BMAD Method - Complete Guide

Universal AI Agent Framework for Software Development

Version: 4.44.1 **Generated**: October 21, 2025 **Project**: Fleet Management V2 **Author**: Maurice (Skycruzer)

Table of Contents

Introduction to BMAD

What is BMAD?

BMAD-METHOD™ (Breakthrough Method of Agile AI-driven Development) is a framework that combines AI agents with Agile development methodologies. It transforms you into a "Vibe CEO" - directing a team of specialized AI agents through structured workflows.

Key Principles

1. **Agent Specialization** - Each agent has specific expertise and responsibilities
2. **Clean Handoffs** - Always start fresh when switching between agents
3. **Status Tracking** - Maintain story statuses (Draft → Approved → InProgress → Done)
4. **Iterative Development** - Complete one story before starting the next
5. **Documentation First** - Always start with solid PRD and architecture

When to Use BMAD

- **New Projects (Greenfield)**: Complete end-to-end development
- **Existing Projects (Brownfield)**: Feature additions and enhancements
- **Team Collaboration**: Multiple roles working together
- **Quality Assurance**: Structured testing and validation
- **Documentation**: Professional PRDs, architecture docs, user stories

---

The BMAD Agents

Core Development Team

Agent	Role	Primary Functions	When to Use
analyst (Mary)	Business Analyst	Market research, competitive analysis, brainstorming,	Project planning, requirements gathering, brownfield

		documenting existing systems	documentation
pm	Product Manager	PRD creation, feature prioritization, epic management	Strategic planning, roadmaps, requirement definition
architect	Solution Architect	System design, technical architecture, technology decisions	Complex systems, scalability planning, technical design
ux-expert	UX Designer	UI/UX design, prototypes, frontend specifications	User experience, interface design, design systems
po	Product Owner	Backlog management, story validation, document sharding	Story refinement, document alignment, quality checks
sm	Scrum Master	Sprint planning, story creation from epics	Story drafting, task breakdown, sprint management
dev	Developer	Code implementation, debugging, testing	All development tasks, feature implementation
qa (Quinn)	Test Architect	Test strategy, quality gates, code review	Testing, quality assurance, risk assessment

Meta Agents

Agent	Role	Primary Functions	When to Use
bmad-orchestrator	Team Coordinator	Multi-agent workflows, role switching	Complex multi-role tasks (Web UI only)
bmad-master	Universal Expert	All capabilities without switching	Single-session comprehensive work

Agent Interaction in Claude Code

Invoke agents using slash commands:

```
/analyst - Business analysis and documentation
/pm - Product management and PRD creation
/architect - System architecture design
/ux-expert - UI/UX specifications
/po - Product owner tasks
/sm - Story creation
/dev - Code implementation
/qa - Quality assurance
/bmad-master - Universal agent
```

Always start a NEW CHAT when switching agents - Fresh context = better results!

The Two-Phase Workflow

Phase 1: Planning (Web UI or IDE)

Best for: Using large context windows (Gemini/ChatGPT/Claude)

Activities:

- Market research and competitive analysis
- PRD creation with epics and stories
- System architecture design
- UX/UI specifications
- Document validation and alignment

Artifacts Created:

- `docs/prd.md` - Product Requirements Document
- `docs/architecture.md` - System Architecture
- Optional: Market research, competitor analysis, UX specs

Phase 2: Development (IDE Required)

Best for: Active coding and implementation

Activities:

- Document sharding (PRD → Epics, Architecture → Components)
- Story creation from epics
- Code implementation
- Testing and quality assurance
- Code review and refactoring

Artifacts Created:

- `docs/prd/epic-1.md` , `epic-2.md` , etc.
- `docs/architecture/` - Sharded architecture files
- `docs/stories/` - User stories
- `docs/qa/` - Quality assessments and gates

Planning Phase Workflow

Step 1: Optional Analysis (Analyst Agent)

For brownfield projects - START HERE!

```
/analyst
*document-project
```

The analyst will:

- Analyze your existing codebase
- Create comprehensive system documentation
- Identify architecture patterns
- Document database schema
- Map component structure

Output: Comprehensive brownfield documentation

Step 2: Project Brief (Optional)

Create a foundation document describing your project vision, goals, and constraints.

