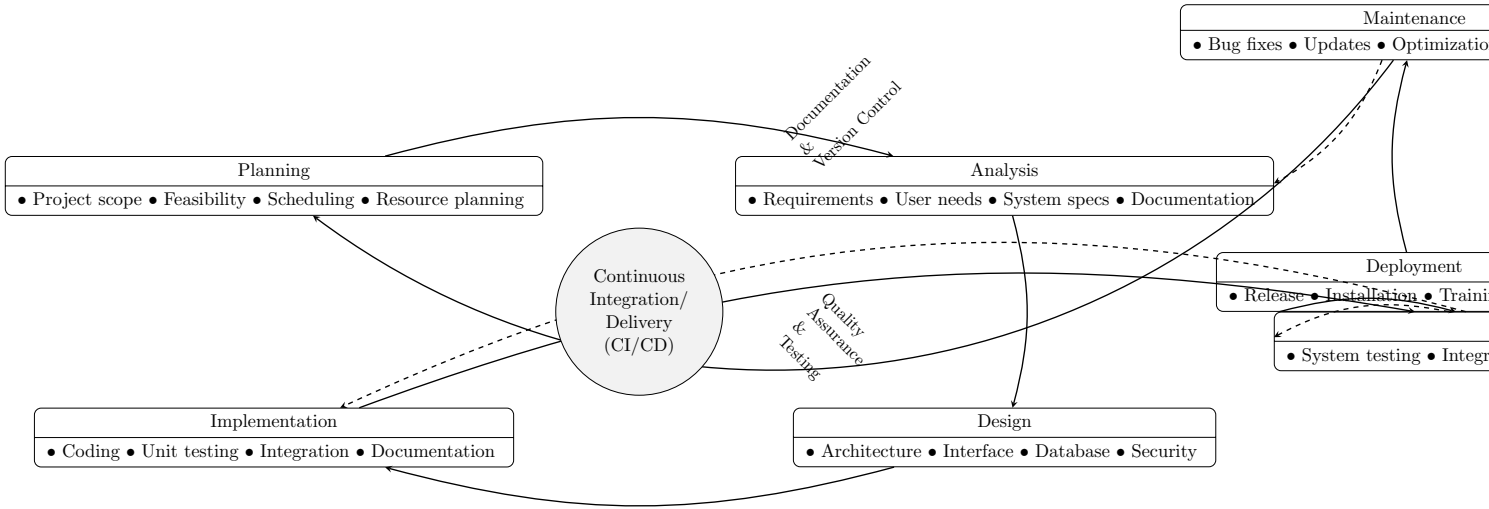
Computing and Software Engineering

GET211

September 23, 2025

Elements of Software Engineering

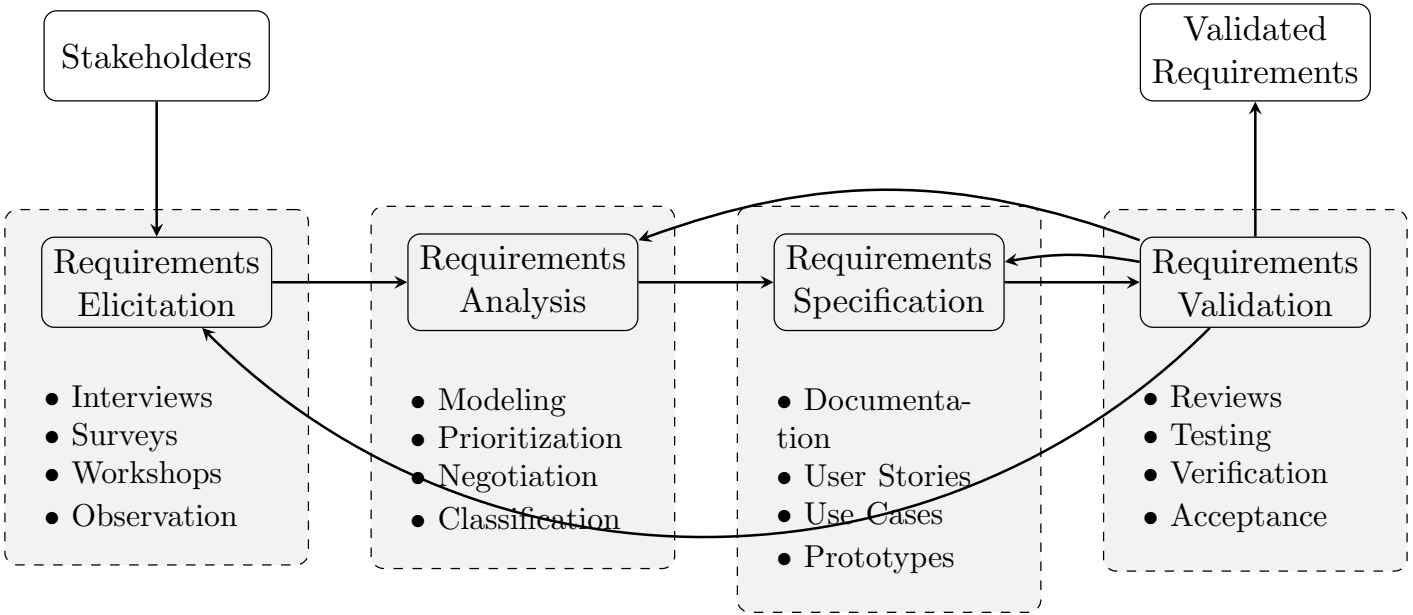
Software engineering is the systematic application of engineering principles to the development, operation, and maintenance of software. It involves a combination of processes, methodologies, tools, and techniques to ensure that software systems are reliable, efficient, maintainable, and meet user requirements.


