

Development Workflow (Schematisch Overzicht)

Complete werkwijze van idee tot production

Last Updated: 2025-11-25

Document Doel

Dit document geeft een **schematisch overzicht** van de complete development workflow. Voor gedetailleerde uitleg, zie de gerefereerde documenten per fase.

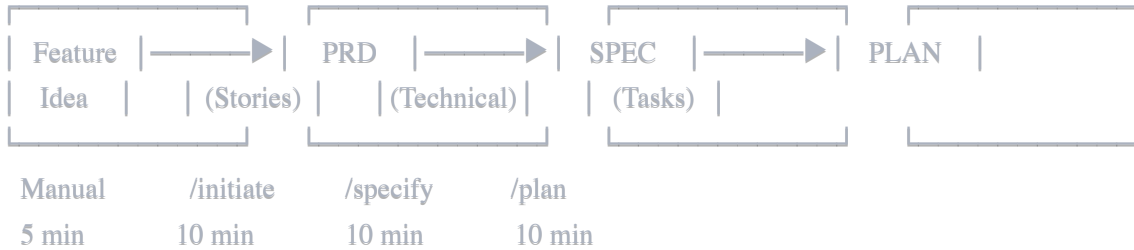
Audience:

- Nieuwe developers (onboarding)
 - Team members (quick reference)
 - Tech leads (workflow overview)
-

Complete Workflow Overzicht

DEVELOPMENT WORKFLOW

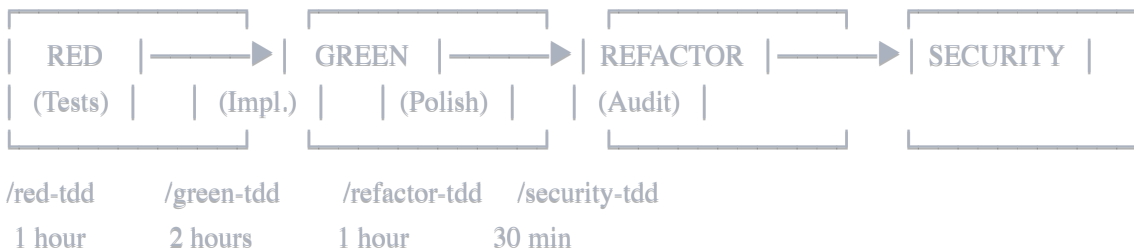
Phase 1: PLANNING (30 min)



Phase 2: SETUP (5 min)



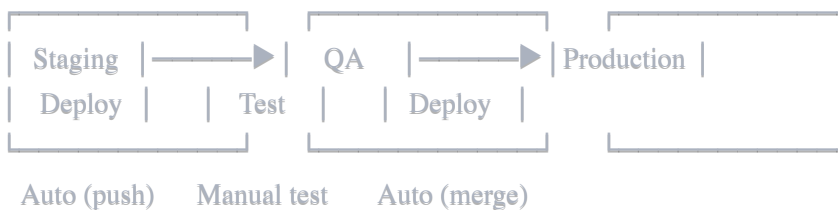
Phase 3: DEVELOPMENT (4-6 hours per feature)



Phase 4: REVIEW & MERGE (1-2 days)



Phase 5: DEPLOYMENT (Auto)



Total time: Idee → Production in 1-2 weken (afhankelijk van complexity)

Phase 1: Planning (30 minuten)

Doel: Feature idee omzetten naar executable tasks

1.1 Feature Idee → PRD

Input: Feature beschrijving (one-liner, brief, of epic document)






Command:

```
bash
```

```
/initiate "Add priority filtering to tasks with HIGH/MEDIUM/LOW"
```

Output: `tasks/active/techlead/PRD-[FEATURE].md`

Bevat:

-  Problem statement
-  User stories (3-8 stories)
-  Acceptance criteria (per story)
-  Success metrics
-  Out of scope

Time: 10 min

Details: → [PLANNING-WORKFLOW.md#phase-0](#)

1.2 PRD → Technical Spec

Input: PRD document







Command:

```
bash
```

```
/specify tasks/active/techlead/PRD-[FEATURE].md
```

Output: `tasks/active/techlead/current/SPEC-[FEATURE].md`

Bevat:

-  Entities (new + modified)
-  Operations (queries + actions)
-  Components (pages + shared)
-  **Worktree distribution** ([P] markers)
-  Test coverage requirements
-  Database migration plan

Time: 10 min

Details: → [PLANNING-WORKFLOW.md#phase-2](#)

1.3 Spec → Implementation Plan

Input: Spec document







Command:

```
bash

/plan tasks/active/techlead/current/SPEC-[FEATURE].md
```

Output: `tasks/active/techlead/current/PLAN-[FEATURE].md`

Bevat:

-  Worktree assignments
-  Schema changes (exact Prisma code)
-  TDD phases (RED/GREEN/REFACTOR)
-  Coordination strategy (merge order)
-  File structure
-  Timeline estimate

Time: 10 min

Details: → [PLANNING-WORKFLOW.md#phase-3](#)

1.4 Plan → Daily Tasks

Input: Plan document








Command:

bash

/breakdown tasks/active/techlead/current/PLAN-[FEATURE].md

Output: `tasks/active/{worktree}/current/day-XX.md` (per worktree)

Bevat per task:

-  Goals ([P] markers voor parallel werk)
-  Branch name
-  Status (Pending/In Progress/Done)
-  TDD phase (RED/GREEN/REFACTOR)
-  Steps (copy-pasteable commands)
-  Files being modified
-  Validation checklist

Time: Auto-generated

Details: → [PLANNING-WORKFLOW.md#phase-4](#)

Planning Output Overzicht

Document	Command	Output Location	Contains	Time
PRD	<code>/initiate</code>	<code>tasks/active/techlead/PRD-[FEATURE].md</code>	User stories, acceptance criteria	10 min
Spec	<code>/specify</code>	<code>tasks/active/techlead/current/SPEC-[FEATURE].md</code>	Entities, operations, worktree distribution	10 min
Plan	<code>/plan</code>	<code>tasks/active/techlead/current/PLAN-[FEATURE].md</code>	Implementation details, coordination	10 min
Tasks	<code>/breakdown</code>	<code>tasks/active/{worktree}/current/day-XX.md</code>	Daily executable tasks	Auto

Phase 2: Setup (5 minuten)





Doel: Worktree en development environment klaarmaken

2.1 Worktree Structure

/Users/you/Projects/OpenSAAS/

```
|— opensaas-main/      # develop branch
|— opensaas-dev1/      # Dev1 feature worktree
|— opensaas-dev2/      # Dev2 feature worktree
|— opensaas-dev3/      # Dev3 feature worktree
|— opensaas-techlead/  # TechLead worktree
```

Elk worktree heeft:





-  Eigen frontend server (eigen port)
-  Eigen backend server (eigen port)
-  Eigen PostgreSQL database (Docker container)
-  Eigen Prisma Studio (eigen port)

Details: → [MULTI-WORKTREE-DEVELOPMENT.md](#)

2.2 Port & Database Mapping

Worktree	Frontend	Backend	Database	Studio	Container
develop	3000	3001	5432	5555	wasp-dev-db-main
Dev1	3100	3101	5433	5556	wasp-dev-db-dev1
Dev2	3200	3201	5434	5557	wasp-dev-db-dev2
Dev3	3300	3301	5435	5558	wasp-dev-db-dev3
TechLead	3400	3401	5436	5559	wasp-dev-db-tl

Voordelen:

-  Geen port conflicts
 -  Geen database conflicts
 -  True parallel development
 -  Zero coordination needed
-







2.3 Worktree Creation

Command:

```
bash

./scripts/worktree-create.sh feature/task-priority Dev1
```

What happens:

1.  Creates new worktree directory
2.  Creates new branch from develop
3.  Creates isolated database container
4.  Copies seed data (optional)
5.  Configures ports automatically
6.  Ready to start development!