Step 3: PRD Creation (PM Agent)

For new projects:

```
/pm
*create-doc prd
```

For brownfield/existing projects:

```
/pm
*create-doc brownfield-prd
```

The PM will:

- Gather requirements through structured questions
- Define functional and non-functional requirements
- Create epics with initial stories
- Establish success metrics
- Define technical constraints

Output: `docs/prd.md`

Step 4: Architecture Design (Architect Agent)

```
/architect
*create-doc architecture
```

For brownfield:

```
/architect
*create-doc brownfield-architecture
```

The architect will:

- Design system architecture
- Define technology stack
- Establish coding standards
- Create component structure
- Plan data models and API contracts

Output: `docs/architecture.md`

Step 5: UX Design (Optional - UX Expert Agent)

```
/ux-expert
*create-doc front-end-spec
```

The UX expert will:

- Create UI/UX specifications
- Define component library
- Establish design system

- Plan user flows
- Create wireframes/mockups

Output: Frontend specification document

Step 6: Validation (PO Agent)

```
/po
*execute-checklist po-master-checklist
```

The PO will:

- Validate document alignment
- Check epic/story consistency
- Verify architecture matches PRD
- Ensure all requirements covered
- Identify gaps or conflicts

Output: Validation report with action items

Development Phase Workflow

CRITICAL: Always Use Fresh Chat Windows

Rule: Start a NEW CHAT for each agent (SM → Dev → QA)

Step 1: Document Sharding (One-Time Setup)

CRITICAL STEP: Documents must be sharded before development!

```
/po
Please shard docs/prd.md
Please shard docs/architecture.md
```

Creates:

- `docs/prd/epic-1.md` , `epic-2.md` , etc. (with stories in development order)
- `docs/architecture/coding-standards.md` , `tech-stack.md` , etc.

Step 2: Story Creation (SM Agent - New Chat)

ALWAYS use a fresh chat window:

```
/sm
*create
```

The SM will:

- Read next story from sharded epic
- Enrich with architecture details
- Create detailed tasks and acceptance criteria
- Add technical implementation notes
- Create story file in `docs/stories/`
- Set status to "Draft"

You: Review and change status to "Approved"

Step 3: Story Implementation (Dev Agent - New Chat)

ALWAYS start new chat:

```
/dev
Please implement story: docs/stories/[story-file]
```

Provide the story file content to save the dev agent lookup time.

The dev will:

- Load coding standards and tech stack
- Implement tasks sequentially
- Write comprehensive tests (unit + integration + E2E)
- Mark tasks as completed
- Maintain file list of all changes
- Set status to "Ready for Review"

Step 4: Quality Assurance (QA Agent - New Chat)

ALWAYS start new chat:

```
/qa
*review docs/stories/[story-file]
```

The QA will:

- Review code architecture and patterns
- Validate test coverage at all levels
- Perform active refactoring (when safe)
- Check non-functional requirements
- Create quality gate assessment
- Update story with QA Results
- Set status to "Done" (if passed)

Step 5: Commit & Continue

```
git add .
git commit -m "feat: implement feature X"
git push
```

Then repeat steps 2-5 for the next story.

Quality Gates Explained

Status	Meaning	Can Proceed?
PASS	All critical requirements met	✅ Yes
CONCERNS	Non-critical issues found	⚠️ With caution
FAIL	Critical issues (security, missing P0 tests)	❌ No - must fix

WAIVED	Issues acknowledged and accepted	✅ With approval

Brownfield Development Guide

What is Brownfield Development?

Brownfield development refers to adding features, fixing bugs, or modernizing existing software projects. Unlike greenfield (new) projects, brownfield work requires understanding existing code, respecting constraints, and ensuring new changes integrate seamlessly.

When to Use Brownfield Approach

- Add significant new features to existing applications
- Modernize legacy codebases
- Integrate new technologies or services
- Refactor complex systems
- Fix bugs that require architectural understanding
- **Document undocumented systems**

Critical First Step: Document Existing System

MOST IMPORTANT: Before adding features, document what you have!

```
/analyst
*document-project
```

The analyst will:

- Ask for focus areas (if scope is large)
- Analyze your codebase systematically
- Document architecture patterns
- Map database schema
- Identify component structure
- Create comprehensive brownfield documentation

Output: Complete system documentation for AI agents

Brownfield Workflow Options

Option A: PRD-First (Recommended for Large Changes)

Best for: Multiple features, complex integrations, major enhancements

1. **Create Brownfield PRD**:

   ```
   /pm
   *create-doc brownfield-prd
   ```

2. **Document Relevant Areas**:

   ```
   /analyst
   *document-project
   ```

   (Analyst will focus on areas identified in PRD)