Maintenance
• Bug fixes • Updates • Optimizatio
Documentation
&
Version Control
Planning
• Project scope • Feasibility • Scheduling • Resource planning
Analysis
• Requirements • User needs • System specs • Documentation
Continuous
Integration/
Delivery
(CI/CD)
Quality
Assurance
&
Testing
Deployment
• Release • Installation • Traini
• System testing • Integr
Implementation
• Coding • Unit testing • Integration • Documentation
Design
• Architecture • Interface • Database • Security

Requirements Engineering

The process of gathering, analyzing, documenting, and validating the needs and constraints of stakeholders for a software system.
Key Activities:
- Requirements Elicitation: Identifying what the stakeholders need through interviews, surveys, and observation.
- Requirements Analysis: Evaluating the requirements for feasibility, consistency, and completeness.
- Requirements Documentation: Creating artifacts like Software Requirements Specification (SRS).
- Requirements Validation: Ensuring the documented requirements align with stakeholder needs.

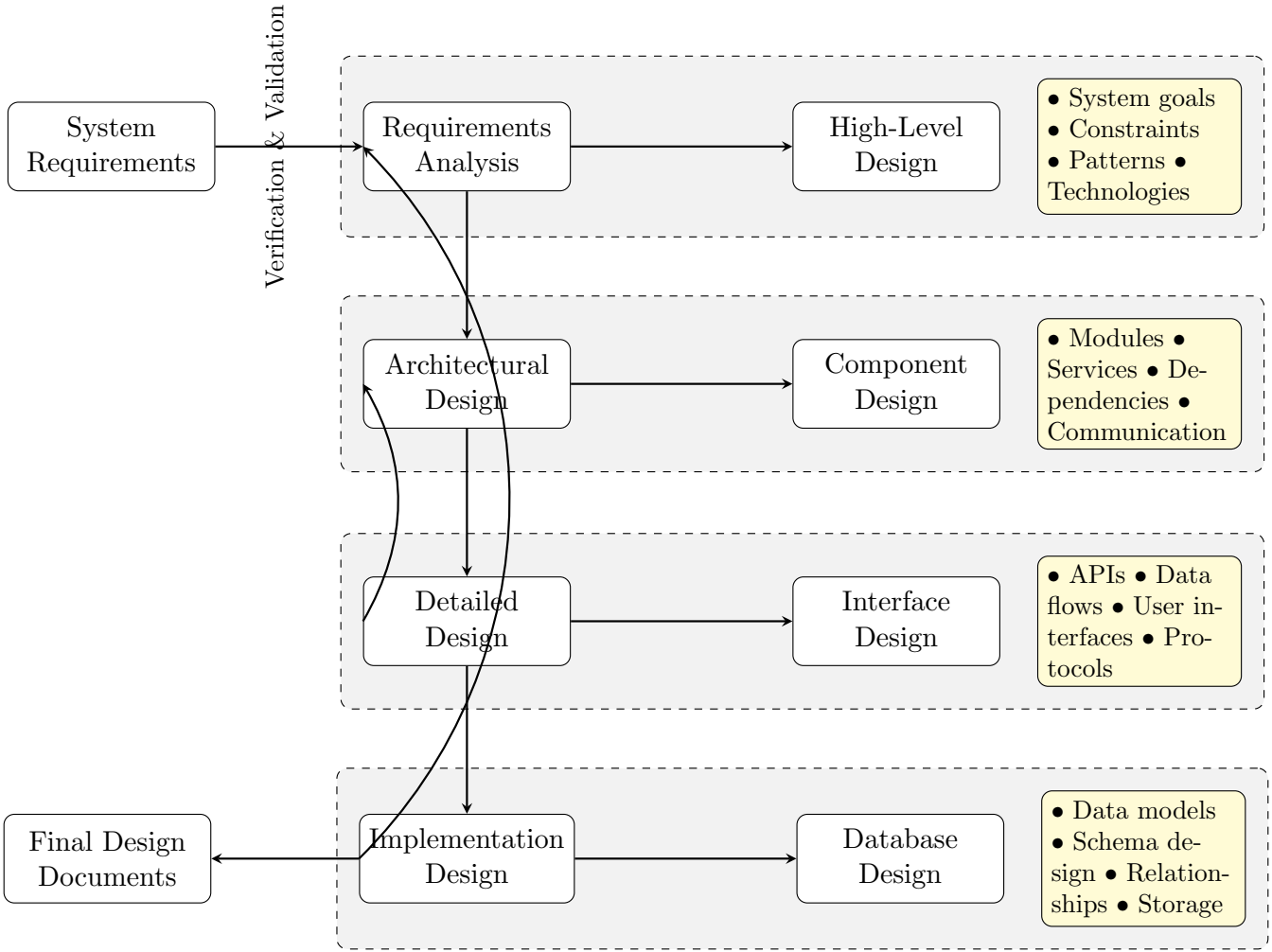
Requirement Engineering



Software Design

The process of defining the architecture, components, interfaces, and data for a system to satisfy specified requirements.
Key Elements:
- Architectural Design: High-level structure of the system, including components and their interactions.
- Detailed Design: Focuses on individual components, algorithms, and data structures.
- Design Principles:
- Modularity: Dividing the system into manageable modules.
- Cohesion and Coupling: High cohesion within modules and low coupling between them.
- Scalability and Maintainability: Ensuring the design supports future growth and updates.