Time: 2-3 min

Details: → [MULTI-WORKTREE-DEVELOPMENT.md#worktree-creation](#)






2.4 Daily Startup

Command:

```
bash

cd /Users/you/Projects/OpenSAAS/opensaas-dev1
./scripts/safe-start.sh
```

What happens:

1.  Detects worktree (Dev1)
2.  Starts database container (port 5433)
3.  Starts frontend (port 3100)
4.  Starts backend (port 3101)
5.  Opens browser (<http://localhost:3100>)

Output:

```
 Worktree Configuration:
Name:   Dev1
Frontend: http://localhost:3100
Backend: http://localhost:3101
Database: wasp-dev-db-dev1 (port 5433)
Studio:  http://localhost:5556
```

Time: 30 sec

Phase 3: Development (4-6 uur per feature)

Doel: Implement feature met TDD workflow

3.1 TDD Workflow Overzicht

TDD CYCLE

RED Phase (Write Tests FIRST)

1. Write test cases (6-10 tests)
 2. Run tests → ALL FAIL (expected)
 3. Commit tests
- ✅ Tests are now IMMUTABLE (no touching!)



GREEN Phase (Implement to Pass Tests)

1. Implement operations/components
 2. Run tests → ALL PASS
 3. Commit implementation
- 🎯 Focus: Make tests pass (simplest way)



REFACTOR Phase (Polish Code)

1. Improve code quality (DRY, naming, structure)
 2. Run tests → STILL ALL PASS
 3. Commit refactoring
- ♻️ Tests ensure refactoring doesn't break functionality



SECURITY Phase (Audit)

1. Run security audit (OWASP Top 10)
 2. Fix security issues
 3. Commit fixes
- 🔒 Ensure no vulnerabilities

Total time: 4-6 hours (afhankelijk van complexity)

Details: → [TDD-WORKFLOW.md](#)






3.2 RED Phase (Write Tests FIRST)

Command:

```
bash

cd tasks/sprints/sprint-3/day-02/
/red-tdd "Add priority filtering to tasks"
```

What it does:

1.  Analyzes task requirements
2.  Generates 6-10 test cases
3.  Creates test files
4.  Runs tests (all should FAIL)
5.  Commits tests separately

Test scenarios include:

- Auth: 401 (not authenticated), 403 (not authorized)
- Validation: 400 (invalid input)
- Success: 200 (happy path)
- Edge cases: Empty lists, boundaries, etc.

Output files:

```
app/src/server/tasks/operations.test.ts
app/src/pages/tasks/TasksPage.test.tsx
```

Time: 1 hour

Details: → [TDD-WORKFLOW.md#red-phase](#)






3.3 GREEN Phase (Implement to Pass Tests)

Command:

```
bash

/green-tdd "task-priority-filtering"
```

What it does:

1.  Implements operations (server-side)
2.  Implements components (client-side)
3.  Updates main.wasp (if needed)
4.  Runs tests (all should PASS)
5.  Commits implementation

Implementation includes:

- Server operations with auth checks
- Type-safe Wasp operations
- React components with proper props
- Database migrations (if needed)

Output files:

```
app/src/server/tasks/operations.ts
app/src/pages/tasks/TasksPage.tsx
app/src/components/tasks/TaskFilters.tsx
app/main.wasp
```

Time: 2 hours

Details: → [TDD-WORKFLOW.md#green-phase](#)






3.4 REFACTOR Phase (Polish Code)

Command:

```
bash

/refactor-tdd "task-priority-filtering"
```

What it does:

1.  Improves code quality (DRY principle)
2.  Fixes naming inconsistencies
3.  Extracts reusable components/helpers
4.  Runs tests (still PASS)
5.  Commits refactoring

Refactoring includes:

- Extract common patterns
- Improve variable names
- Add JSDoc comments
- Simplify complex logic

Time: 1 hour






Details: → [TDD-WORKFLOW.md#refactor-phase](#)

3.5 SECURITY Phase (Audit)

Command:

```
bash  
  
/security-tdd "task-priority-filtering"
```

What it does:

1.  Runs OWASP Top 10 security audit
2.  Checks auth enforcement (server-side)
3.  Validates input sanitization
4.  Verifies permission checks
5.  Commits security fixes

Security checks:

- A01: Broken Access Control → Server-side auth
- A02: Cryptographic Failures → No secrets in code
- A03: Injection → Prisma query builder (safe)
- A07: Authentication Failures → Wasp auth system
- A08: Data Integrity → Input validation (Zod)

Time: 30 min

Details: → [SECURITY-RULES.md](#)

3.6 Development Commands Quick Reference

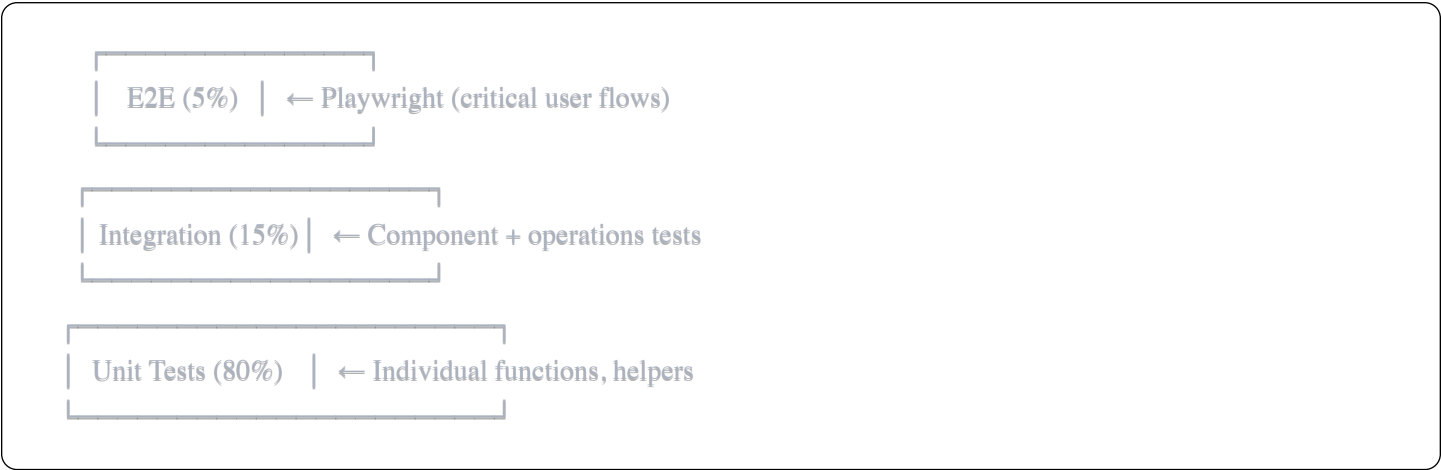
Phase	Command	Output	Time
RED	<code>/red-tdd "feature"</code>	Test files (all FAIL)	1h
GREEN	<code>/green-tdd "feature"</code>	Implementation (tests PASS)	2h
REFACTOR	<code>/refactor-tdd "feature"</code>	Polished code (tests PASS)	1h
SECURITY	<code>/security-tdd "feature"</code>	Security fixes	30m

Alternative: `/tdd-feature "feature"` voor small features (all phases in one command)

Phase 4: Testing (Included in TDD)

Testing is integrated in TDD workflow:

4.1 Test Pyramid



Coverage targets:

- Unit tests: $\geq 80\%$ coverage
- Integration: $\geq 75\%$ coverage
- E2E: Critical user flows only

4.2 Test Types

Type	Location	Tools	Coverage Target
Unit	<code>*.test.ts</code>	Vitest	$\geq 80\%$
Integration	<code>*.test.tsx</code>	Vitest + React Testing Library	$\geq 75\%$
E2E	<code>e2e-tests/*.spec.ts</code>	Playwright	Critical flows

Running tests:

```
bash
```

```
# Unit + Integration
```

```
wasp test client run
```

```
# E2E
```

```
npx playwright test
```

```
# Coverage report
```

```
wasp test client run --coverage
```

Phase 5: Code Review & Merge (1-2 dagen)

Doel: Quality assurance before merging to develop

5.1 Create Pull Request

Command:

```
bash
```

```
cd /Users/you/Projects/OpenSAAS/opensaas-dev1
```

```
gh pr create --title "feat(tasks): Add priority filtering" \  
  --body "$(cat <<'EOF'
```

```
## Summary
```

