Announcements

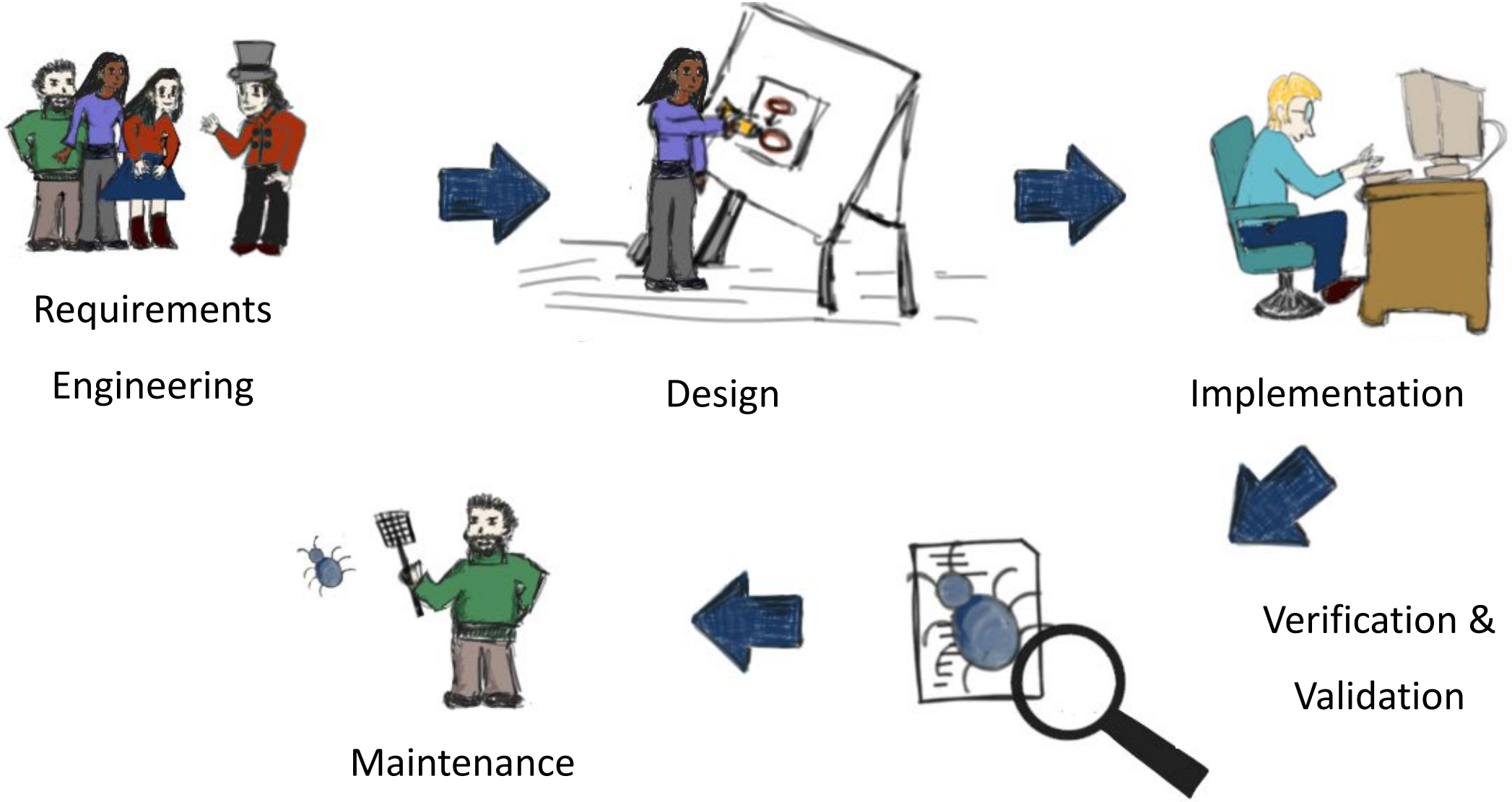
- Groups formed. Mentors assigned
- Next class will be with mentors. Decide on your project topic, 6 MMFs and tech stack; submit the assignment (due same day) and get feedback/graded from mentors in class.
- Extra Credit Opportunity in today's class
- Maven Assignment based on todays lecture Out. Due 5/25
- Project 1 Planning assignment due 5/28.

CS3300 A Introduction to Software Engineering

Lecture 03: SDLC; Life Cycle Models; TOTT #2 – IDE, Junit Testing, Maven

Dr. Nimisha Roy ▸ nroy9@gatech.edu

Traditional Software Development Phases



Software Development Phases: Semester Assignments

Requirements Engineering

Project MMF, Project Planning; Project RE

Design

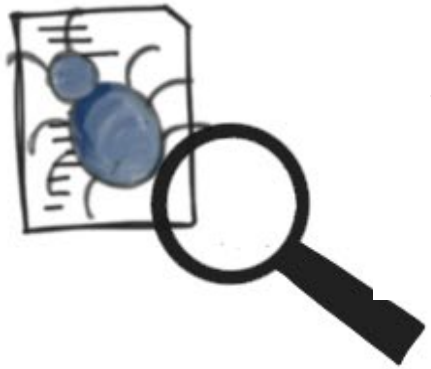
Project Design

Implementation

Project code, report, ppt

Verification & Validation

Project Test



Maintenance

Software Development Life Cycle

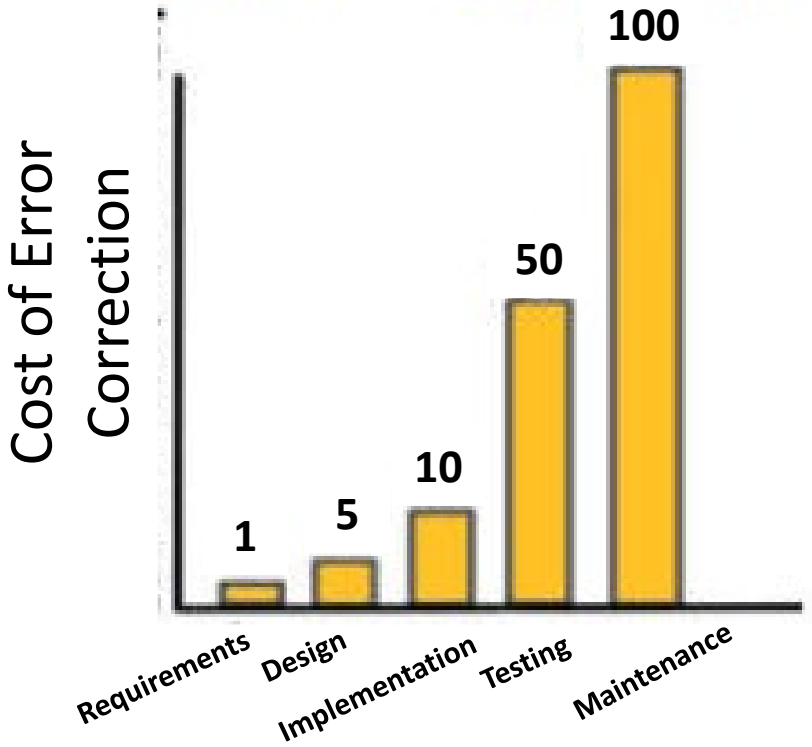
Stage 1 Planning & Requirement Analysis	Stage 2 Defining Requirements	Stage 3 Design	Stage 4 Development	Stage 5 Testing	Stage 6 Deployment & Maintenace
Planning	Defining	Design	Development	System Testing	Deployment and Maintenace
Define Project Scope	Functional Requirement	HLD	Coding Standard	Manual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

6 Stages of Software Development Life Cycle

Requirements Engineering

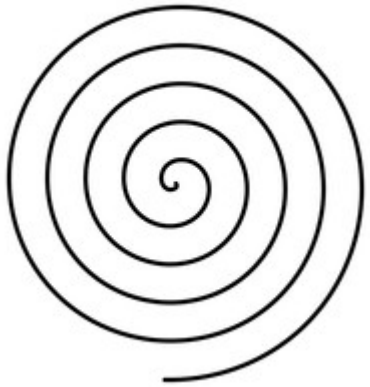
RE is the process of establishing the needs of stakeholders that are to be solved by software

Cost of Late Correction



Elicitation

Management



Analysis

Validation

Specification

RE Example: Task Management Software

Elicitation: Conduct interviews with potential users (project managers, team members) to gather what features they need in a task management tool. For example, task assignment, deadlines, notifications, and progress tracking might be common requirements.

Analysis: Analyze the feedback to determine essential features versus nice-to-have features. This might involve grouping similar requirements and identifying conflicts or unrealistic expectations.

Specification: Create a detailed SRS document that clearly lists all functional and non-functional requirements, such as user roles, security levels, and user interface preferences.

Validation: Review the SRS with stakeholders to ensure the requirements are complete, consistent, realistic, and verifiable. Make Modifications based on feedback to better align with user expectations and technical feasibility.

Management: Establish a process to accommodate changes in requirements throughout the project lifecycle, ensuring traceability and controlled integration of changes.

