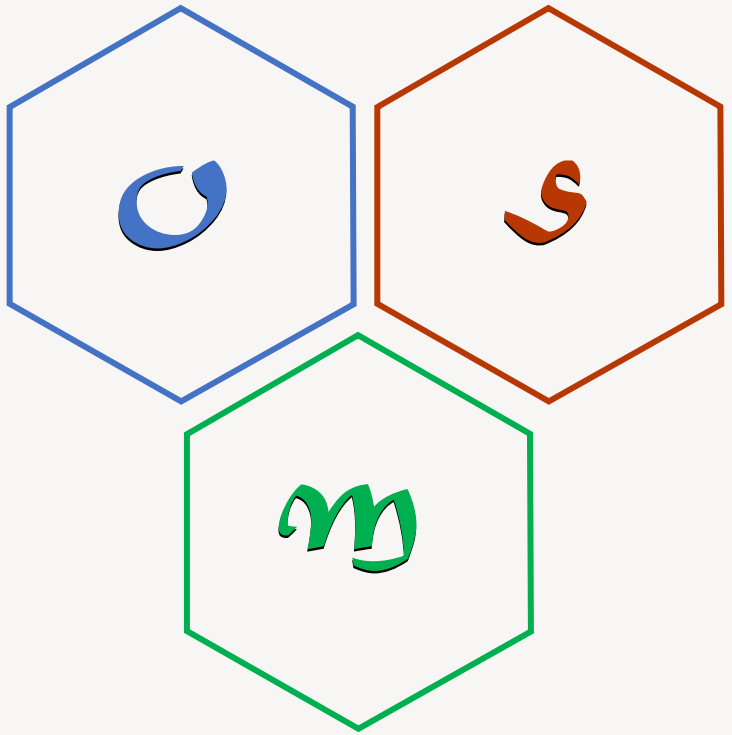
Principles of Applied Software Engineering

Informations Software Engineering

Prakriti Jain

Date : 26-02-2024



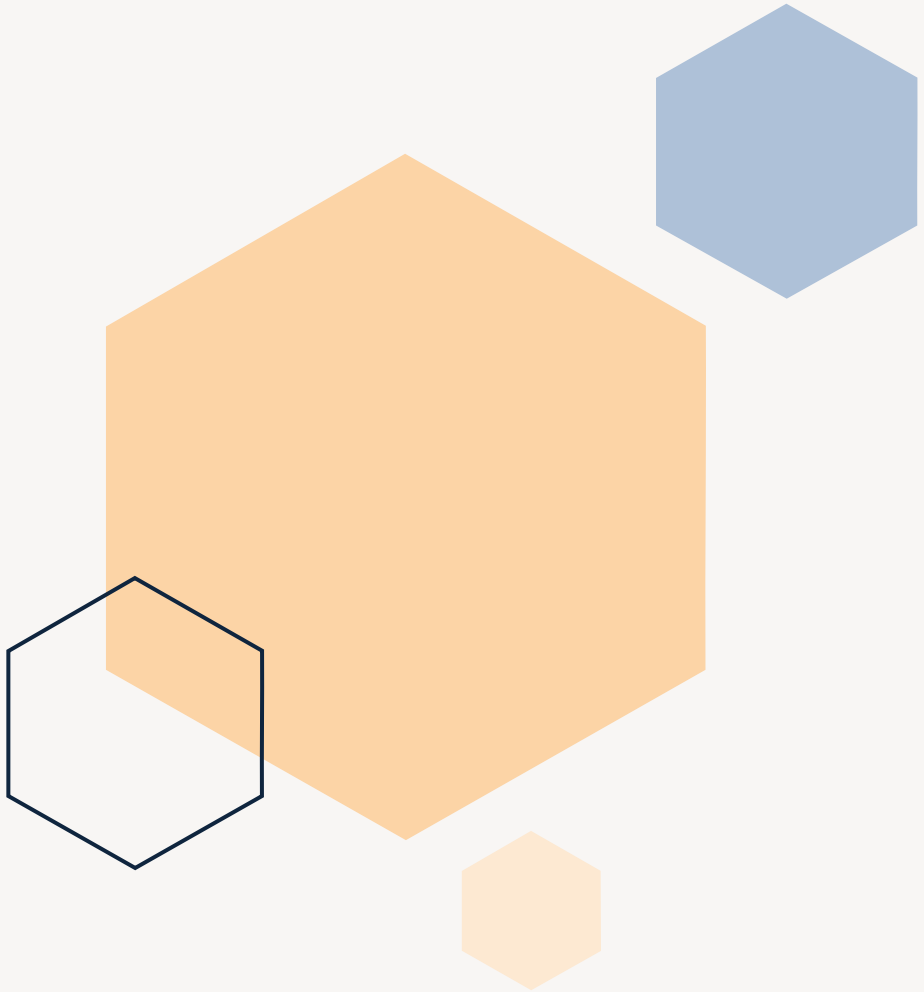
Agenda

Introduction

Principles

Description

Conclusion



Introduction

Principles of Applied Software Engineering are fundamental practices that enhances the quality, maintainability, and efficiency of software development processes and outcomes.