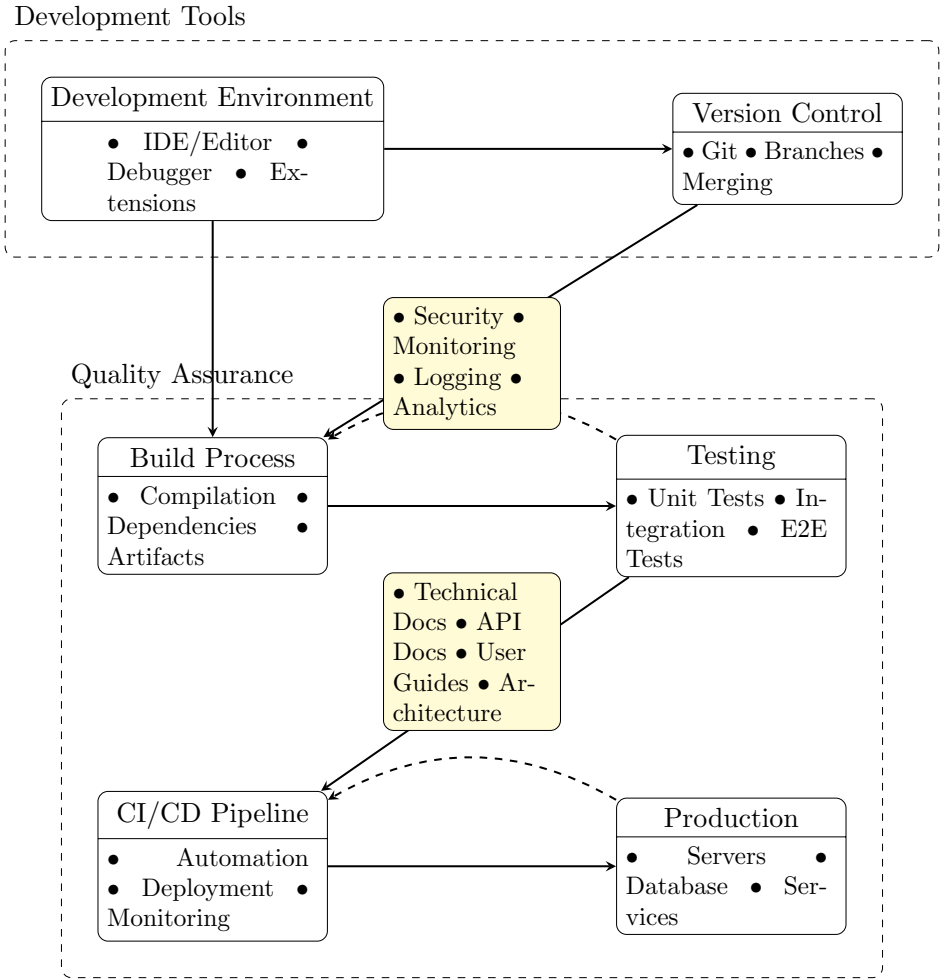
Software Design Process



Software Development

The actual coding or programming of the software system.
Key Aspects:
- Programming Languages: Choosing the appropriate language(s) based on the project (e.g., Python, Java, C++).
- Coding Standards: Adhering to guidelines for readability, maintainability, and consistency.
- Version Control: Using tools like Git to manage changes in the codebase.
- Testing During Development: Writing unit tests to verify individual components.
Coding standards: Following consistent naming conventions, formatting, and documentation practices
Code organization: Proper modularization, separation of concerns, and package structure
Design patterns: Implementing proven solutions for common programming challenges
Code review: Regular peer reviews to maintain quality and share

Software Development



Software Testing

The process of evaluating software to detect and fix defects and ensure it meets the requirements.

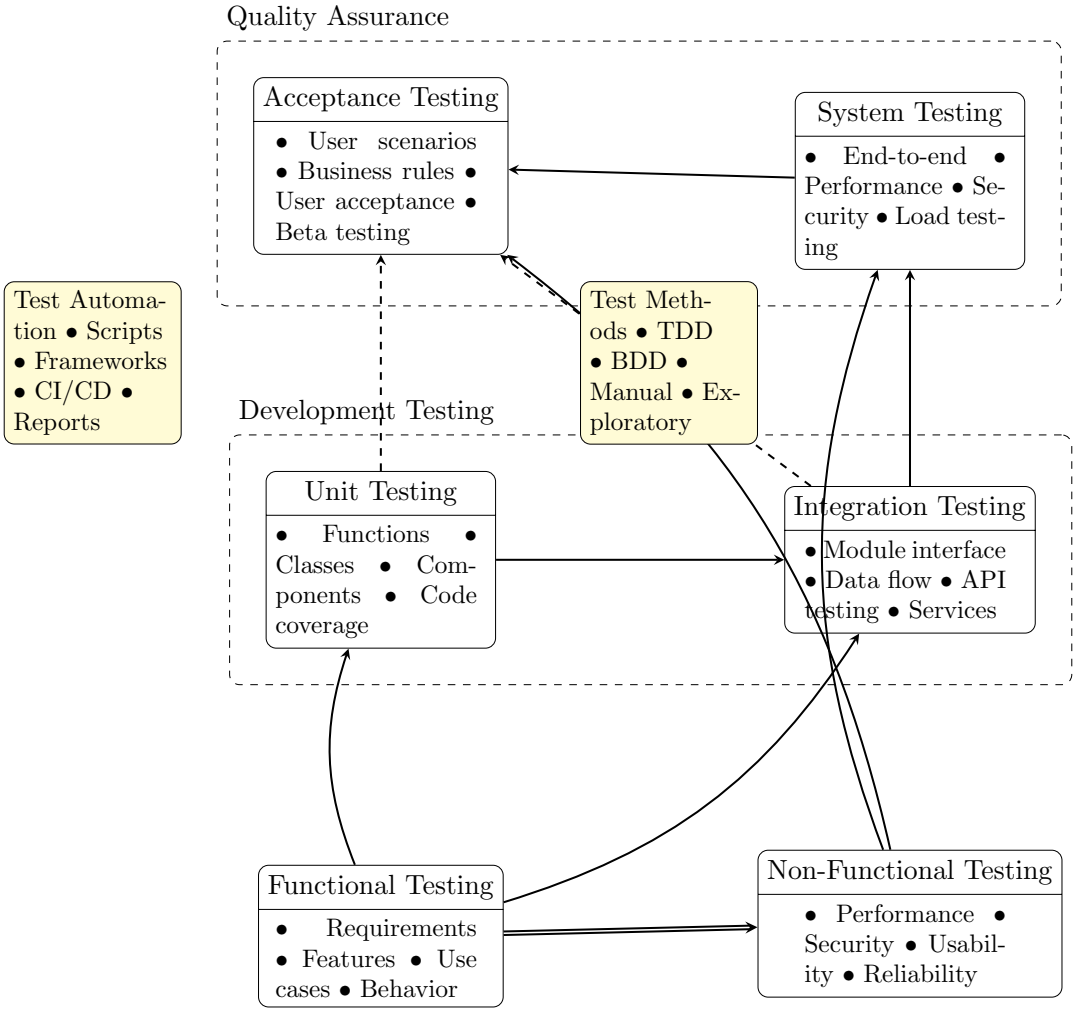
Types of Testing:

- Unit Testing: Testing individual components or functions.
- Integration Testing: Testing the interaction between components.
- System Testing: Testing the complete system as a whole.
- Acceptance Testing: Validating the system with end-users or stakeholders.

Testing Techniques:

- Black-Box Testing: Focuses on inputs and outputs without considering internal code.
- White-Box Testing: Tests internal structures or workings of the code.
- Performance testing: Load testing, stress testing, and scalability testing
- Security testing: Vulnerability scanning, penetration testing
- Automation Testing: Using tools like Selenium, JUnit, or TestNG to automate repetitive tests.

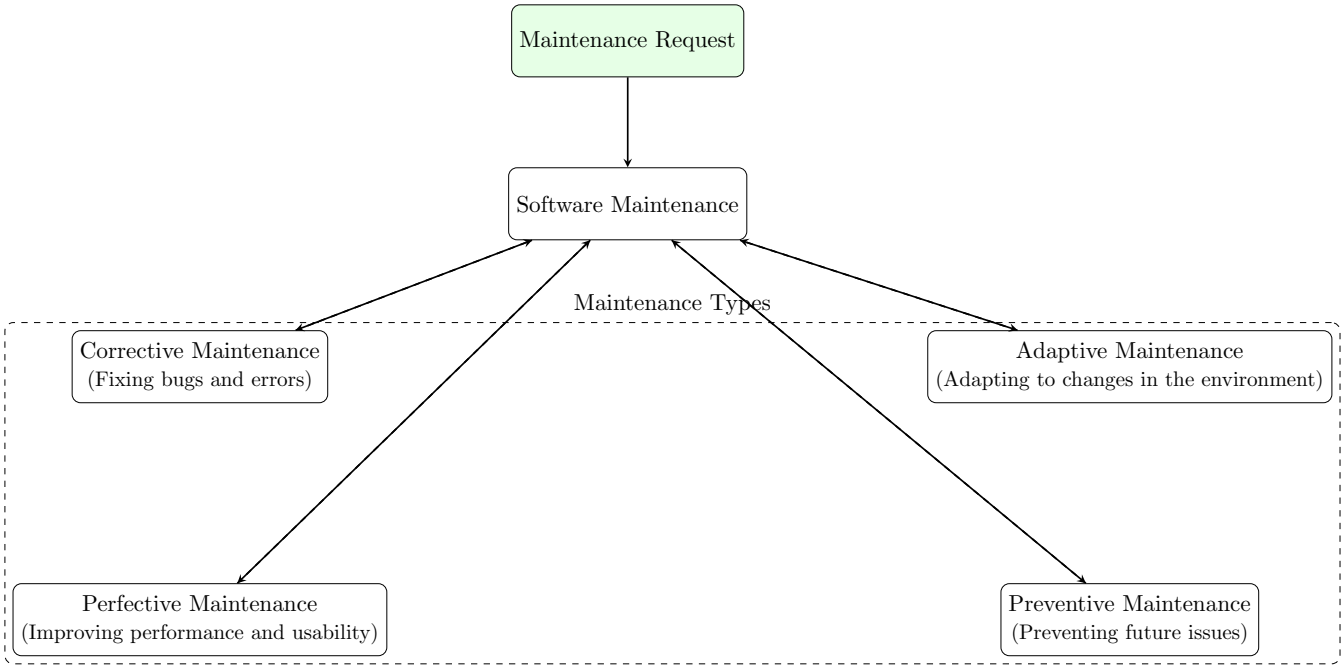
Software Testing

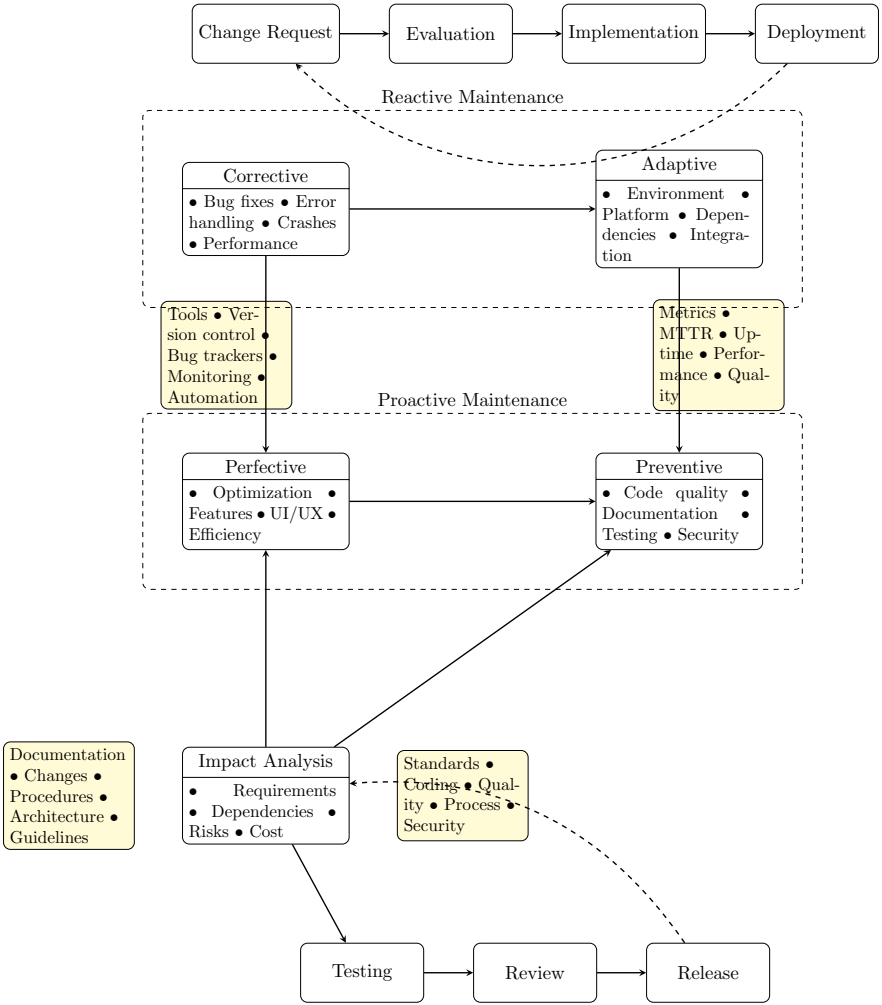


Software Maintenance

The process of modifying software after delivery to correct issues, improve performance, or adapt to changes.
Types of Maintenance:
- Corrective Maintenance: Fixing bugs or defects.
- Adaptive Maintenance: Updating software to work in new environments.
- Perfective Maintenance: Enhancing functionality or performance.
- Preventive Maintenance: Making changes to prevent future issues.

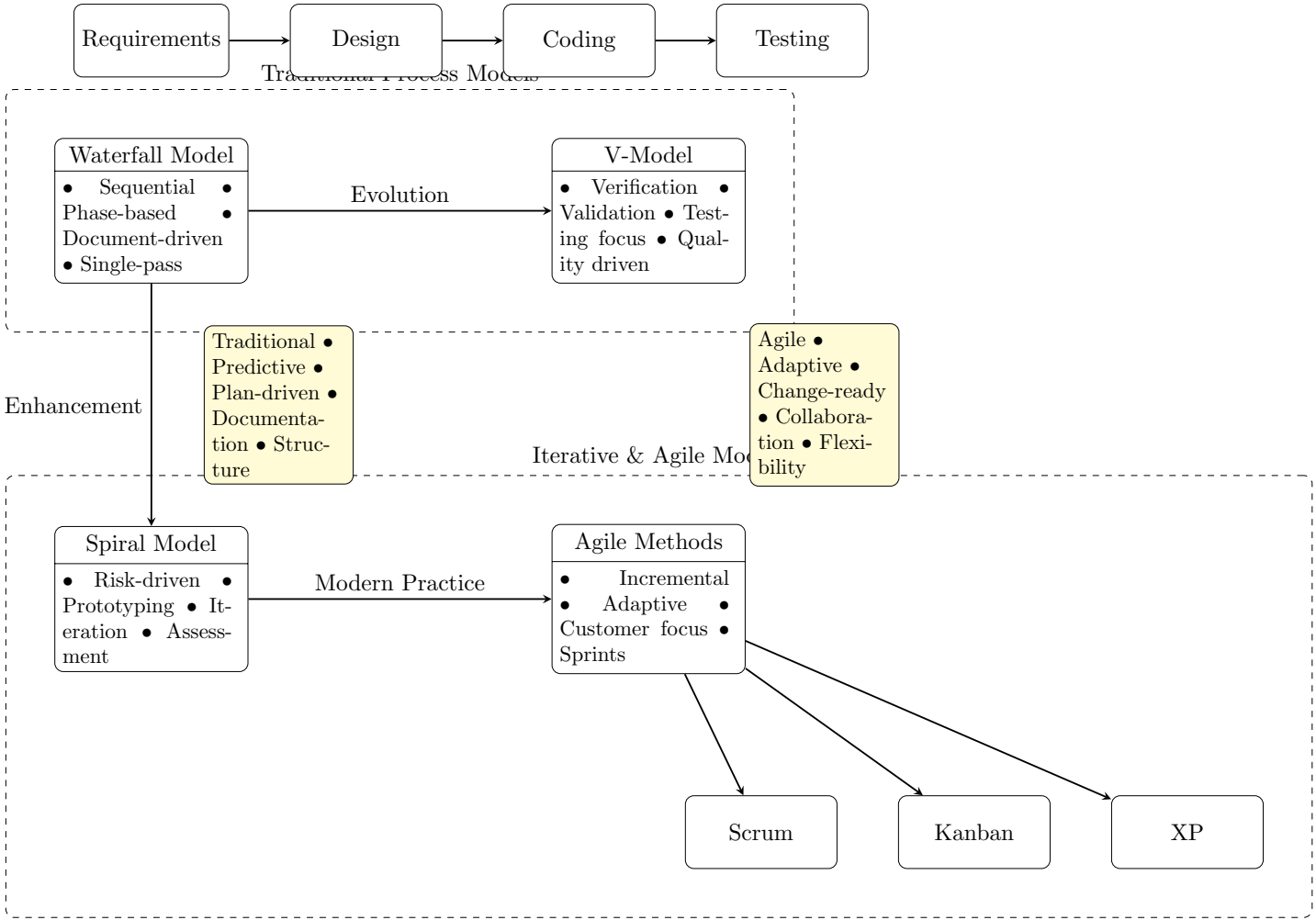
Software Maintenance Process




Change Request
Evaluation
Implementation
Deployment
Reactive Maintenance
Corrective
• Bug fixes • Error handling • Crashes • Performance
Adaptive
• Environment • Platform • Dependencies • Integration
Tools • Version control • Bug trackers • Monitoring • Automation
Metrics • MTTR • Uptime • Performance • Quality
Proactive Maintenance
Perfective
• Optimization • Features • UI/UX • Efficiency
Preventive
• Code quality • Documentation • Testing • Security
Documentation • Changes • Procedures • Architecture • Guidelines
Impact Analysis
• Requirements • Dependencies • Risks • Cost
Standards • Coding • Quality • Process • Security
Testing
Review
Release