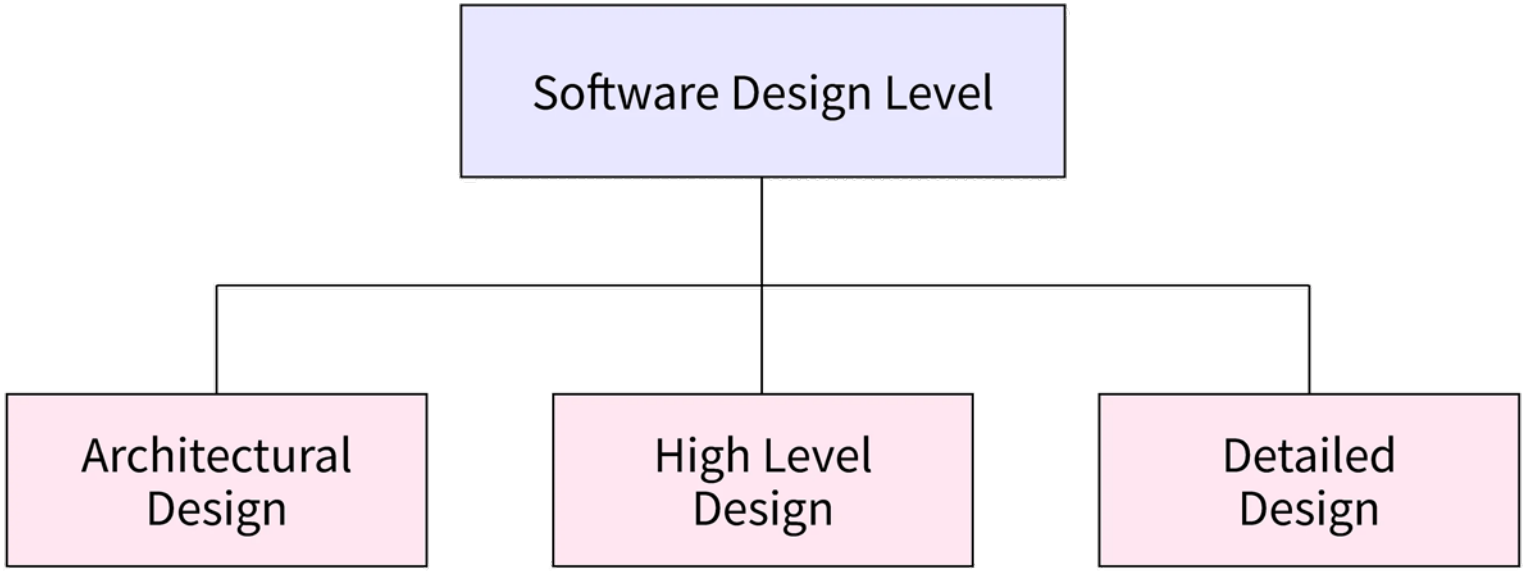
Design

SRS (Software Requirements Specification) is a reference for software designers to come up with the best design for the software.

Multiple designs for the product architecture are present in the Design Document Specification (DDS).

This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for development.

Design



The **architectural design** characterizes the software as a system with numerous interconnected components. The designers acquire an overview of the proposed solution domain at this level.

The high-level design deconstructs the architectural design's 'single entity-multiple component' notion into a less abstract perspective of subsystems and modules, depicting their interaction with one another

Each module is extensively investigated at this level of software design to establish the **data structures** and algorithms to be used. The outcome of all stages is documented in DDS. It defines the **logical structure** of each module as well as **its interfaces with other modules**.

RE and Design Example: Task Management Software

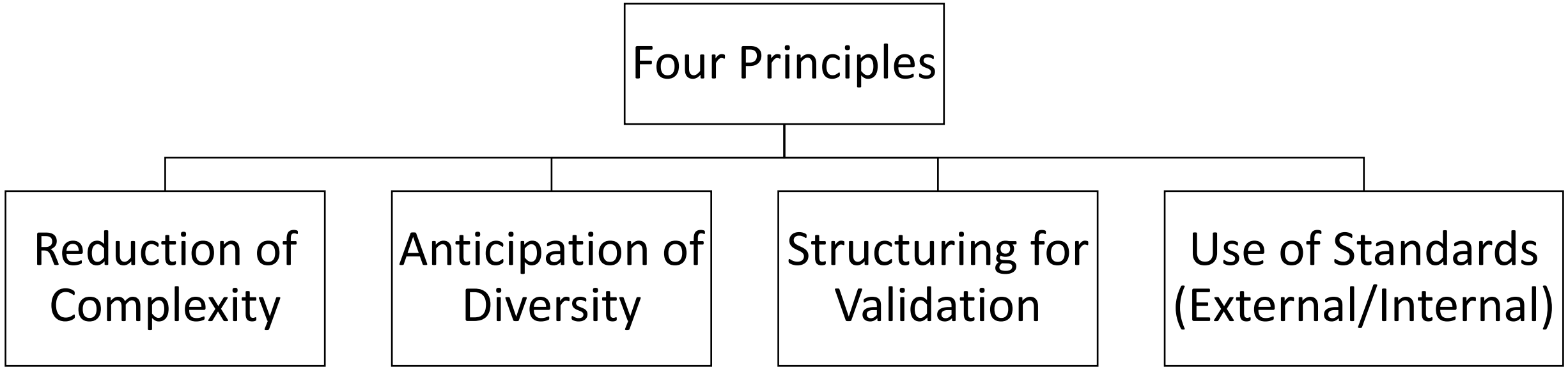
Architectural Design: Define the overall structure of the system. For this task management software, you might decide on a **web-based architecture** with **client-server** model where the **server handles logic and database** interactions, and the **client provides interactive user interfaces.**

High-Level Design: Break down the architecture into major components or modules such as **User Management, Task Management, Notification System, and Database**. Define the relationships and data flow between these modules.

Detailed Design: Focus on the specifics of each module. For instance, the **Task Management module** might involve detailed designs of the database **schema for tasks, classes, and methods to handle task creation, updates, and queries**. Interfaces for each module should also be defined to ensure they can interact seamlessly.

Implementation

Phase where we take care of realizing the design of the system and create a natural softer system



High Cohesion
Low Coupling

Anticipate for changes
Modular

Easily Testable
TDD, Mockito

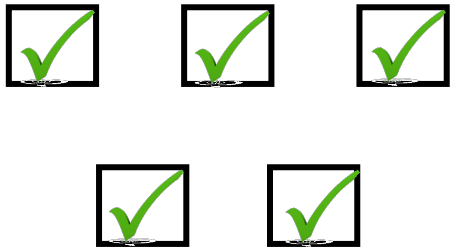
Naming Standard
Client Regulations

Verification & Validation

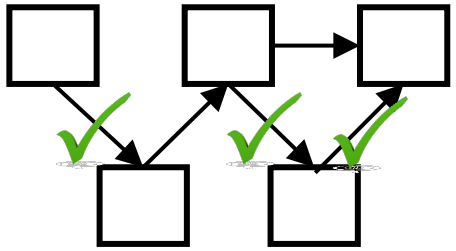
Phase that aims to check that software system meets its specifications and fulfils its intended purpose

Verification: did we build the system right?
Validation: did we build the right system?