Principles

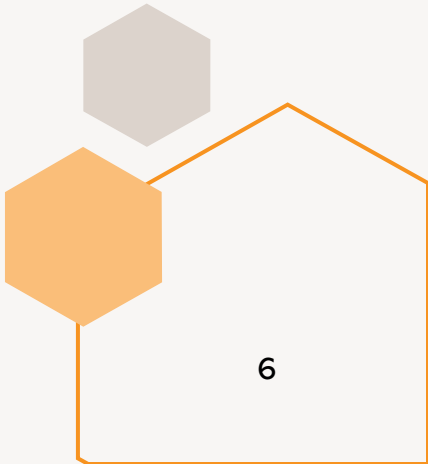
- Text-Based
- Feedback-oriented
- Versioned
- Automated
- Test-Centered
- Traceable
- Always Integrated
- Quality-Focused

Text-Based

- Utilizing text formats for all artifacts ensures they are stored in a human-readable and machine-readable format
- It facilitates version control systems like Git, enabling developers to track changes, compare versions, and collaborate effectively
- Text-based formats, such as Markdown for documentation and source code files, provide a standardized way to represent information
- Example :- PlantUML

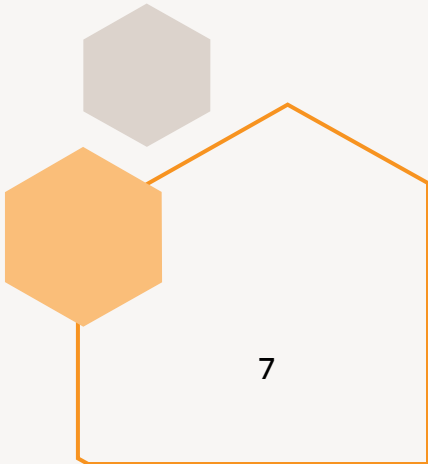
Versioned

- Version control systems like Git track changes to files over time
- Version control helps manage conflicts, revert changes if needed, and maintain a detailed history of project evolution
- We adhere to a single-branch workflow. Upon completing a task, we integrate the changes directly into the main codebase using rebasing once they are finalized
- Example – Git hub



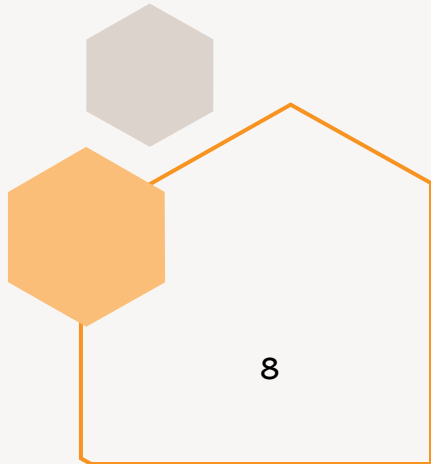
Test-Centered

- Developer-testing, including unit tests, integration tests, and end-to-end tests, ensures software behaves as expected
- Test-driven development (TDD) involves writing tests before implementing features, promoting better design and code quality
- Automation of testing ensures tests are executed automatically during the build process, providing fast feedback to developers
- All Tests are versioned
- Test Pyramid :-
  - -> Unit
  - -> Performance
  - -> System/e2e
  - -> Integration



Always Integrated

- Trunk-based development encourages frequent integration of code changes into the main branch
- Continuous integration (CI) involves automatically building and testing code changes whenever they are pushed to the repository
- A continuously releasable main branch ensures the codebase is stable and ready for deployment at any time



Feedback-Oriented

- Fast feedback loops, including build feedback and user feedback, help identify issues early in the development process
- Fast integration feedback through trunk-based development
- Continuous deployment allows for frequent releases, enabling rapid validation of changes in a real-world environment
- Monitoring user behavior and application performance provides valuable insights for further improvements