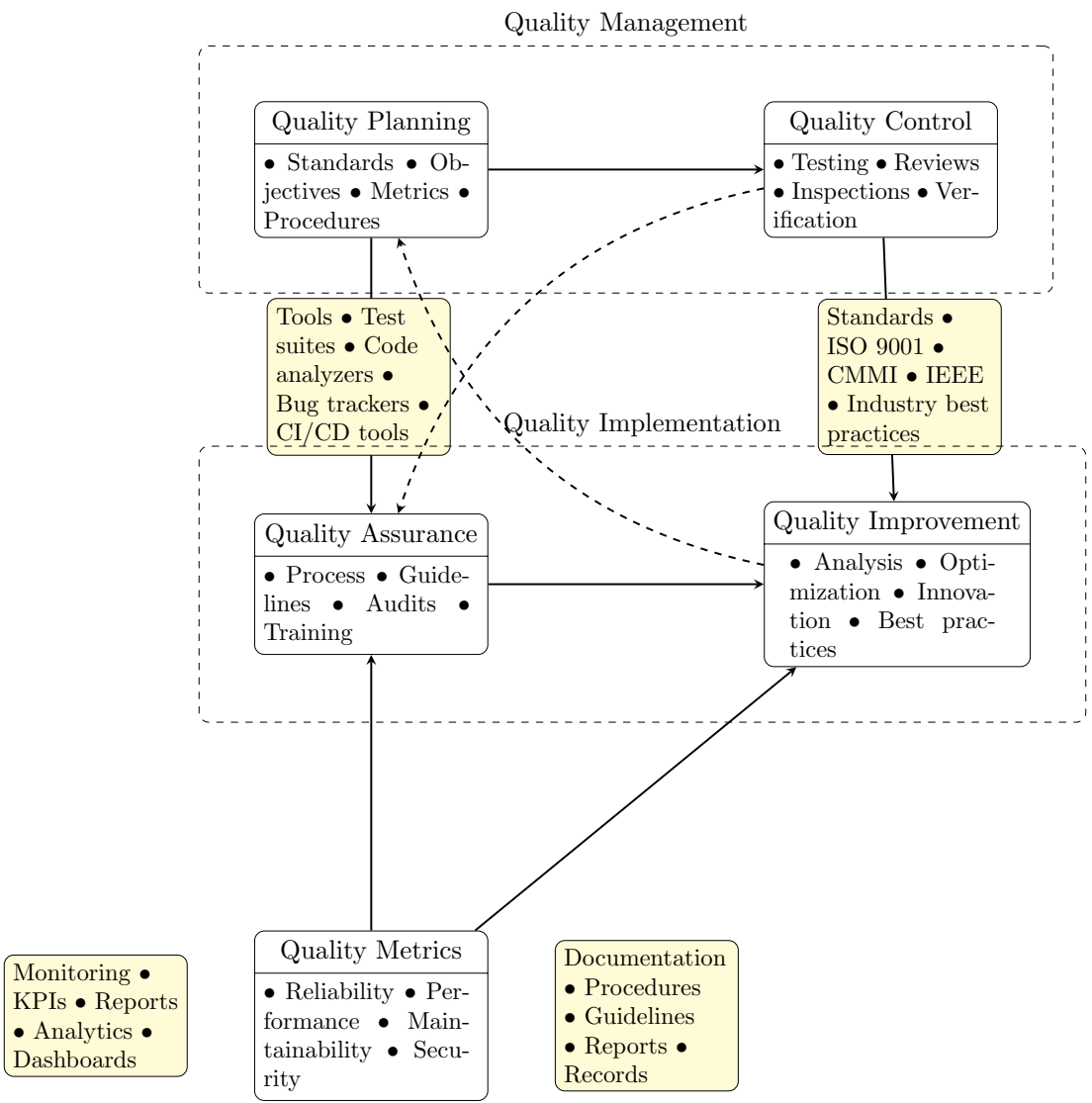
Software Process Models

Frameworks that define the steps involved in software development.
Common Models:
- Waterfall Model: Sequential phases (e.g., requirements → design → implementation → testing).
- Agile Model: Iterative and incremental approach focusing on flexibility and collaboration.
- Scrum: A subset of Agile with sprints and defined roles (Scrum Master, Product Owner).
- DevOps: Combines development and operations for continuous delivery and integration (CD/CI).
- Spiral Model: Combines iterative development with risk assessment.