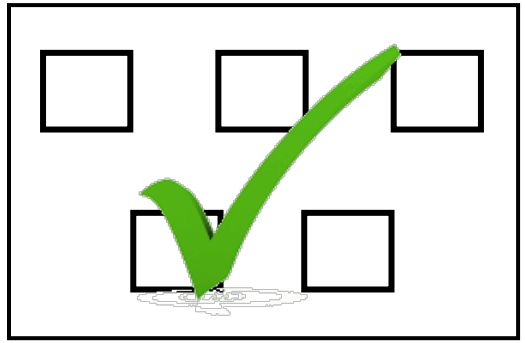
Unit



Integration

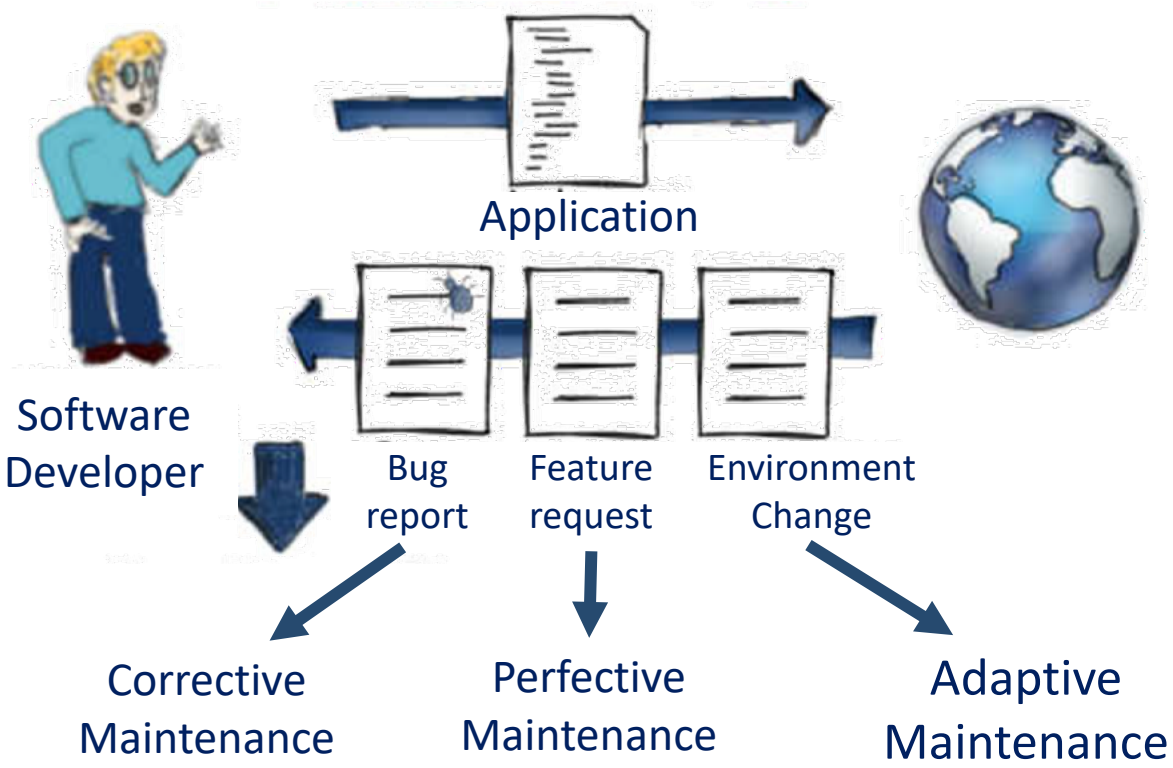


System



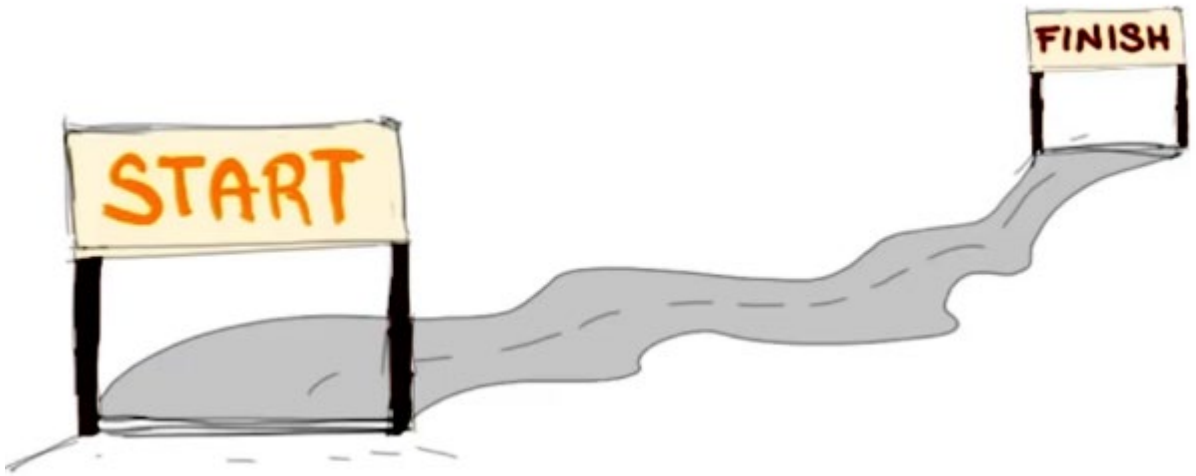
Maintenance

Once Software released to final users and in operation, many things can happen: environment change -new libraries, new systems, additional functionality requests, bug reports



- Maintenance is a fundamental and expensive phase
- *Regression testing* – retesting a modified version of software before release, no introduction of new errors

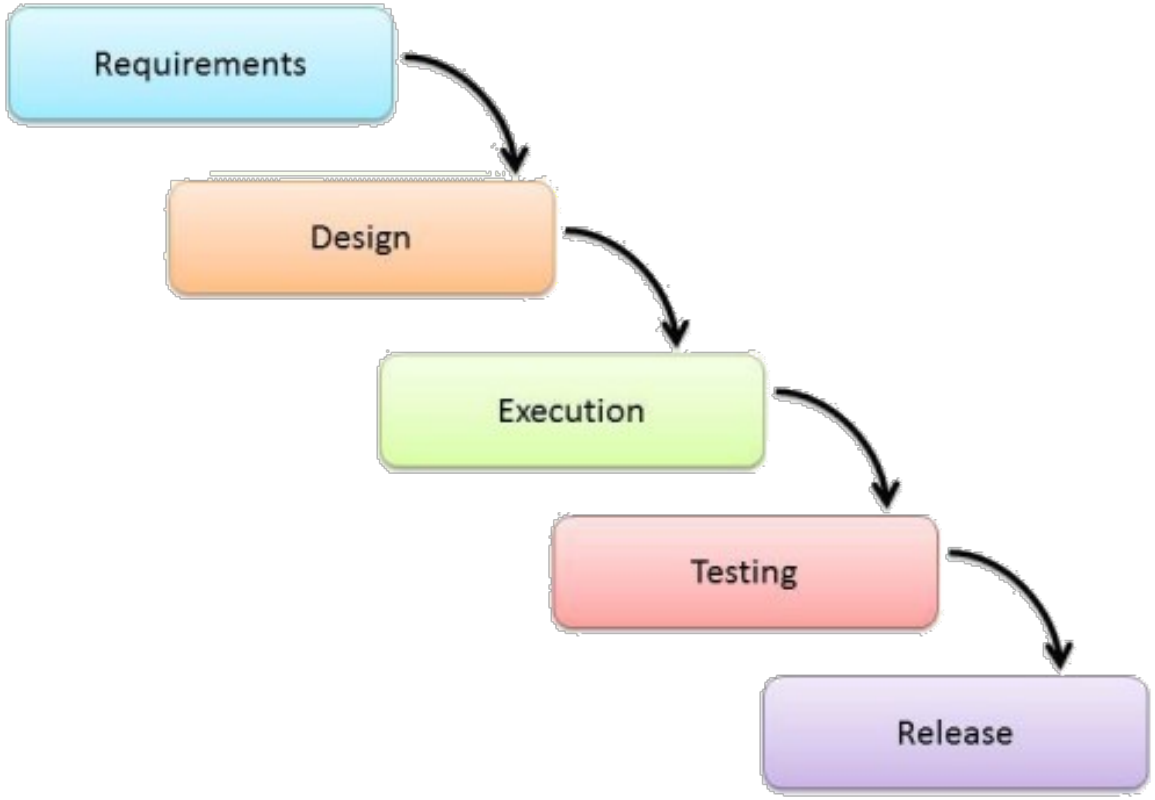
Software Process Model/ Life Cycle Model



Functions:

- Order of activities
- Transition Criteria between Activities
- What should we do next and for how long?

Waterfall Method

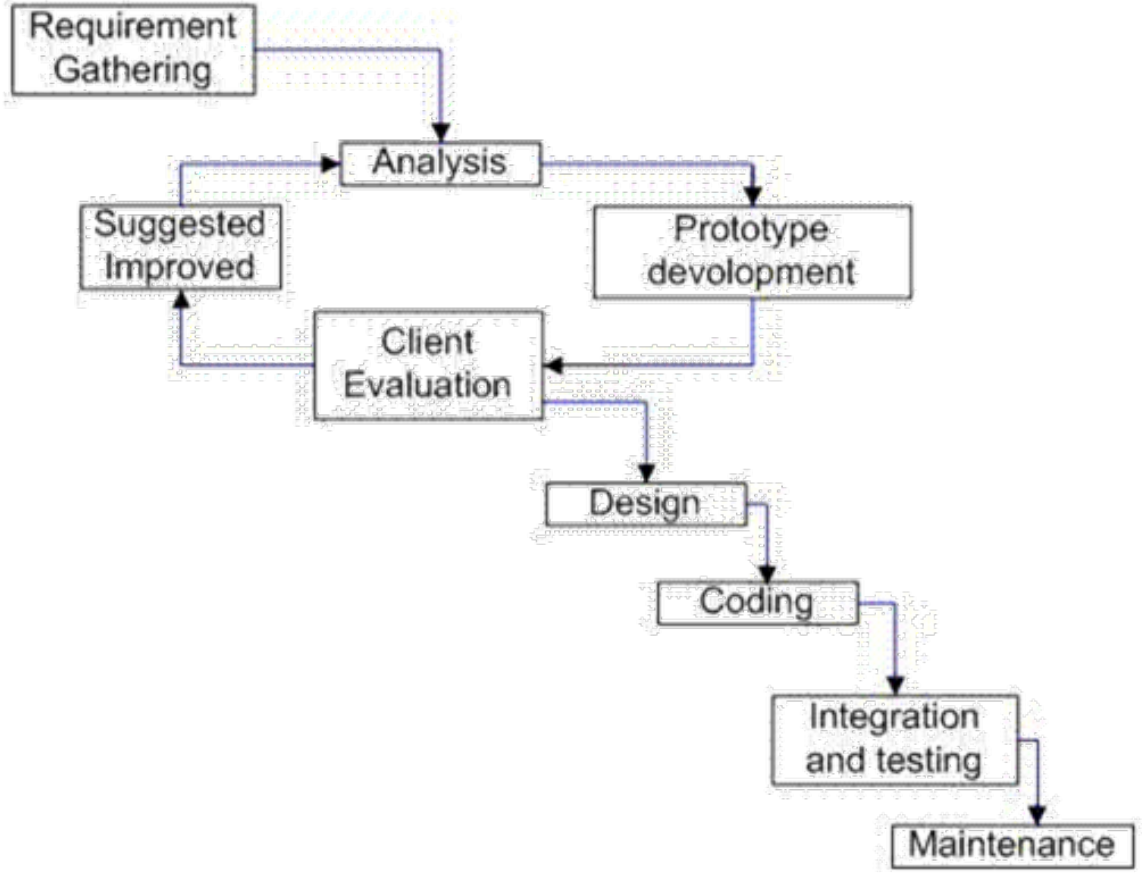


Early Error Detection

No Flexibility

- Project progresses in an orderly sequence of steps
- Pure Waterfall model performs well for software products with a stable product definition- well known domain, technologies involved, Request for Proposals (RFP)
- Waterfall method finds errors in early local stages
- Not flexible- not for projects where requirements change, developers not domain experts, or technology used are new and evolving

