditional project managers rely on **task lists**.
- Over time, these plans **fall apart** as new tasks appear and old ones become obsolete.
- Agile prefers **short-term planning** (2 weeks to 1 month).
- Plans should be **detailed for the next week**, **solid for the next 3 months**, and **flexible beyond that**.

Agile SDLC’s

- Speed up or bypass one or more life cycle phases
- Usually less formal and reduced scope
- Used for time-critical applications

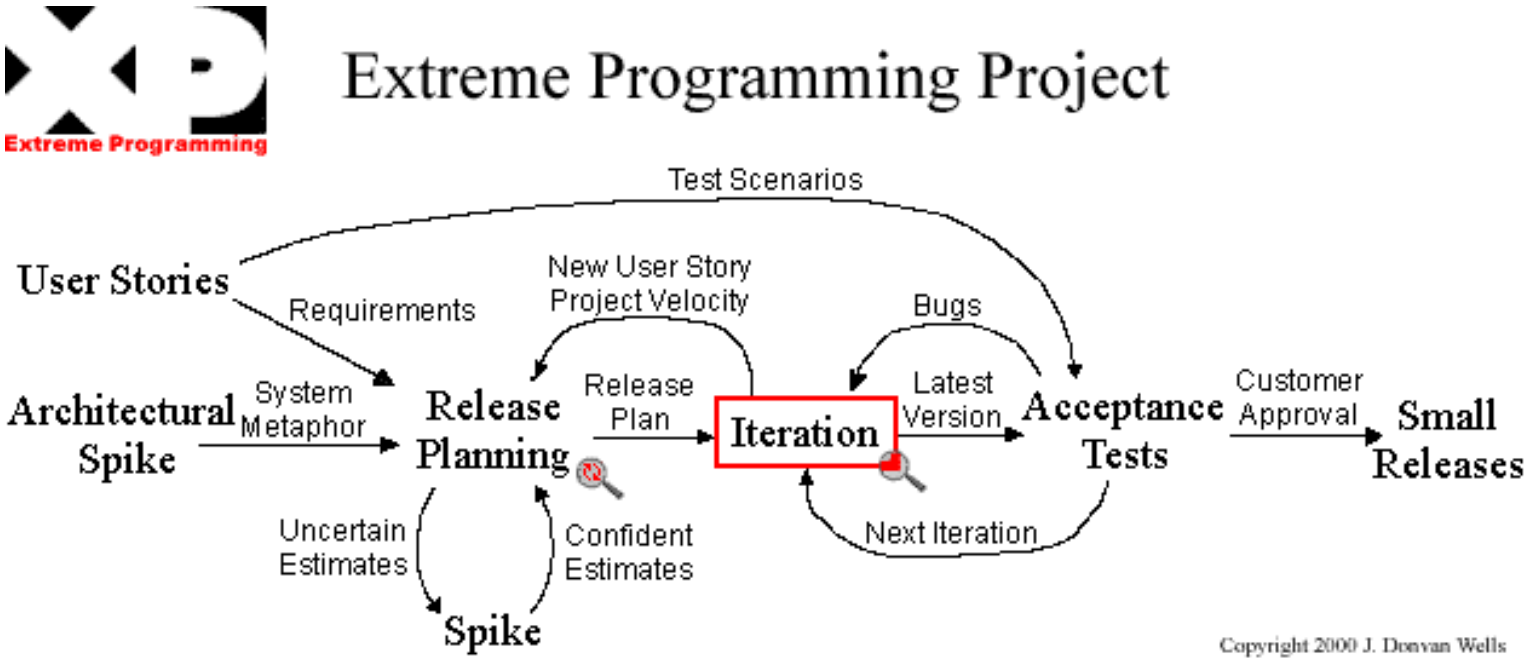
Some Agile Methods

- Adaptive Software Development (ASD)
- Feature Driven Development (FDD)
- Crystal Clear
- Dynamic Software Development Method (DSDM)
- Rapid Application Development (RAD)
- Scrum
- Extreme Programming (XP)
- Rational Unify Process (RUP)

Extreme Programming - XP

- Created in 1995 by Kent Beck and Ward Cunningham
- XP is a lightweight, efficient, low-risk, flexible, scientific, and enjoyable approach to software development
- Designed for medium-sized teams with incomplete and/or vague specifications
- Coding is the **core** of XP

XP model


Extreme Programming
Extreme Programming Project
Test Scenarios
User Stories
Requirements
New User Story
Project Velocity
Bugs
Architectural
Spike
System
Metaphor
Release
Planning
Release
Plan
Iteration
Latest
Version
Acceptance
Tests
Customer
Approval
Small
Releases
Uncertain
Estimates
Confident
Estimates
Spike
Next Iteration
Copyright 2000 J. Donvan Wells

XP Practices (1-6)

1. Planning game – determine scope of the next release by combining business priorities and technical estimates
2. Small releases – put a simple system into production, then release new versions in very short cycle
3. Metaphor – all development is guided by a simple shared story of how the whole system works
4. Simple design – system is designed as simply as possible (extra complexity removed as soon as found)
5. Testing – programmers continuously write unit tests; customers write tests for features
6. Refactoring – programmers continuously restructure the system without changing its behavior to remove duplication and simplify