Requirements
Design
Coding
Testing
Traditional Process Models
Waterfall Model
• Sequential • Phase-based • Document-driven • Single-pass
Evolution
V-Model
• Verification • Validation • Testing focus • Quality driven
Enhancement
Traditional • Predictive • Plan-driven • Documentation • Structure
Agile • Adaptive • Change-ready • Collaboration • Flexibility
Iterative & Agile Mo
Spiral Model
• Risk-driven • Prototyping • Iteration • Assessment
Modern Practice
Agile Methods
• Incremental • Adaptive • Customer focus • Sprints
Scrum
Kanban
XP

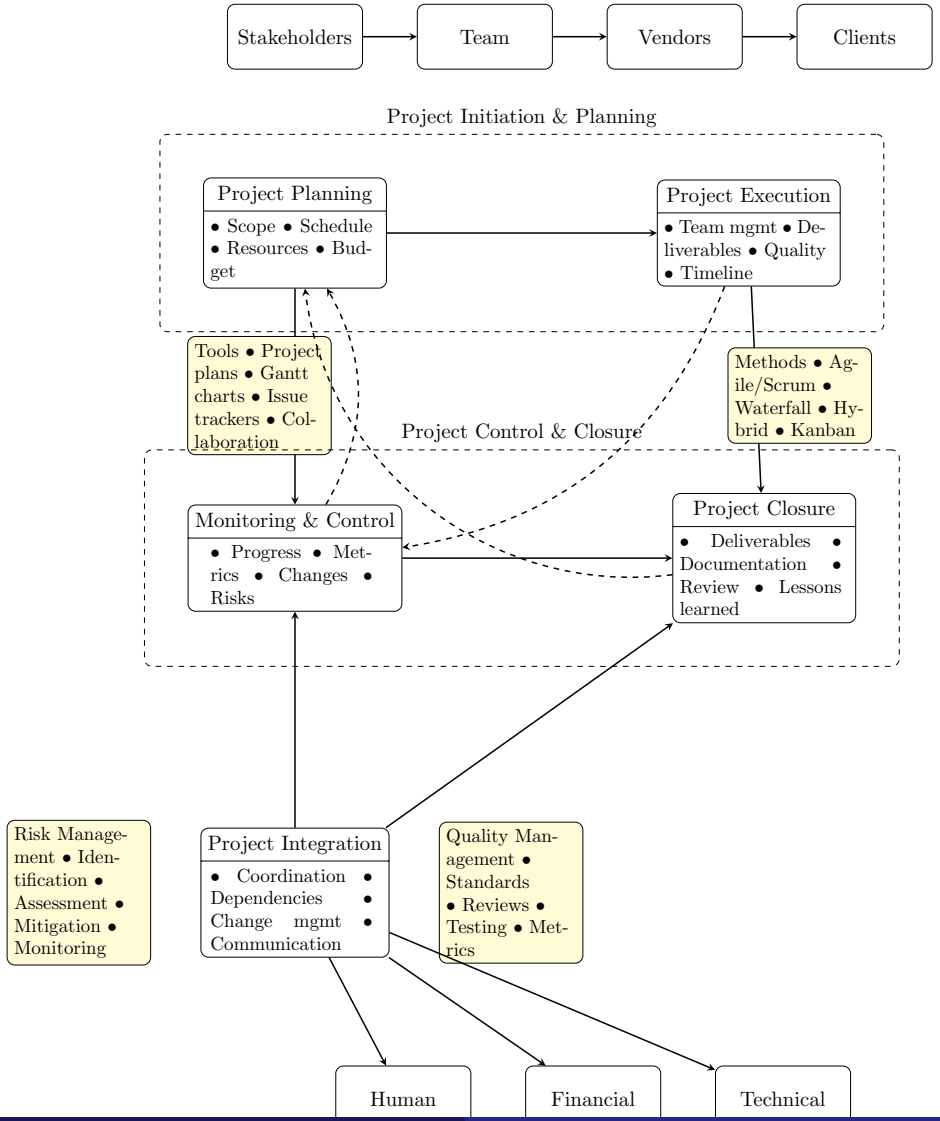
Software Quality Assurance (SQA)

Ensuring that the software development process and final product meet quality standards.
Key Activities:
- Process Audits: Reviewing the adherence to defined processes.
- Code Reviews: Peer review of code for quality and compliance.
- Metrics Collection: Monitoring performance, defect density, and other quality metrics.
- Standards Compliance: Adhering to industry standards like ISO 9001 or CMMI.


Quality Management
Quality Planning
• Standards • Objectives • Metrics • Procedures
Quality Control
• Testing • Reviews • Inspections • Verification
Tools • Test suites • Code analyzers • Bug trackers • CI/CD tools
Standards • ISO 9001 • CMMI • IEEE • Industry best practices
Quality Implementation
Quality Assurance
• Process • Guidelines • Audits • Training
Quality Improvement
• Analysis • Optimization • Innovation • Best practices
Quality Metrics
• Reliability • Performance • Maintainability • Security
Monitoring • KPIs • Reports • Analytics • Dashboards
Documentation • Procedures • Guidelines • Reports • Records