Evolutionary Prototyping



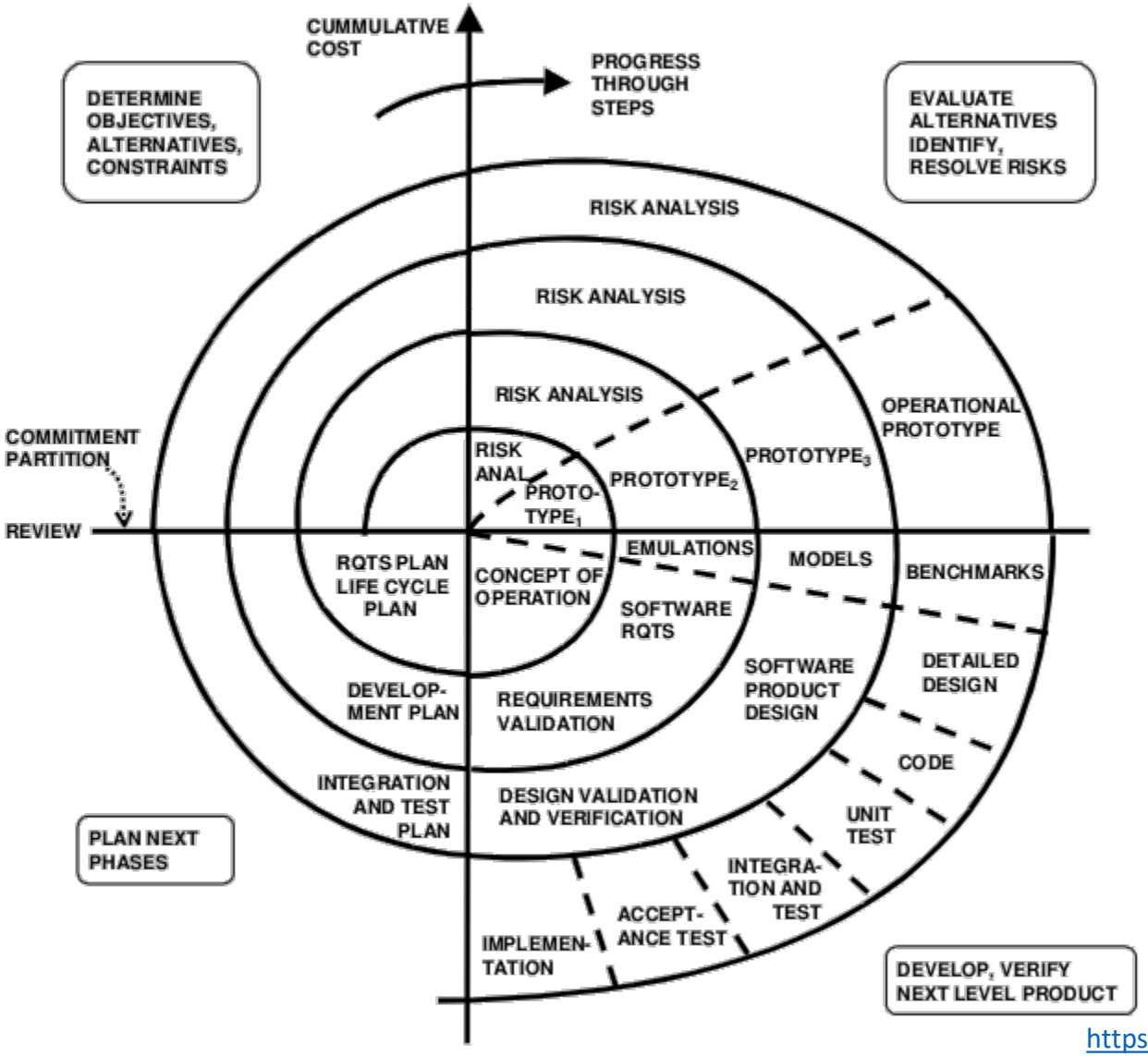
Immediate feedback
Helps Requirements understanding

Difficult to Plan
Can deteriorate to code-and-fix

- Prototypes that evolve into the final system through an iterative incorporation of user feedback.
- Ideal when not all requirements are well-understood. System keeps evolving based on customer feedback

Spiral Method

Incremental risk-oriented lifecycle model with 4 main phases



Risk Reduction
Functionality can be added
Software produced early, Early feedback

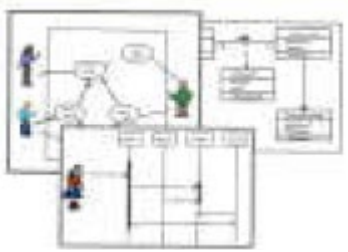
Specific Expertise
Highly dependent on risk analysis
Complex, Costly

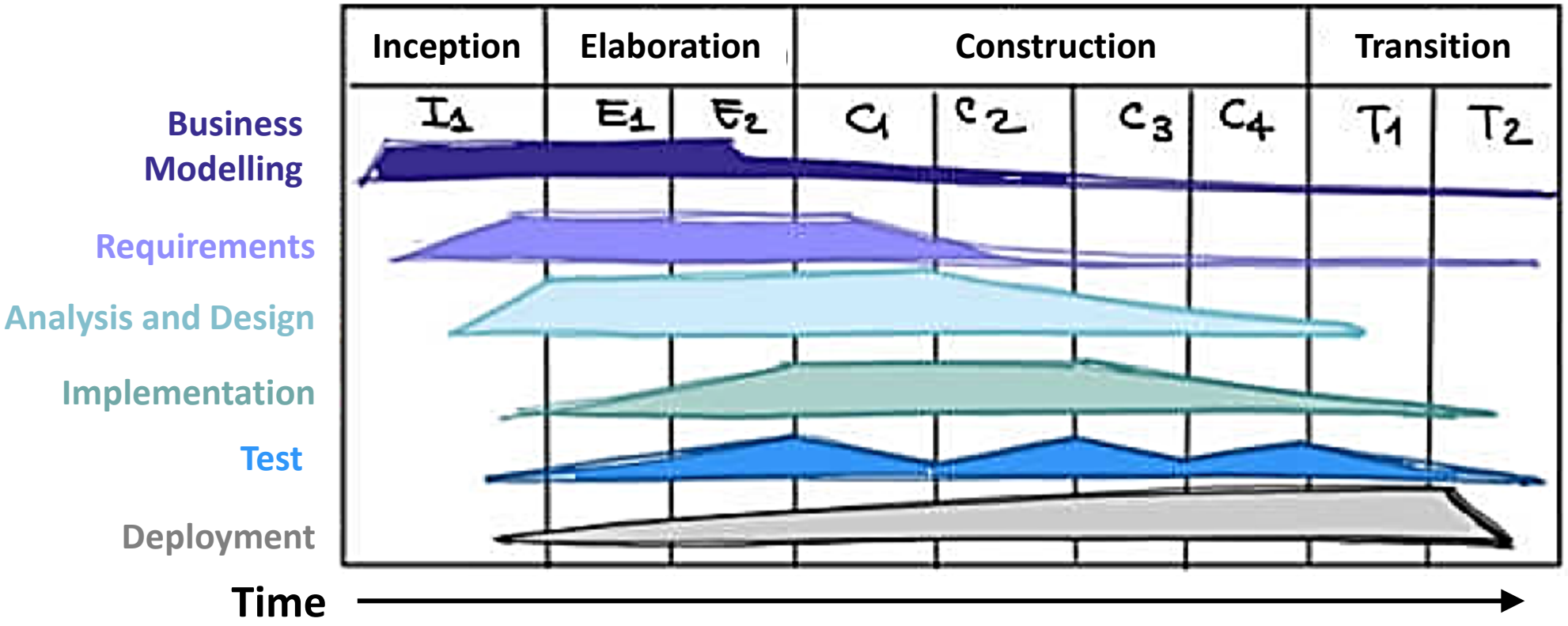
https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/

Spiral Method Example: Online Banking App

Spiral Pass	Objectives	Prototype/Deliverable	Key Risks to Analyze & Mitigate
1	• Clarify core features (login, view balance)	• Paper- or wireframe-prototype of UI	• **Requirements risk:** Do users really need X? • **Usability risk:** Is the UI intuitive?
2	• Architect back end (API design, data schema)	• Stubbed API service	• **Technical risk:** Will chosen database scale? • **Performance risk:** Can queries return in <200 ms?
3	• Implement authentication & security layers	• Minimal working login flow	• **Security risk:** Are credentials stored safely? • **Compliance risk:** Meets banking regulations (e.g. PCI)
4	• Add transactions, statements, transfers	• Full “read-only” transaction view	• **Integration risk:** Third-party payment gateway reliability • **Error-handling risk:** What if network drops?
5…n	• Extend to mobile, add notifications, budgeting tools	• Increasingly complete app	• **Mobile-specific risk:** Battery impact, offline behavior • **Adoption risk:** Will customers use feature Y?

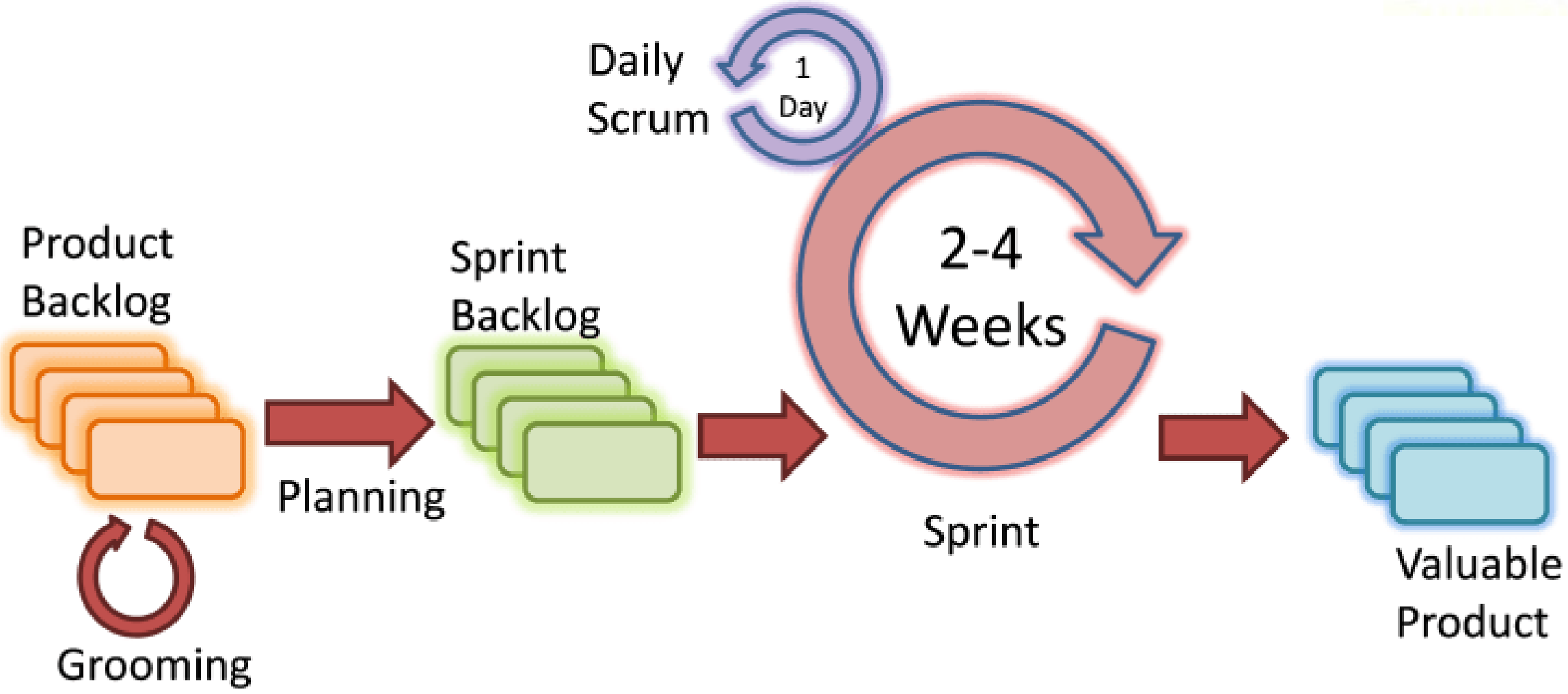
Rational Unified Process (RUP)