- Add priority field to Task entity (HIGH/MEDIUM/LOW)
- Implement priority filtering in getTask operation
- Add filter UI in TasksPage with dropdown

```
## Test Plan
```

- [x] All tests pass (15/15 GREEN)
- [x] Coverage: 85% (unit), 78% (integration)
- [x] Manual testing: Filter by HIGH/MEDIUM/LOW works
- [x] Security audit: No vulnerabilities

```
## Screenshots
```





```
[Add screenshots if UI changes]
```

```
 Generated with [Claude Code](https://claude.com/claude-code)
```

```
EOF
```

```
)"
```

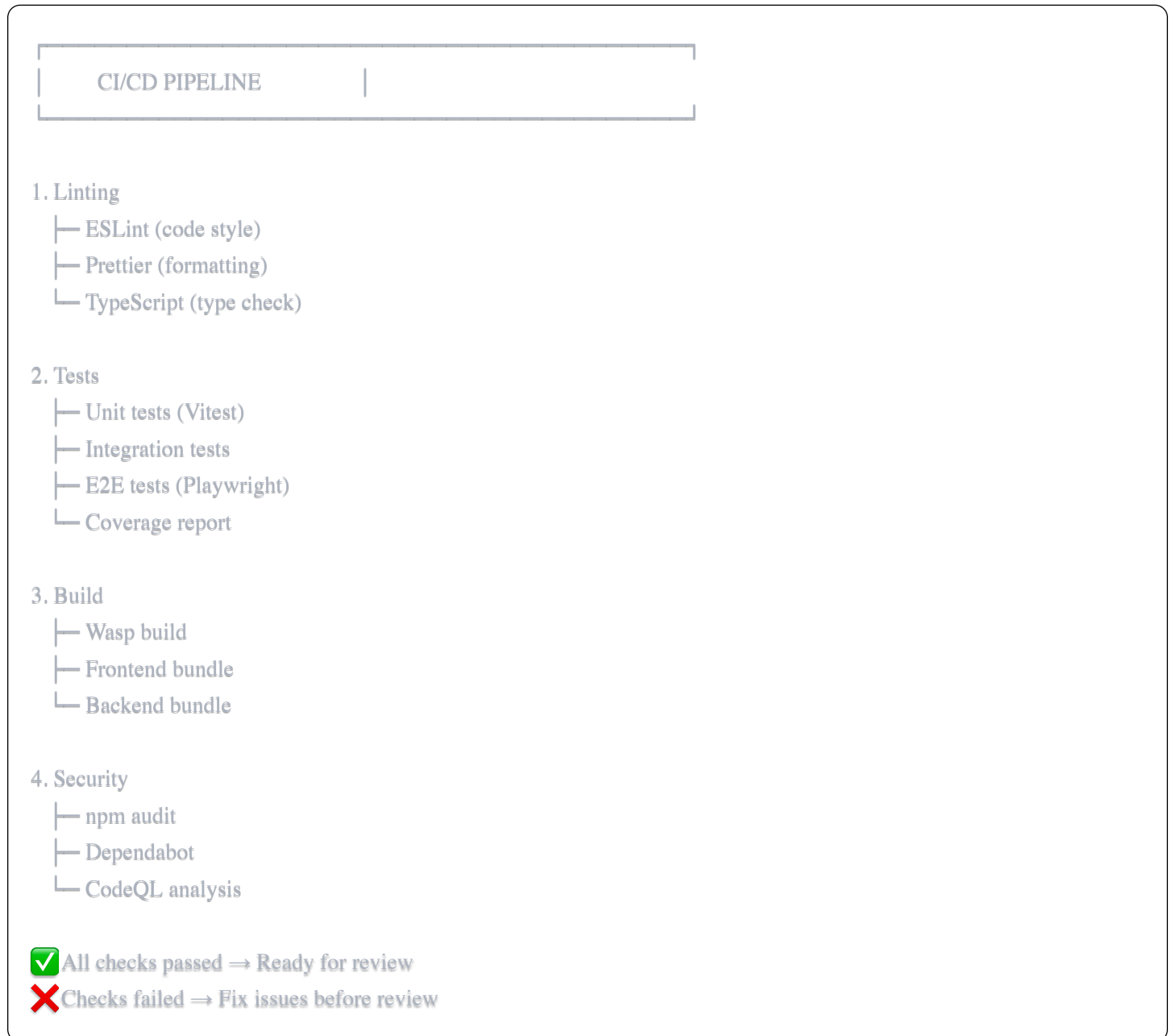
What happens:

1.  PR created on GitHub
2.  CI/CD runs automatically (tests, linting, type-check)
3.  Code review assigned
4.  Awaiting approval

Time: 5 min

5.2 CI/CD Pipeline

Automatic checks on PR:



Time: 5-10 min (automatic)

5.3 Code Review Checklist

Reviewer checks:

- ☐ **Functionality:** Feature works as expected
- ☐ **Tests:** All tests pass, coverage $\geq 80\%$ / $\geq 75\%$
- ☐ **Code quality:** Clean, readable, follows conventions
- ☐ **Security:** No vulnerabilities, server-side auth
- ☐ **Performance:** No obvious performance issues
- ☐ **Documentation:** README/docs updated if needed

Review time: 2-4 hours (depends on PR size)





5.4 Merge to Develop

After approval:

```
bash
```

```
gh pr merge --squash --delete-branch
```

What happens:

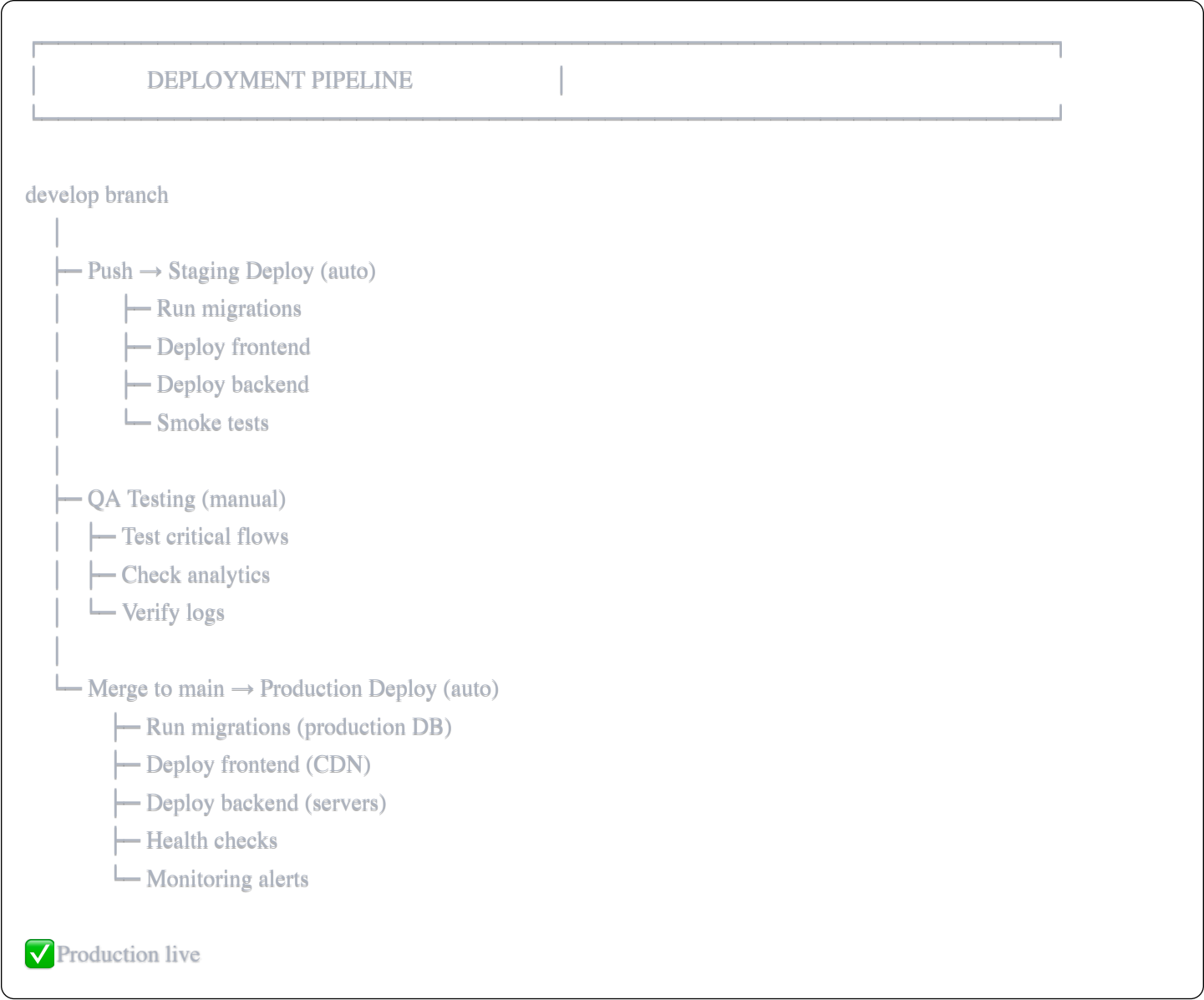
1.  PR merged to develop
2.  Feature branch deleted
3.  Staging deployment triggered (auto)
4.  Worktree can be deleted