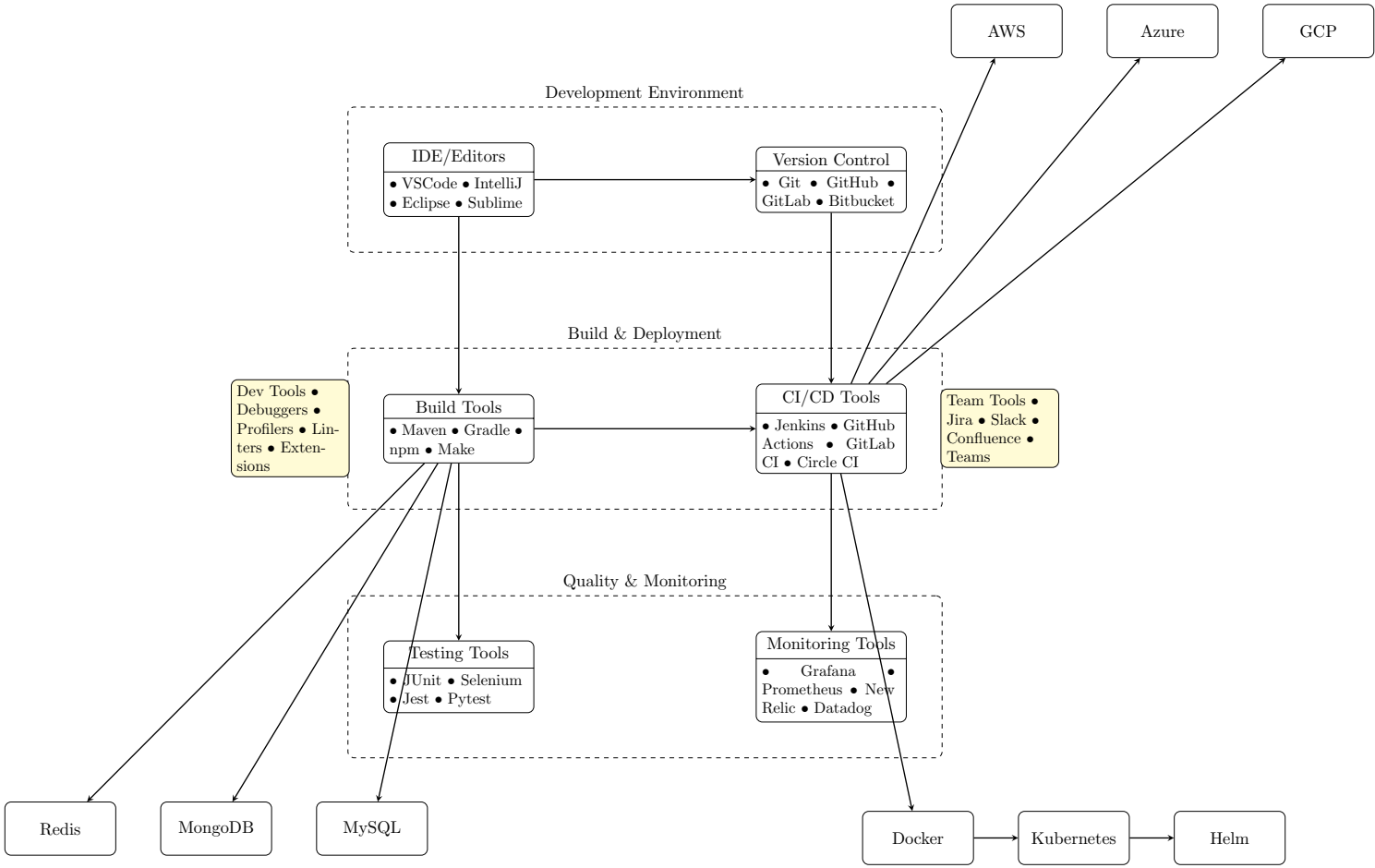
Project Management

Planning, executing, and monitoring software projects to meet goals within time, budget, and scope constraints.
Key Components:
- Scope Management: Defining and controlling what is included in the project.
- Time Management: Creating schedules and meeting deadlines.
- Cost Management: Budgeting and controlling expenses.
- Risk Management: Identifying and mitigating potential risks.
- Resource Management: Allocating and managing team members and tools.


Stakeholders
Team
Vendors
Clients
Project Initiation & Planning
Project Planning
• Scope • Schedule • Resources • Budget
Project Execution
• Team mgmt • Deliverables • Quality • Timeline
Tools • Project plans • Gantt charts • Issue trackers • Collaboration
Methods • Agile/Scrum • Waterfall • Hybrid • Kanban
Project Control & Closure
Monitoring & Control
• Progress • Metrics • Changes • Risks
Project Closure
• Deliverables • Documentation • Review • Lessons learned
Risk Management • Identification • Assessment • Mitigation • Monitoring
Project Integration
• Coordination • Dependencies • Change mgmt • Communication
Quality Management • Standards • Reviews • Testing • Metrics
Human
Financial
Technical

Software Tools and Environments

Essential tooling for modern software development:
- Development environments: Integrated Development Environments (IDEs), code editors, debugging tools. Tools like Eclipse, Visual Studio, or IntelliJ IDEA.
- Build tools: Compilers, build automation, package managers (e.g., Maven, Gradle).
- Version control systems: Git, SVN, and related tools
- CI/CD tools: Jenkins, GitLab CI, GitHub Actions
- Containerization: Docker, Kubernetes
- Cloud infrastructure: AWS, Azure, Google Cloud
- Monitoring tools: Application performance monitoring, log aggregation
- Collaboration tools: JIRA, Confluence, Slack


AWS
Azure
GCP
Development Environment
IDE/Editors
• VSCode • IntelliJ • Eclipse • Sublime
Version Control
• Git • GitHub • GitLab • Bitbucket
Build & Deployment
Dev Tools • Debuggers • Profilers • Linters • Extensions
Build Tools
• Maven • Gradle • npm • Make
CI/CD Tools
• Jenkins • GitHub Actions • GitLab CI • Circle CI
Team Tools • Jira • Slack • Confluence • Teams
Quality & Monitoring
Testing Tools
• JUnit • Selenium • Jest • Pytest
Monitoring Tools
• Grafana • Prometheus • New Relic • Datadog
Redis
MongoDB
MySQL
Docker
Kubernetes
Helm

Ethics in Software Engineering

Adhering to ethical standards in software development.
Key Principles:
- Public Interest: Ensuring software benefits society and avoids harm.
- Confidentiality: Respecting user privacy and data security.
- Professionalism: Delivering high-quality work and avoiding conflicts of interest.
- Accountability: Taking responsibility for software failures or defects.

Emerging Trends in Software Engineering

- Artificial Intelligence and Machine Learning: Incorporating AI into development and testing.
- Cloud Computing: Developing scalable applications on platforms like AWS or Azure.
- Microservices Architecture: Designing systems with loosely coupled services.
- Blockchain: Building secure and decentralized applications.
- Low-Code/No-Code Platforms: Enabling faster development with minimal coding.