XP Practices (7 – 12)

7. Pair-programming -- all production code is written with two programmers at one machine
8. Collective ownership – anyone can change any code anywhere in the system at any time.
9. Continuous integration – integrate and build the system many times a day – every time a task is completed.
10. 40-hour week – work no more than 40 hours a week as a rule
11. On-site customer – a user is on the team and available full-time to answer questions
12. Coding standards – programmers write all code in accordance with rules emphasizing communication through the code

XP is “extreme” because

Commonsense practices taken to extreme levels

- If code reviews are good, review code all the time (pair programming)
- If simplicity is good, keep the system in the simplest design that supports its current functionality. (simplest thing that works)
- If integration testing is important, build and integrate test several times a day (continuous integration)
- If short iterations are good, make iterations really, really short (days rather than weeks)

XP Methodology Disadvantages

- Requires a certain level of developer maturity
- Pair programming may not always be feasible
- Difficult to plan and budget a project
- Stress due to continuous integration and frequent releases
- Limited documentation can be problematic if developers leave

Scrum

Scrum is a framework that has been used to manage work on complex products since the early 1990s.

Scrum is not a process, technique, or definitive method. Rather, it is a framework within which you can employ various processes and techniques.

Ken Schwaber and Jeff Sutherland

Scrum Principles

- **Simple**
  - Can be combined with other methods
  - Compatible with best practices
- **Empirical**
  - Short iterations (*sprints*)
  - Continuous feedback
- **Simple Techniques**
  - Sprints of 2 to 4 weeks
  - Requirements captured as *user stories*
- **Teams**
  - Self-organizing
  - Cross-functional
- **Optimization**
  - Rapid detection of issues
  - Simple organization
  - Requires openness and visibility

Scrum Team

The Product Owner is the person responsible for managing the Product Backlog. Product Backlog management includes:

1. Clearly expressing Product Backlog items;
2. Ordering the items in the Product Backlog to best achieve goals and missions;
3. Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next;
4. Ensuring the Development Team understands items in the Product Backlog to the level needed.

Scrum Team

The Development Team consists of professionals who do the work of delivering a potentially releasable Increment of “Done” product at the end of each Sprint.

A “Done” increment is required at the Sprint Review. Only members of the Development Team create the Increment.

<= 3 Size of team < 9

Scrum Team

The Scrum Master is responsible for promoting and supporting Scrum as defined in the Scrum Guide.

Scrum Masters do this by helping everyone understand Scrum theory, practices, rules, and values.

Scrum Events

The Sprint

The heart of Scrum is a Sprint, a time-box of one month or less during which a “Done”, useable, and potentially releasable product Increment is created. Sprints have consistent durations throughout a development effort.

A new Sprint starts immediately after the conclusion of the previous Sprint.

Sprints contain and consist of the Sprint Planning, Daily Scrums, the development work, the Sprint Review, and the Sprint Retrospective.

During the Sprint: **No changes** are made that would endanger the Sprint Goal; Quality goals do not decrease; and, Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned.

Scrum Events

Sprint Planning

Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.

Sprint Planning answers the following:

1. What can be delivered in the Increment resulting from the upcoming Sprint?
2. How will the work needed to deliver the Increment be achieved?

What => select from Program Backlog

How => designing the product

Scrum Events

Daily Scrum

The Daily Scrum is a 15-minute time-boxed event for the Development Team. The Daily Scrum is held every day of the Sprint.

Here is an example of what might be used:

What did I do yesterday that helped the Development Team meet the Sprint Goal?

What will I do today to help the Development Team meet the Sprint Goal?

Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

Scrum Events

Sprint Review

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed.

During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done to optimize value.

This is an informal meeting, not a status meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration. This is at most a four-hour meeting for one-month Sprints. For shorter Sprints, the event is usually shorter

Scrum Events

Sprint Retrospective

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

1. Inspect how the last Sprint went with regards to people, relationships, process, and tools;
2. Identify and order the major items that went well and potential improvements;
3. Create a plan for implementing improvements to the way the Scrum Team does its work.

Scrum Artefacts

Product Backlog

The Product Backlog is an ordered list of everything that is known to be needed in the product.

The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items often include test descriptions

Scrum Artefacts

Sprint Backlog

The Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal.

The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality into a “Done” Increment.

As work is performed or completed, the estimated remaining work is updated.

Scrum Model Strengths

- Very simple and highly effective
- Adopted by industry giants:
  - Microsoft, Google, Nokia…
- Project-oriented
  - Unlike XP, which is development-oriented
- Flexible
  - Can include practices from other methodologies (*especially XP*)

Scrum Model Weaknesses

Not 100% specific to Software Engineering

- Scrum is a general project management framework.
- It doesn’t cover all technical aspects of software development.

Difficulty in budgeting a project

- Because Scrum uses iterations and evolving requirements,
- It’s harder to predict exact cost and timeline upfront.