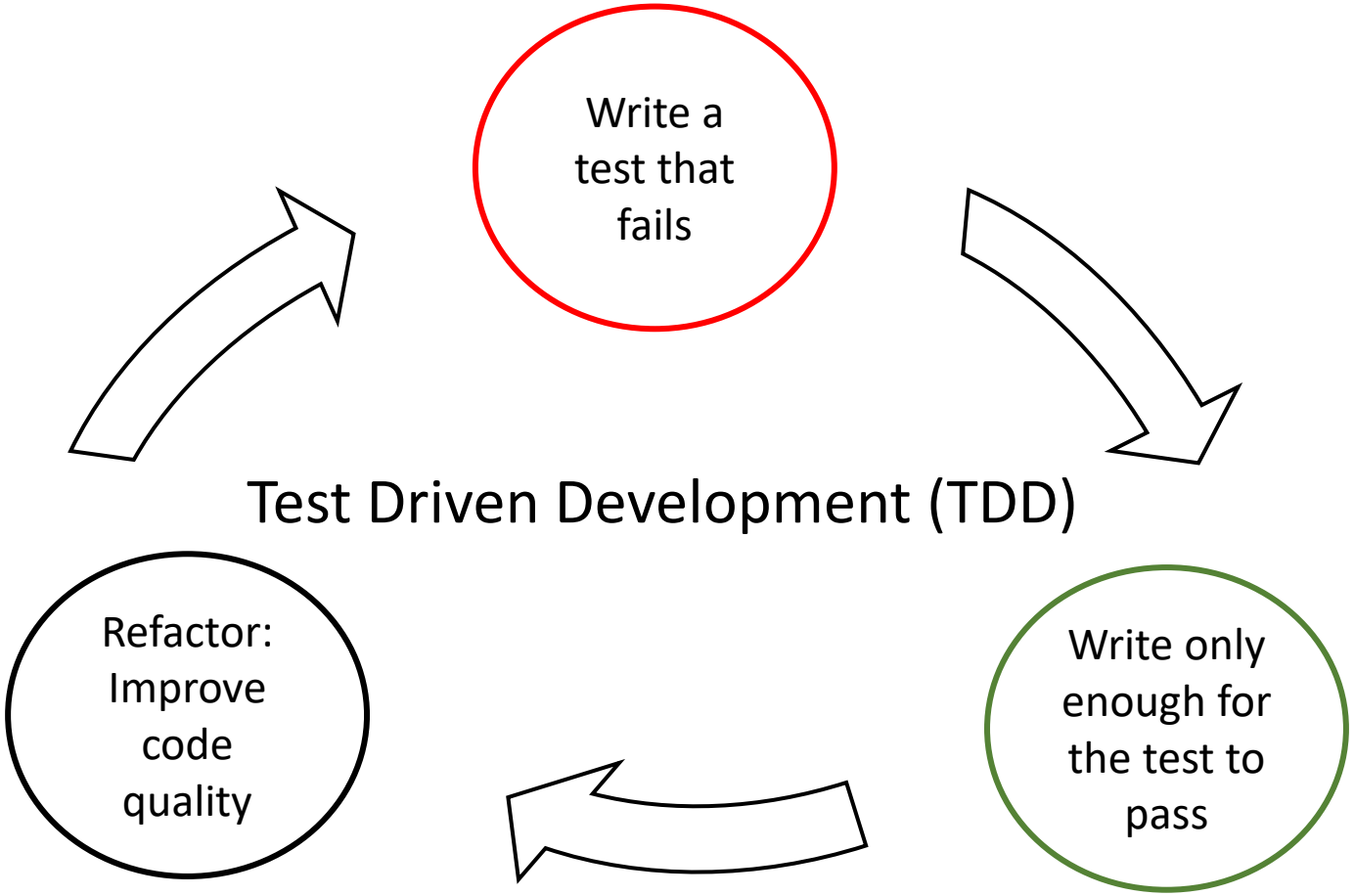
- Popular Process based on UML. Works iteratively, performs 4 phases in each iteration
- Inception phase: Scope the system - Scope of project, domain, initial cost, budget estimates
- Elaboration phase: domain analysis and basic architecture
- Construction phase: Bulk of development
- Transition: From development to production, available to users

Agile - Scrum



Highly iterative and incremental development process

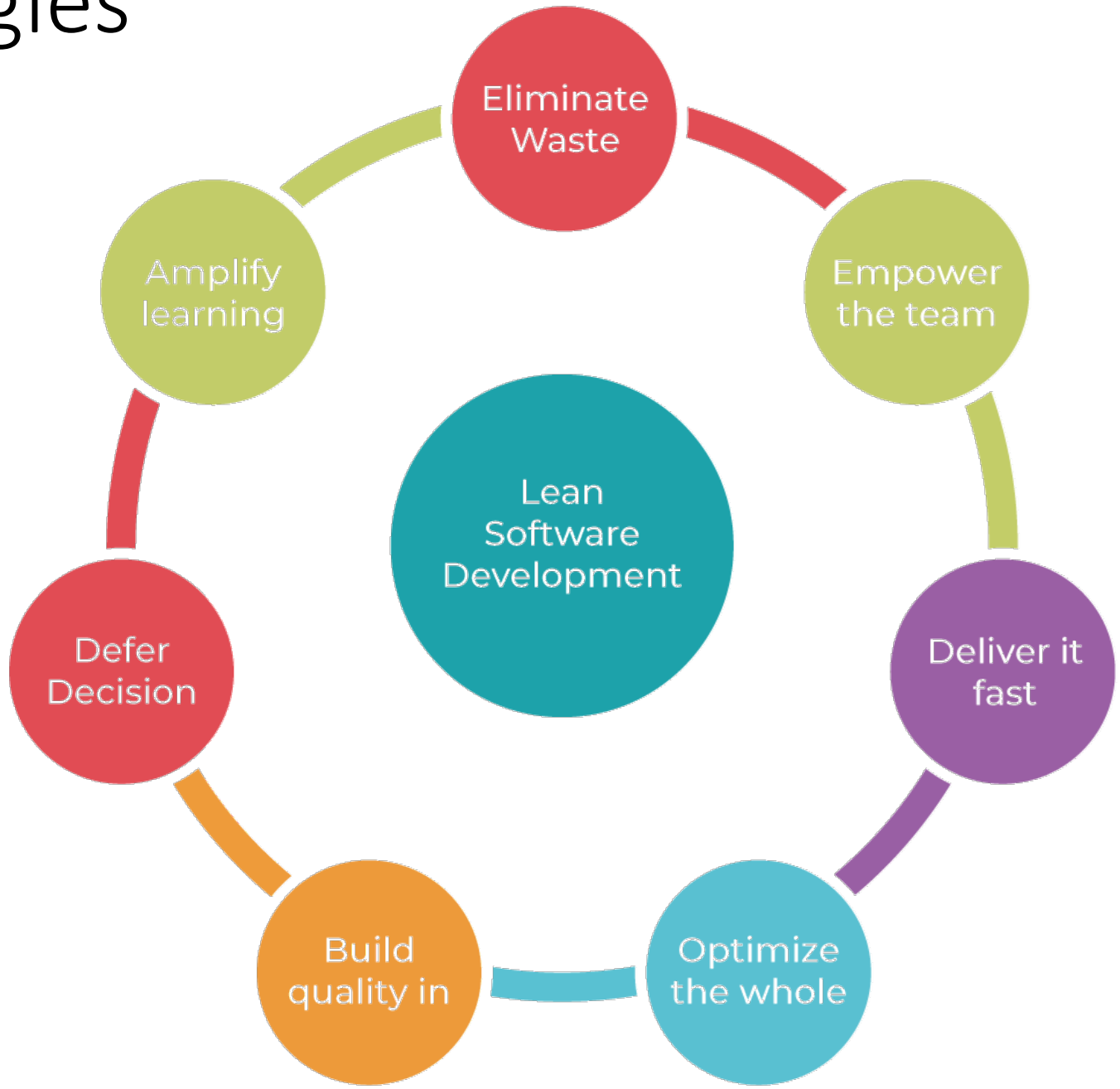
Agile - XP



- Highly iterative and incremental development process
- Focus on Continuous Integration

Other Agile Methodologies

Kanban: Simplest in IT World;
May Pose time related problems



Some industry-based examples

Waterfall

Military And Aircraft Programs Where Requirements Are Declared Early On And Remain Constant

Evolutionary Prototyping

- **Company**: **Broderbund Software**.
- **Project**: The creation of the original **"Prince of Persia"** video game. The initial version of the game was created and then improved upon based on feedback and playtesting.