Automated

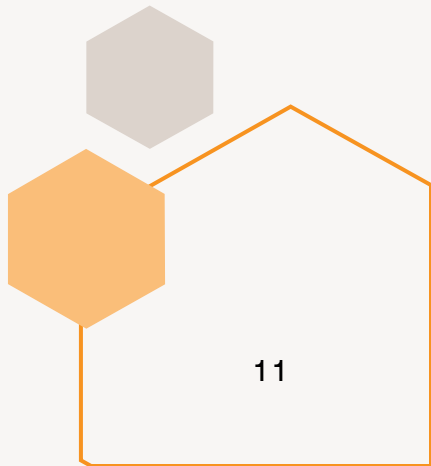
- Automation streamlines repetitive tasks such as testing, deployment, and infrastructure provisioning

- Automatic source code quality checks : -

  -> Method length (17)
  -> Cyclomatic complexity (4)
  -> Test coverage (80% per module)

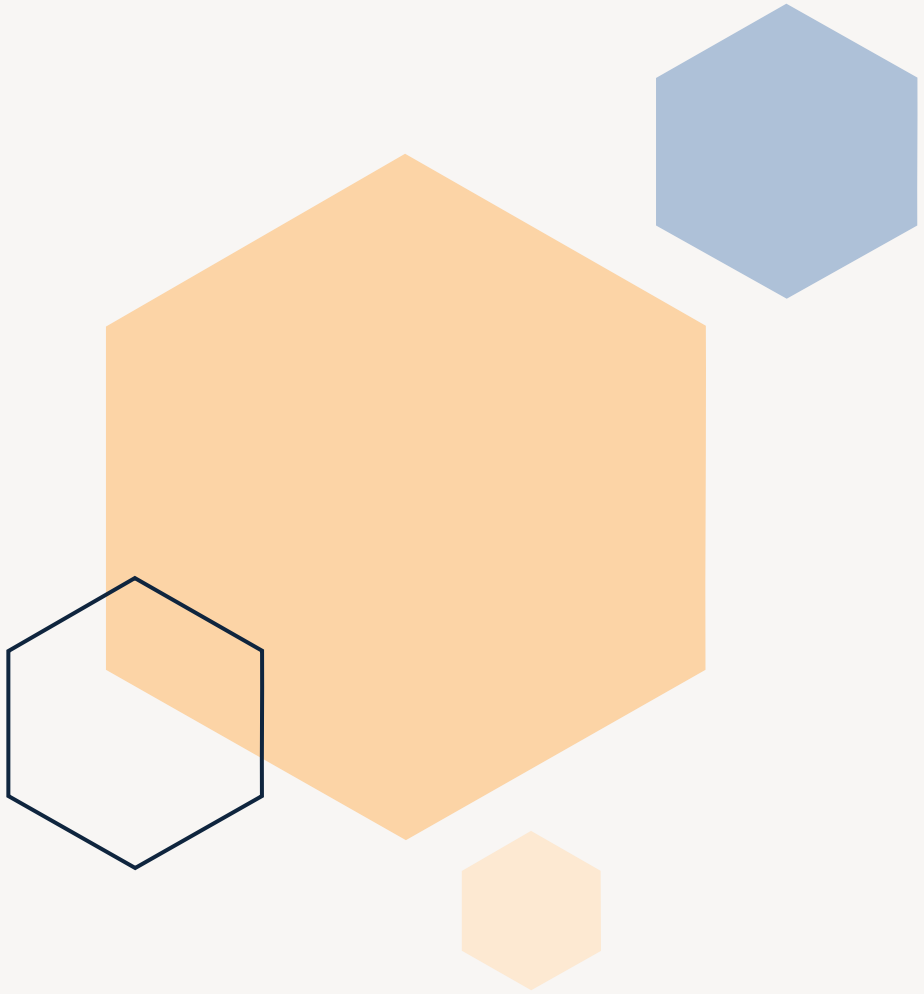
Traceable

- Traceability ensures that artifacts, such as code changes and documentation, are linked and can be traced back to their origin
- Version control systems provide a detailed history of changes, allowing developers to understand why and how code evolved over time
- Architecture Decision Records (ADRs) document important design decisions, providing context for future development



Quality-Focused

- Structural integrity and readability are essential for maintaining code quality and facilitating collaboration among team members
- Continuous refactoring improves code maintainability and reduces technical backlogs
- Code quality metrics, such as method length, cyclomatic complexity, and test coverage, provide objective measures of code quality



Conclusion

The principles of applied software engineering provide a solid foundation for developing high-quality software

Thank you