3. **Update Architecture**:

```
/architect
*create-doc brownfield-architecture
```

4. **Continue with standard development workflow**

Option B: Single Epic/Story (For Small Changes)

Best for: Single focused enhancement, isolated bug fixes

1. **Create Single Epic**:

```
/pm
*brownfield-create-epic
```

2. **Or Create Single Story**:

```
/sm
*brownfield-create-story
```

3. **Implement directly** with Dev agent

Key Brownfield Principles

1. **Documentation First** - Always run `*document-project` if docs are outdated/missing
2. **Context Matters** - Provide agents access to relevant code sections
3. **Integration Focus** - Emphasize compatibility and non-breaking changes
4. **Incremental Approach** - Plan for gradual rollout and testing
5. **Respect Constraints** - Work within existing architecture patterns

---

Agent Commands Reference

Analyst Agent Commands

- `*help` - Show available commands
- `*brainstorm {topic}` - Facilitate structured brainstorming
- `*create-competitor-analysis` - Create competitor analysis document
- `*create-project-brief` - Create project brief
- `*document-project` - **Document existing codebase (CRITICAL for brownfield)**
- `*perform-market-research` - Conduct market research
- `*research-prompt {topic}` - Create deep research prompt

PM Agent Commands

- `*help` - Show available commands
- `*create-doc prd` - Create PRD for new project
- `*create-doc brownfield-prd` - Create PRD for existing project
- `*yolo` - Toggle YOLO mode (rapid generation)

Architect Agent Commands

- `*help` - Show available commands
- `*create-doc architecture` - Create architecture for new project

- `*create-doc brownfield-architecture` - Create architecture for existing project
- `*document-project` - Document existing system architecture

UX Expert Agent Commands

- `*help` - Show available commands
- `*create-doc front-end-spec` - Create frontend specification
- `*generate-ui-prompt` - Generate prompt for Lovable/V0

PO Agent Commands

- `*help` - Show available commands
- `*execute-checklist po-master-checklist` - Validate document alignment
- `*shard-doc {file}` - Shard PRD or Architecture document
- `*validate-next-story` - Validate story against artifacts

SM Agent Commands

- `*help` - Show available commands
- `*create` - Create next story from epic
- `*status` - Show current progress

Dev Agent Commands

- `*help` - Show available commands
- `*develop-story {story}` - Implement story with checklist
- `*status` - Show current context

QA Agent Commands

- `*help` - Show available commands
- `*risk {story}` - Risk assessment (run BEFORE development)
- `*design {story}` - Test strategy design (run BEFORE development)
- `*trace {story}` - Requirements tracing (run DURING development)
- `*nfr {story}` - NFR validation (run DURING development)
- `*review {story}` - **Comprehensive review (REQUIRED after development)**
- `*gate {story}` - Update quality gate status

Quick Start Guide

For New Projects (Greenfield)

1. **Plan** (Web UI or IDE):

   ```
   /pm → Create PRD
   /architect → Create Architecture
   /po → Validate Alignment
   ```

2. **Prepare** (IDE):

   ```
   /po → Shard documents
   ```

3. **Develop** (IDE - Sequential):

```
/sm → Create story (NEW CHAT)
/dev → Implement story (NEW CHAT)
/qa → Review story (NEW CHAT)
Commit → Repeat
```

For Existing Projects (Brownfield)

1. **Document** (IDE):

```
/analyst → *document-project
```

2. **Plan Enhancement** (Web UI or IDE):

```
/pm → Create brownfield PRD
/architect → Update architecture
/po → Validate alignment
```

3. **Prepare** (IDE):

```
/po → Shard documents
```

4. **Develop** (Same as greenfield):

```
/sm → Create story
/dev → Implement
/qa → Review
```

Best Practices

✅ DO These Things

1. **Always start NEW CHAT** when switching agents (SM → Dev → QA)
2. **Always shard documents** before development starts
3. **Always use SM agent** for story creation (never bmad-master during dev)
4. **Always use Dev agent** for implementation (never bmad-master during dev)
5. **Always run QA review** before marking story done
6. **Work ONE story at a time** - sequential, not parallel
7. **Document brownfield systems** before adding features

❌ DON'T Do These Things

1. **Don't reuse chat windows** - fresh context = better results
2. **Don't skip sharding** - dev agents need lean context
3. **Don't use bmad-master for dev work** - use specialized agents
4. **Don't work on multiple stories** - focus on one at a time
5. **Don't skip QA review** - quality gates prevent regressions
6. **Don't add features without documentation** - brownfield needs context