Some industry-based examples

Spiral

- NASA's space shuttle program in the 1970s
- Gantt Chart Software – GanttPRO

Agile

- **Apple, IBM, Microsoft, and Procter & Gamble**
- **Cisco:** defects were reduced by 40% when compared to waterfall
- **Barclays:** 300% increase in throughput
- **Panera Bread:** 25% increase in company sales
- **PlayStation Network:** Saved the company $30 million a year

Choosing the right Software Process Model

Requirements Understanding

Expected Lifetime

Risk

Schedule Constraints

Interaction with Management/Customers

Expertise

As much influence over a project's success as any other major planning decision

Industry Standards: Factors affecting choice of project LCM

- Degree of Project Complexity
- Work/Time Flexibility
- Project Focus/ Client involvement
- Size of organization
- Role Specialization
- Budget

https://asana.com/resources/project-management-methodologies
https://thedigitalprojectmanager.com/projects/pm-methodology/project-management-methodologies-made-simple/

Industry Standards: Factors affecting choice of project LCM

Factors	Waterfall	Evolutionary Prototyping	Agile Methodologies	Spiral
Unclear User Requirements	Poor	Good	Excellent	Excellent
Unfamiliar Technology	Poor	Excellent	Poor	Excellent
Complex System	Good	Excellent	Poor	Excellent
Reliable System	Good	Poor	Good	Excellent
Short time schedule	Poor	Good	Excellent	Excellent
Strong Project Management	Excellent	Excellent	Excellent	Excellent
Cost Limitation	Poor	Poor	Excellent	Poor
Visibility of stakeholder	Good	Excellent	Excellent	Excellent
Skills Limitation	Good	Poor	Poor	Poor
Documentation	Excellent	Good	Poor	Good
Component Reusability	Excellent	Poor	Poor	Poor

https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/

Industry Standards: Most Popular Methods

1. **Agile** – collaborating to iteratively deliver whatever works
2. **Scrum** – enabling a small, cross-functional, self-managing team to deliver fast
3. **Kanban** – improving speed and quality of delivery by increasing visibility of work in progress and limiting multi-tasking
4. **Scrumban** – limiting work in progress like Kanban, with a daily stand up like Scrum
5. **Lean** – streamlining and eliminating waste to deliver more with less
6. **eXtreme Programming (XP)** – doing development robustly to ensure quality
7. **Waterfall** – planning projects fully, then executing through phases
8. **PRINCE2** – controlled project management that leaves nothing to chance
9. **PMI's PMBOK** – applying universal standards to Waterfall project management

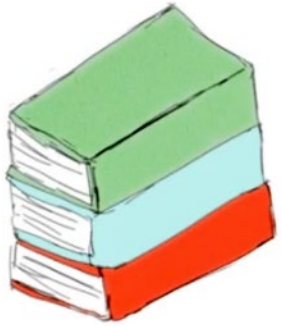
https://thedigitalprojectmanager.com/projects/pm-methodology/project-management-methodologies-made-simple/

Lifecycle Documents

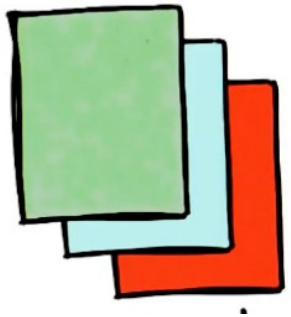
Documenting the activities carried out during the different phases of the lifecycle is a very important task.

Can be used for different purposes like:

- Communicate details of the software systems to different stakeholders
- Ensure the correct implementation of the system
- Facilitate maintenance and so on.



IEEE Documents



Light-weight Documents

Classic Mistakes : People

Heroics

Work Environment

People Management

Classic Mistakes : Process

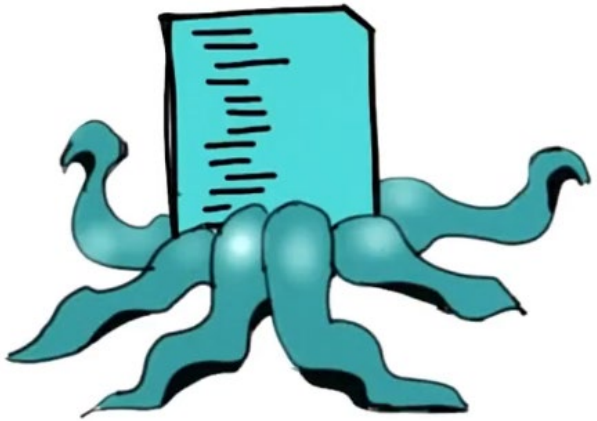
Schedule Issues

Planning Issues

Failure

Classic Mistakes : Product

Gold Plating of Requirements

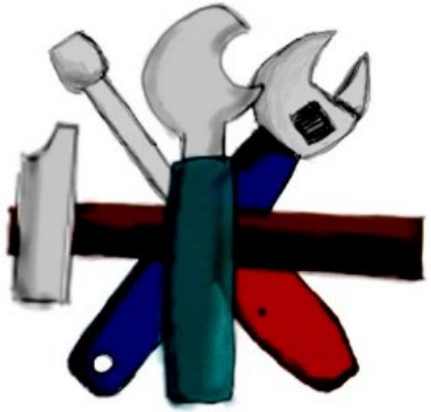


Feature Creep

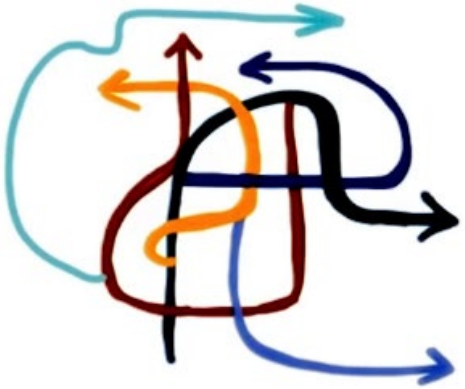
Research ≠ Development

Classic Mistakes : Technology

Silver-Bullet Syndrome



Switching Tools

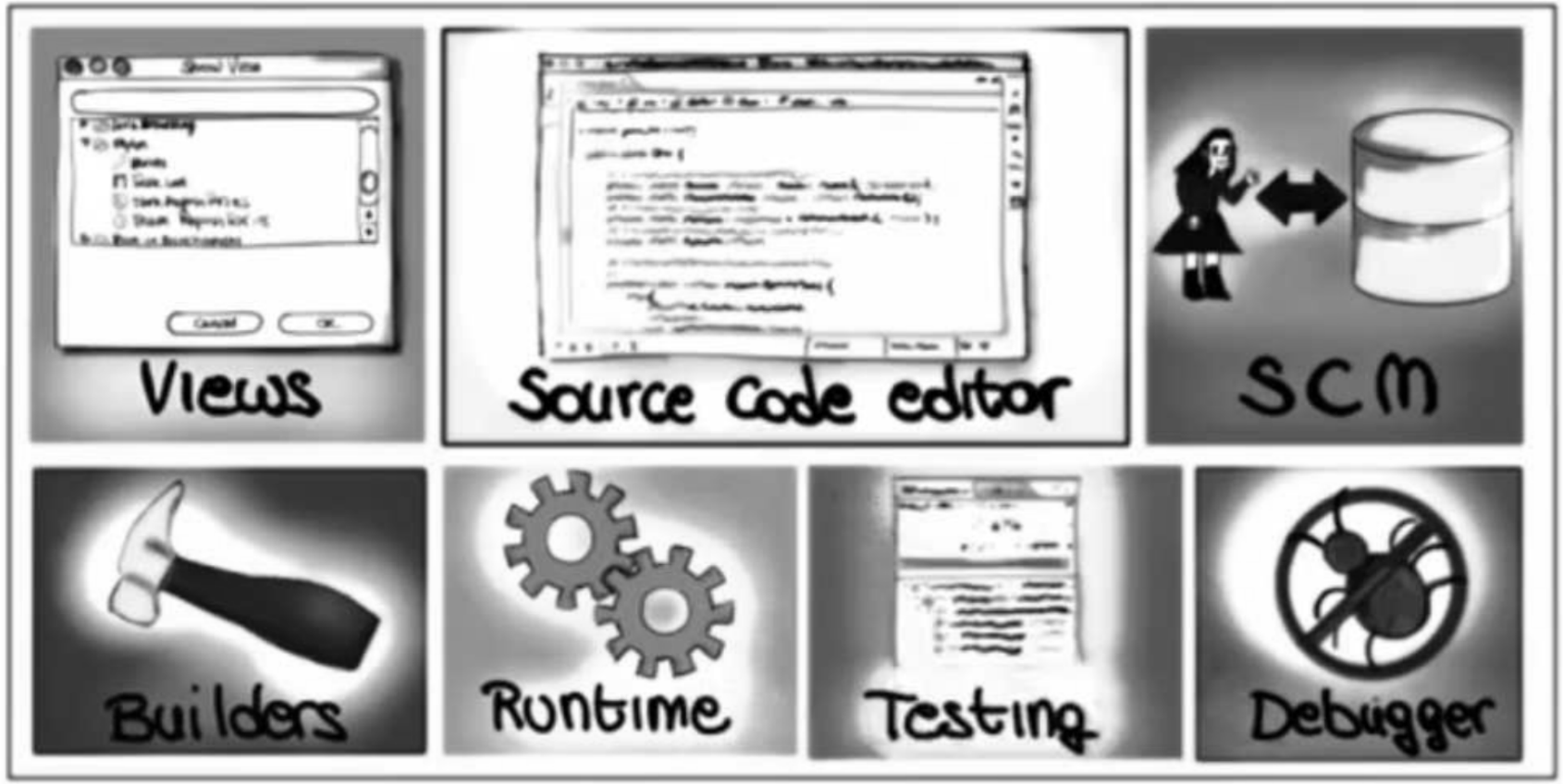


No version control

5 min break

Quizizz

What is an IDE?



