Claude Code Best Practices: A Four-Phase Development Framework

Introduction

- Overview of Claude Code as an agentic command line tool
- Benefits of structured development workflow
- Introduction to the Assess → Plan → Execute → Test framework

Phase 1: Assess

Purpose

- Understand the current state of your project
- Identify requirements and constraints
- Set clear objectives for the development session

Key Activities

- **Project Analysis**
  - Review existing codebase structure
  - Identify technical debt and areas for improvement
  - Assess dependencies and compatibility requirements
- **Requirement Gathering**
  - Define functional requirements clearly
  - Identify non-functional requirements (performance, security, etc.)
  - Document expected inputs and outputs
- **Resource Assessment**
  - Evaluate available development time
  - Identify knowledge gaps that need addressing
  - Assess testing requirements and environments

Best Practices

- Use descriptive prompts that provide context about your project
- Share relevant code snippets or file structures with Claude
- Be explicit about your technical stack and constraints

- Document assumptions and requirements before proceeding

Development Process Improvements

- Reduces time spent on misaligned solutions
- Minimizes back-and-forth iterations
- Establishes clear success criteria upfront
- Helps Claude provide more targeted assistance

Phase 2: Plan

Purpose

- Create a structured approach to implementation
- Break down complex tasks into manageable components
- Establish a clear roadmap for development

Key Activities

- **Task Decomposition**
  - Break large features into smaller, testable components
  - Identify dependencies between tasks
  - Prioritize implementation order
- **Architecture Planning**
  - Design overall system structure
  - Define interfaces and data flows
  - Plan for scalability and maintainability
- **Implementation Strategy**
  - Choose appropriate design patterns
  - Plan error handling and edge cases
  - Define coding standards and conventions

Best Practices

- Create step-by-step implementation plans
- Use Claude to validate architectural decisions
- Plan for incremental development and testing
- Document design decisions and rationale

Development Process Improvements

- Provides clear direction and reduces decision fatigue
- Enables parallel development of independent components
- Facilitates better code organization and maintainability
- Reduces risk of architectural mistakes

Phase 3: Execute

Purpose

- Implement the planned solution efficiently
- Maintain code quality throughout development
- Leverage Claude's coding capabilities effectively

Key Activities

- **Code Generation**
  - Request specific functions or modules
  - Implement complex algorithms with Claude's assistance
  - Generate boilerplate code and templates
- **Code Review and Refinement**
  - Iterate on generated code for optimization
  - Ensure adherence to coding standards
  - Integrate new code with existing codebase
- **Documentation Creation**
  - Generate inline comments and docstrings
  - Create README files and API documentation
  - Document configuration and setup procedures

Best Practices

- Provide clear, specific prompts for code generation
- Review and understand all generated code before integration
- Maintain consistent coding style across the project
- Use Claude for code explanation and learning

Development Process Improvements

- Accelerates development velocity
- Reduces boilerplate coding time
- Provides learning opportunities through code explanation
- Maintains consistency in code quality

Phase 4: Test

Purpose

- Validate implementation against requirements
- Identify and resolve issues early
- Ensure code reliability and maintainability

Key Activities

- **Test Planning**
  - Define test strategies and coverage requirements
  - Identify test cases and scenarios
  - Plan integration and system testing
- **Test Implementation**
  - Generate unit tests for individual components
  - Create integration tests for system interactions
  - Implement automated testing pipelines
- **Issue Resolution**
  - Debug failing tests and identify root causes
  - Refactor code to improve testability
  - Validate fixes and regression testing

Best Practices

- Use Claude to generate comprehensive test suites
- Implement tests incrementally during development
- Focus on edge cases and error conditions
- Maintain test documentation and rationale

Development Process Improvements

- Increases confidence in code reliability
- Reduces debugging time in production
- Facilitates refactoring and code maintenance
- Provides regression protection for future changes

Leveraging Artifacts in Claude Code Workflows

What Are Artifacts?

- Self-contained pieces of content (code, documentation, configurations)
- Persistent across conversation sessions
- Can be iteratively updated and refined
- Ideal for complex, substantial content

Best Practices for Artifacts

- **Code Artifacts**
  - Use for complete modules, classes, or substantial functions
  - Maintain version control awareness
  - Include comprehensive documentation
- **Documentation Artifacts**
  - Create living documentation that evolves with the project
  - Include setup instructions, API references, and examples
  - Maintain consistency with codebase changes
- **Configuration Artifacts**
  - Store configuration files and templates
  - Document configuration options and their purposes
  - Version control integration strategies

Integration with Development Phases

- **Assess Phase**: Use artifacts to document requirements and constraints
- **Plan Phase**: Create architectural diagrams and implementation plans
- **Execute Phase**: Generate and refine code artifacts
- **Test Phase**: Maintain test documentation and coverage reports

Using Markdown Files for Activity Direction and Tracking

Project Organization

- **Master Plan Files**
  - Overall project roadmap and milestones
  - Task prioritization and dependencies
  - Progress tracking and status updates
- **Phase-Specific Documentation**
  - Detailed requirements (assess.md)
  - Implementation plans (plan.md)
  - Execution logs (execute.md)
  - Test results and reports (test.md)

Activity Direction

- **Task Templates**
  - Standardized formats for common development tasks
  - Checklists for quality assurance
  - Guidelines for code review and testing
- **Decision Logs**
  - Document architectural decisions and rationale
  - Track changes and their impact
  - Maintain historical context for future reference

Progress Tracking

- **Status Dashboards**
  - Visual representation of project progress
  - Milestone completion tracking
  - Issue and risk identification
- **Retrospective Documentation**
  - Lessons learned and best practices
  - Process improvements and refinements
  - Knowledge transfer and team learning

Best Practices for Markdown Integration

- Use consistent formatting and structure
- Maintain clear linking between related documents
- Regular updates and synchronization with development progress
- Integration with version control systems

Implementation Recommendations

Getting Started

1. Set up a structured directory for your Claude Code projects
2. Create template markdown files for each phase
3. Establish naming conventions and organization standards
4. Define your project's specific workflow adaptations

Workflow Integration

- Begin each development session with assessment documentation
- Use artifacts for substantial code and documentation
- Maintain real-time updates to tracking documents
- Regular reviews and retrospectives for process improvement

Team Collaboration

- Share markdown templates and standards across team members
- Use collaborative documentation platforms
- Establish review processes for artifacts and documentation
- Regular knowledge sharing sessions

Conclusion

- Recap of the four-phase framework benefits
- Emphasis on iterative improvement and learning
- Encouragement to adapt the framework to specific project needs
- Future considerations for scaling and team adoption

Q&A and Discussion Points

- Common challenges and solutions
- Customization strategies for different project types
- Integration with existing development workflows
- Measuring success and continuous improvement