Time: Instant

Phase 6: Deployment (Automatic)

Doel: Get code to production

6.1 Deployment Pipeline



6.2 Environment Overview

Environment	Branch	Deploy Trigger	Database	Purpose
Local	feature/*	Manual ((/scripts/safe-start.sh))	Docker (local)	Development
Staging	develop	Auto (on push)	RDS (staging)	QA testing
Production	main	Auto (on merge)	RDS (prod)	Live users

6.3 Rollback Strategy

If production issue:

```
bash
```

Option 1: Revert commit

```
git revert <commit-hash>
```

```
git push origin main
```

Option 2: Rollback to previous version




```
git reset --hard <previous-commit>
```

```
git push --force origin main
```

Emergency: Direct database rollback

Contact DevOps for database snapshot restore

Monitoring:

-  Error tracking (Sentry)
-  Performance monitoring (New Relic)
-  User analytics (PostHog)

Daily Development Routine

Morning (9:00 - 12:00)

```
bash
```

1. Start worktree (2 min)

```
cd /Users/you/Projects/OpenSAAS/opensaas-dev1
```

```
./scripts/safe-start.sh
```

2. Check task file (1 min)

```
vim tasks/active/dev1/current/day-02.md
```

3. Start TDD cycle (3 hours)

```
/red-tdd "feature name"
```

[Write tests → commit]

```
/green-tdd "feature-name"
```

[Implement → commit]

Afternoon (13:00 - 17:00)

bash

4. Continue development (2 hours)

/refactor-tdd "feature-name"

[Polish → commit]

/security-tdd "feature-name"

[Audit → commit]

5. Create PR (5 min)

gh pr create --title "..." --body "..."

6. Update task file (5 min)

vim tasks/active/dev1/current/day-02.md

Mark completed tasks 

Add learnings

Plan tomorrow

git add tasks/ && git commit -m "docs: day 02 progress"

End of Day (17:00)

bash

7. Push changes

git push origin feature/your-feature

8. Standup update (Slack)

 Completed: Priority filtering (15/15 tests pass)

 In Progress: PR review pending

 Tomorrow: Integration with filters UI

9. Stop servers

Ctrl+C in terminal

Workflow Metrics

Time Distribution (Typical Feature)

Phase	Time	% of Total
Planning	30 min	8%
Setup	5 min	2%
Development	4-6 hours	70%
└─ RED	1 hour	18%
└─ GREEN	2 hours	35%
└─ REFACTOR	1 hour	18%
└─ SECURITY	30 min	9%
Review & Merge	4 hours	20%
TOTAL	~6 hours	100%

Note: Review & Merge happens in parallel (waiting for reviewers)