- Integrated Development Environments (IDEs) are software applications that support developers in many of their everyday tasks, such as writing, compiling, and debugging code.
- Some IDEs are designed to support only one programming language such as Java, while others can be used for various languages
- Most popular Java IDEs: VSCode, IntelliJ IDEA, Eclipse, Netbeans

What is an IDE?- VSCode

- VS Code is a lightweight, open-source IDE developed by Microsoft.
- It is highly versatile and supports multiple languages, including Java, Python, JavaScript, and C++.
- Plug-ins provide additional functionality to VS Code, allowing developers to customize the IDE to suit their specific needs. For example, the "Extension Pack for Java" adds Java support, and there are many other plugins for various programming languages and tools, such as Git, Docker, and more.
- VS Code is available on all major operating systems: Mac, Windows, and Linux.

Maven

- Powerful build management tool that can be used for building and managing any Java based project.
  - Automates compilation (turning your .java into .class files)
  - Resolves dependencies (downloads the exact library versions you need)
  - Generates documentation (Javadoc, dependency reports, test reports)
  - Packages artifacts (JARs/WARs) without hand-written scripts
  - Installs those artifacts into local or remote repositories
- Based on POM (Project Object Model)
  - POM files are XML files that contain information related to the project and configuration information such as dependencies, source directory, plugin, goals etc. used by Maven to build the project.

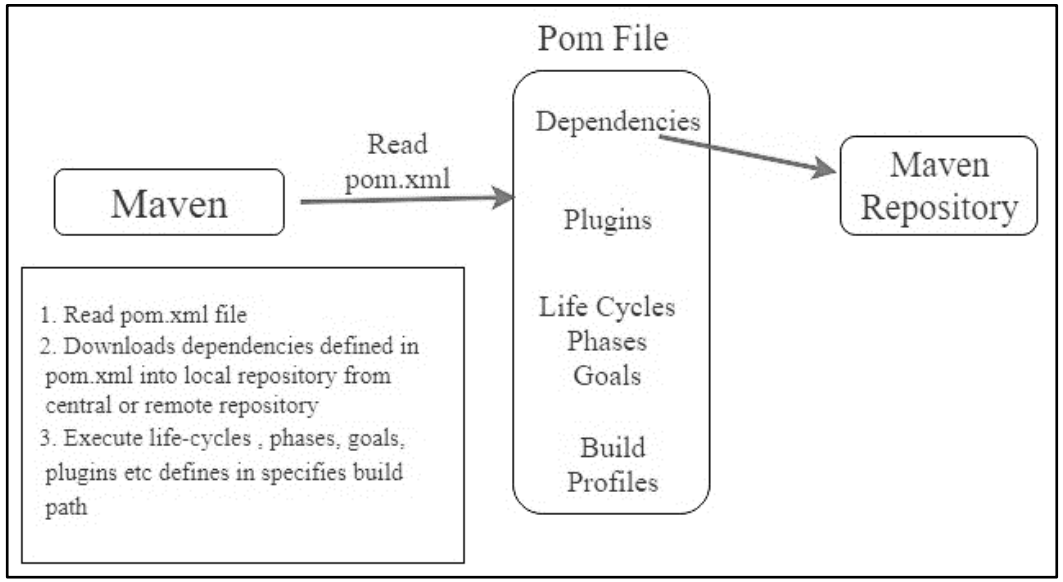


Image courtesy: https://www.geeksforgeeks.org/introduction-apache-maven-build-automation-tool-java-projects/

Junit Testing

- Lightweight framework for creating repeatable tests for your application
- Unit testing in Java
  - Imposes developers' discipline
  - Provides incremental specification
  - Avoids regression errors
  - Allows for changing with confidence
- Very helpful in Test driven development

JUnit

Junit Best Practices

- Keep test cases separate from source code (src/main/tests)
- Method names need to be specific
- Keeping tests simple and focused on one feature or expected result
- Making sure to test for edge cases and not only “perfect scenarios”
- Use appropriate assertions to verify what's expected and what the function returns (actual)
- Tests are repeatable and reliable

1.assertEquals(expected, actual): Verifies that the expected and the actual values are equal.
2.assertNotEquals(first, second): Verifies that two values are not equal.
3.assertTrue(condition): Verifies that the condition is true.
4.assertFalse(condition): Verifies that the condition is false.
5.assertNull(object): Verifies that the object is null.
6.assertNotNull(object): Verifies that the object is not null.

Demo – Run unit tests with Maven in VSCode

- **Install JDK**: Download and install OpenJDK.
- **Set Up VSCode**: Install VSCode and Java Extension Pack.
- **Create Java Project**: Set up a new Java project in VSCode.
- **Add and Run Java Classes**: Create and run simple Java classes.
- **Configure Debugging**: Set up launch.json for custom debugging.
- **Install Maven**
- **Create Maven Project**: Initialize and configure a Maven project.
- **Manage Dependencies**: Edit pom.xml to add dependencies like JUnit.
- **Write and Run JUnit Tests**: Develop and execute unit tests.
- **Demonstrate Debugging**: Use breakpoints and debug features.

Demo – Run unit tests with Maven in VSCode

How to Install JDK

1. Browse to the AdoptOpenJDK website.
2. Choose your **Operating System**
3. Package type is **JDK**
4. Choose **OpenJDK 16** or the latest version
5. Click the **Latest release** download button to download the package.

How to Install and setup VSCode

1. Browse to https://code.visualstudio.com/download
2. Select Download based on your device type
3. Follow the prompted steps and continue
4. Once installed, open VS Code and install the **Extension Pack for Java** from the Extensions Marketplace. This pack includes the essential tools for Java development in VS Code.

Reference: https://code.visualstudio.com/docs/java/java-tutorial. You can also download the coding pack and the extension pack instead from the above link.

Demo – Run unit tests with Maven in VSCode

How to Install Maven

1. Browse to the Apache Maven Project Downloads page.
2. Install Maven
3. Add Maven to the PATH environment variable
4. Set JAVA_HOME (if not already set):

Demo – Run Configurations

By default using launch.json. Doesn’t need to be touched unless:

- Custom Run Configurations: If you have specific requirements for launching your application, such as setting environment variables, JVM arguments, or choosing among multiple main classes, you can define these configurations in launch.json.
- Debugging with Specific Parameters: If your application requires command line arguments to run, you can specify these in the launch.json file. This is useful when your program expects user input that would typically be provided when running the program from a command line.
- Complex Build and Launch Processes: For applications that require more complex build setups or pre-launch tasks (like compiling resources other than Java files, moving files, etc.), you can define pre-launch tasks in launch.json that execute these commands before debugging starts.
- Remote Debugging: When you need to connect to a Java application running on a different machine or environment, you can configure the necessary remote debugging settings in launch.json.

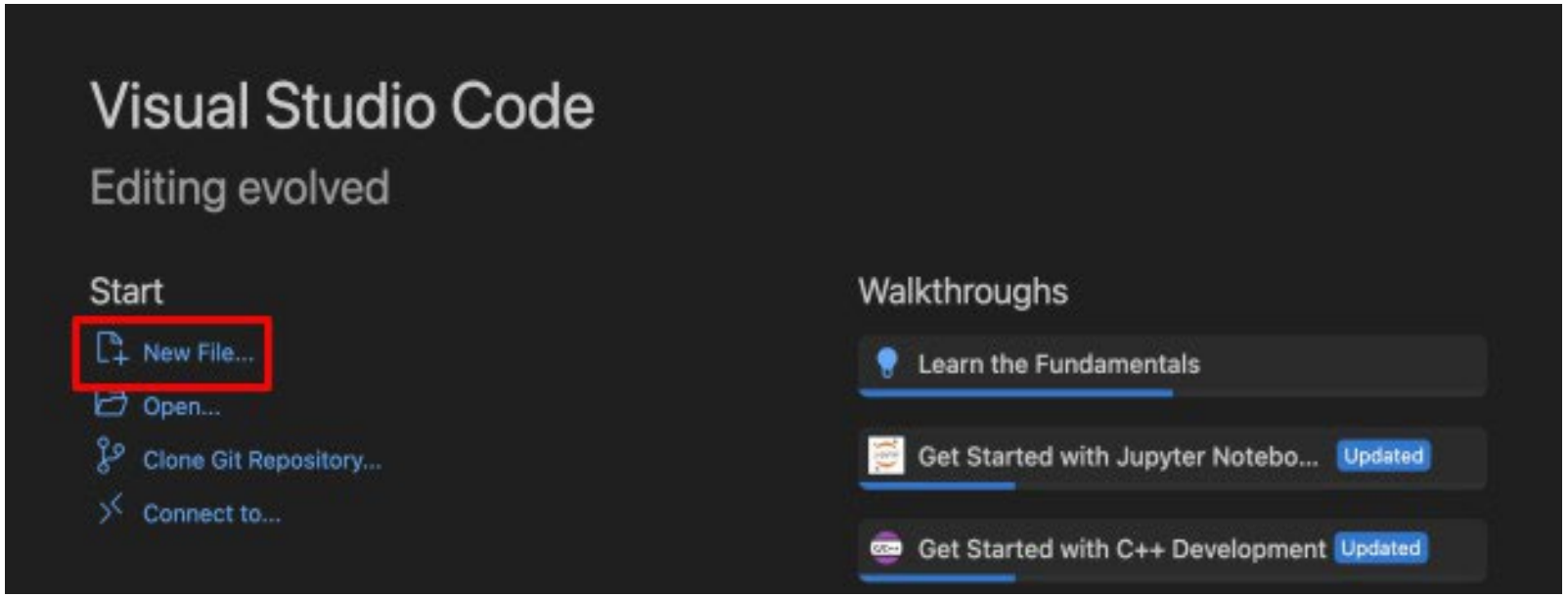
Debug:

Add breakpoints – conditional breakpoint; data breakpoint; logpoints

Demo – create project

- **<u>How to create a new project with Maven</u>**

- Select New File
- Search Java Project in the search bar and click New Java Project
- Select No Build Tools and Name the project myFirstProject
- Open the Command Palette (Ctrl + Shift + P or Cmd + Shift + P)
- Search for Java: Configure Java Runtime and select it. Ensure that the JDK downloaded before is the one selected

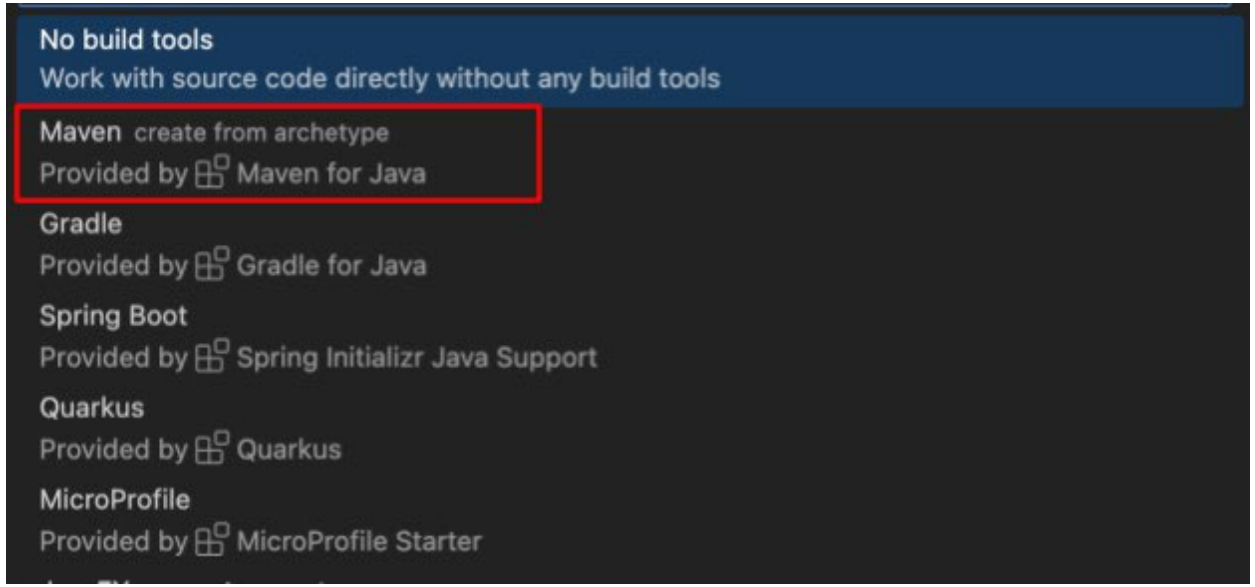


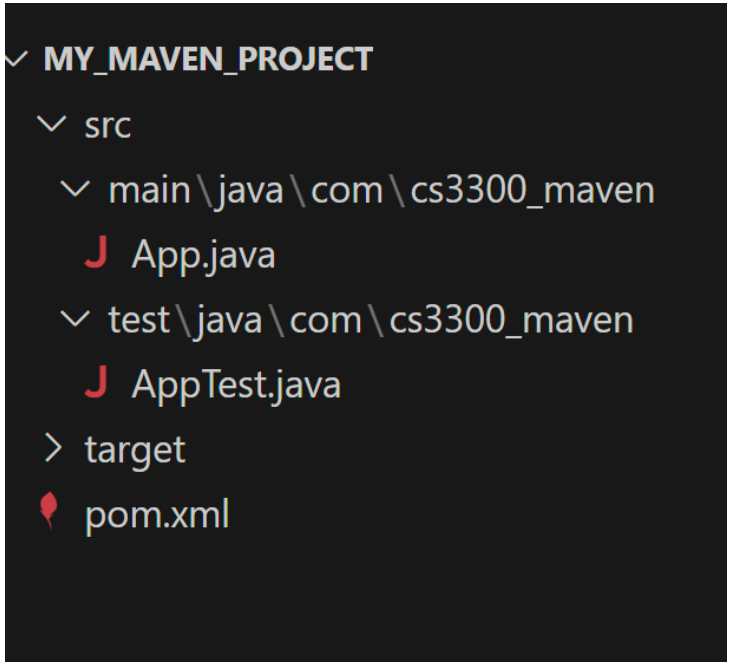
Demo – create project

How to create a project with Maven

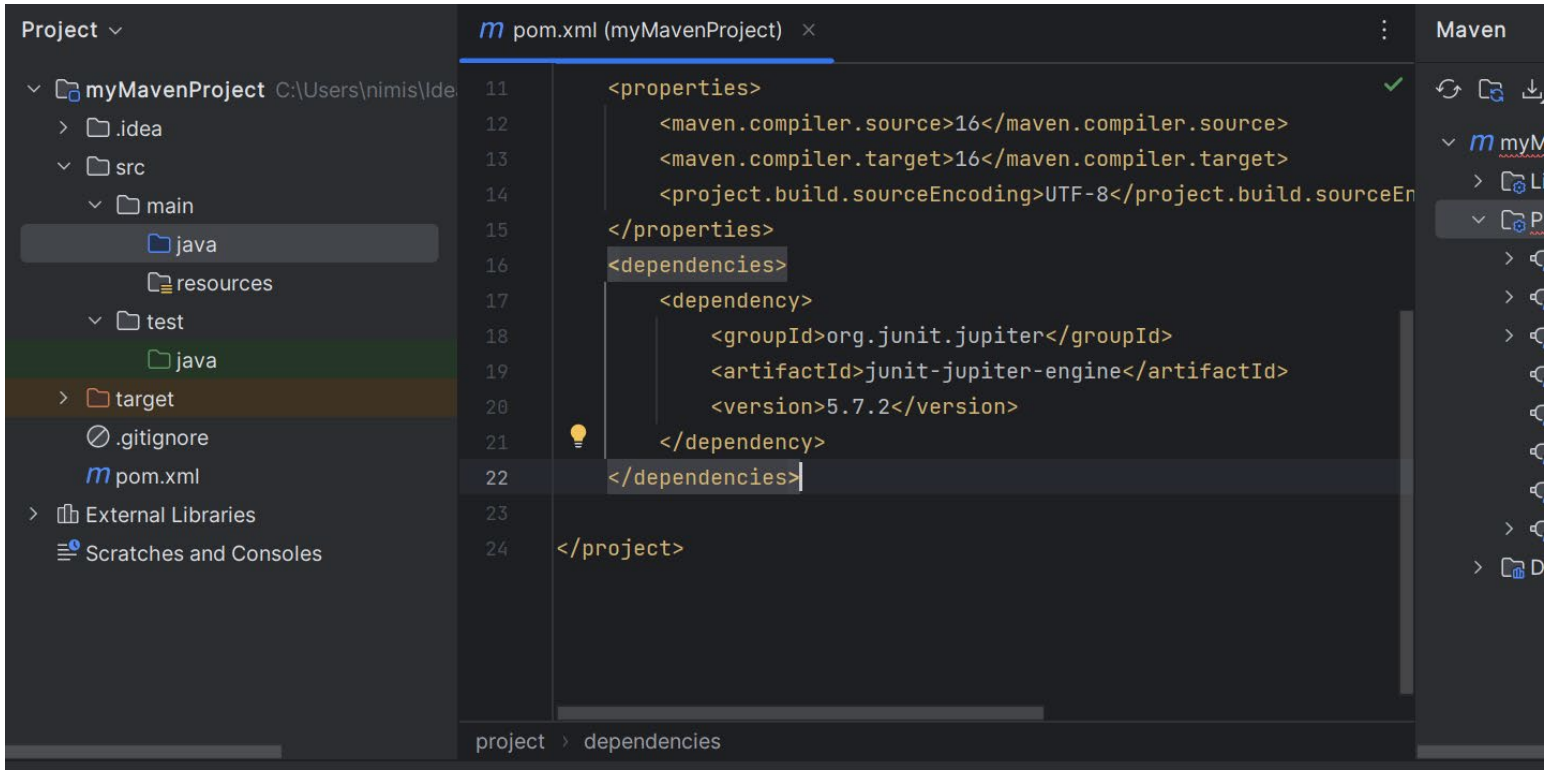
- Choose groupid and artifactid
  - Groupid: uniquely identifies your project across all projects. Kind of like package. *com.cs3300_maven*
  - artifactId is the name of the project. Let's say *my_maven_project*

- You will be able to see pom.xml file in the package explorer





Demo- Editing pom.xml to add dependency



Edit pom.xml to add Junit as a dependency

1. In **pom.xml**, press Alt + Insert and select **Dependency**.
2. In the dialog that opens, type org.junit.jupiter:junit-jupiter in the search field. Locate the necessary dependency in the search results and click **Add**.

Demo- Adding code and test cases

- Create class in src/main/java. *AddConcatenate.java*
- Add 2 methods, 1 to add 2 numbers and 1 to concatenate 2 strings.
- Create corresponding 2 junit tests to test the 2 functions
- Run tests

What would be good test cases here?

- Add: addition with 0, addition with negative numbers, addition resulting in 0, addition with MAX INTEGER and MIN INTEGER (underflow and overflow)
- Concat: Concatenate with an Empty String, concatenate Special Characters, Concatenate with Numbers, Concatenate Two Empty Strings, concatenate non English characters