Context Management

- **Use powerful models** for SM story creation (they plan best)
- **Keep dev context lean** - only load necessary files

- **Clean chat windows** - start fresh for each agent
- **Document as you go** - maintain architectural documentation

Git Workflow

Branch Strategy:

- `main` - Production-ready code
- `develop` - Integration branch
- `feature/*` - Feature branches
- `bugfix/*` - Bug fix branches

Commit Conventions:

```
feat: Add pilot certification upload
fix: Resolve date calculation in leave eligibility
docs: Update API documentation
refactor: Simplify pilot service logic
test: Add E2E tests for certification flow
```

Quality Assurance with QA Agent

The Test Architect (Quinn)

The QA agent is a **Test Architect** with deep expertise in test strategy, quality gates, and risk-based testing. It provides advisory authority on quality matters while actively improving code when safe to do so.

QA Commands Throughout Workflow

Stage	Command	When	Purpose
Story Drafting	`*risk`	After SM drafts story	Identify pitfalls early
	`*design`	After risk assessment	Guide dev on test strategy
Development	`*trace`	Mid-implementation	Verify test coverage
	`*nfr`	While building	Catch quality issues early
Review	`*review`	Story marked complete	Full quality assessment
Post-Review	`*gate`	After fixing issues	Update quality decision

Test Quality Standards

Quinn ensures all tests meet these standards:

- **No Flaky Tests**: Ensures reliability through proper async handling
- **No Hard Waits**: Dynamic waiting strategies only
- **Stateless & Parallel-Safe**: Tests run independently
- **Self-Cleaning**: Tests manage their own test data
- **Appropriate Test Levels**: Unit for logic, integration for interactions, E2E for journeys
- **Explicit Assertions**: Keep assertions in tests, not helpers

Risk-Based Testing

The Test Architect uses risk scoring to prioritize testing:

Risk Score	Testing Priority	Gate Impact
9	P0 - Must test thoroughly	FAIL if untested
6	P1 - Should test well	CONCERNS if gaps
4	P1 - Should test	CONCERNS if notable gaps
2-3	P2 - Nice to have	Note in review
1	P2 - Minimal	Note in review

Fleet Management V2 Specific Guidelines

Mandatory Architecture Patterns

Service Layer Pattern (MANDATORY):

- All database operations MUST go through service functions
- Never call Supabase directly from API routes or components
- Services located in `lib/services/`

Three-Tier Supabase Client:

- Browser Client (`lib/supabase/client.ts`) - Use in Client Components
- Server Client (`lib/supabase/server.ts`) - Use in Server Components, APIs
- Middleware Client (`lib/supabase/middleware.ts`) - Use in middleware only

Critical Business Rules

1. **Roster Period System** - 28-day cycles (RP1-RP13 annual)
2. **Certification Compliance** - FAA color coding (Red/Yellow/Green)
3. **Leave Eligibility** - Rank-separated (Captains vs First Officers)
4. **Captain Qualifications** - JSONB tracking (Line/Training/Examiner)
5. **Seniority System** - Based on commencement_date

Development Standards

Type Safety:

- Strict TypeScript mode
- Generated types from Supabase schema (`npm run db:types`)
- All database operations properly typed

Error Handling:

- Use standardized error messages (`lib/utils/error-messages.ts`)
- Handle constraint errors (`lib/utils/constraint-error-handler.ts`)
- Comprehensive error logging

Testing Strategy:

- E2E tests with Playwright

- Component stories with Storybook
- Service layer unit tests

Conclusion

BMAD Method provides a structured, agent-driven approach to software development that works for both new and existing projects. By leveraging specialized AI agents and following proven workflows, you can accelerate development while maintaining high quality standards.

Key Takeaways

1. **Agents are specialized** - Use the right agent for each task
2. **Fresh context matters** - Start new chats when switching agents
3. **Document first** - Especially critical for brownfield projects
4. **Quality gates work** - QA agent prevents regressions
5. **One story at a time** - Focus and complete before moving on

Getting Help

- **Discord Community**: Join Discord
- **GitHub Issues**: Report bugs
- **YouTube**: BMadCode Channel
- **Documentation**: Review `.bmad-core/user-guide.md`

BMAD Method™ - Breakthrough Method of Agile AI-driven Development Generated for Fleet Management V2 Version 4.44.1 | October 21, 2025