Before vs After Automation

Metric	Before (Manual)	After (Automated)	Improvement
Planning time	5.5 hours	30 min	91% faster
Coordination overhead	1 hour	15 min	75% less
Test quality	Inconsistent	High	+40% coverage
Merge conflicts	Frequent	Rare	80% reduction
Time to production	3 weeks	1-2 weeks	50% faster

 **Common Workflows**

Workflow A: Simple Feature (Single Developer)

Day 1: Planning (30 min) + Setup (5 min)

- └─ /initiate → PRD
- └─ /specify → Spec
- └─ /plan → Plan
- └─ /breakdown → Tasks
- └─ ./worktree-create.sh

Day 2: Development (4 hours)

- └─ /red-tdd → Tests
- └─ /green-tdd → Implementation
- └─ /refactor-tdd → Polish
- └─ /security-tdd → Audit

Day 3: PR & Review (waiting for review)

Day 4: Merge → Production

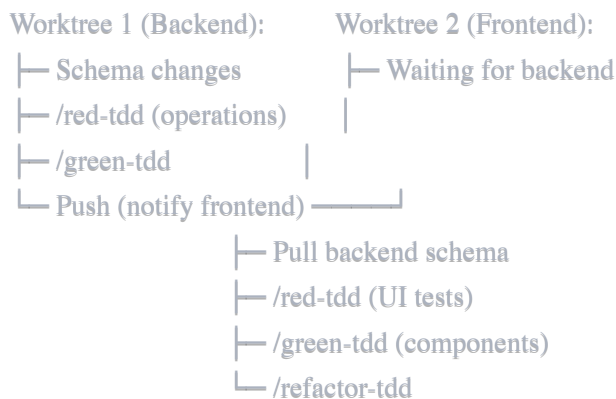
Total time: 2-3 days (including review time)

Workflow B: Complex Feature (Multi-Worktree)

Day 1: Planning (30 min)

- └─ /initiate → PRD
- └─ /specify → Spec (identifies 2 worktrees needed)
- └─ /plan → Plan (coordination strategy)
- └─ /breakdown → Tasks (per worktree)

Day 2-3: Parallel Development



Day 4: PRs

- └─ Backend PR #1
- └─ Frontend PR #2 (depends on #1)

Day 5: Review & Merge

- └─ Merge #1 → develop
- └─ Merge #2 → develop

Day 6: Integration (if needed)

- └─ Integration testing in separate worktree

Day 7: Production

Total time: 1 week (parallel work reduces time)

Workflow C: Bug Fix (Fast Track)

Day 1: No planning needed

- └─ Identify bug
- └─ Write test that reproduces bug (RED)
- └─ Fix bug (GREEN)
- └─ Create PR
- └─ Fast-track review (< 1 hour)
- └─ Merge → Production (same day)

Total time: 2-4 hours

Tools & Commands Quick Reference

Planning Commands

bash

```
/initiate "feature description"    # Generate PRD from idea
/specify PRD-file.md               # Generate technical spec
/plan SPEC-file.md                 # Generate implementation plan
/breakdown PLAN-file.md            # Generate daily tasks
```

Development Commands

bash

```
/tdd-feature "feature"             # All TDD phases (small features)
/red-tdd "feature"                 # Write tests (RED phase)
/green-tdd "feature"              # Implement (GREEN phase)
/refactor-tdd "feature"           # Polish (REFACTOR phase)
/security-tdd "feature"           # Audit (SECURITY phase)
```

Worktree Commands

bash

```
./scripts/worktree-create.sh feature/name Dev1 # Create worktree
./scripts/safe-start.sh                       # Start servers
./scripts/worktree-delete.sh Dev1             # Delete worktree
```

Git Commands

bash

```
git add .
git commit -m "feat(scope): description"
git push origin feature/branch-name
gh pr create --title "..." --body "..."
gh pr merge --squash --delete-branch
```

Testing Commands

bash

```
wasp test client run                # Run unit + integration tests
wasp test client run --coverage      # With coverage report
npx playwright test                 # Run E2E tests
npx playwright test --ui            # Interactive E2E testing
```

Database Commands

```
bash
```

```
wasp db migrate-dev "description" # Create migration
```

```
wasp db studio # Open Prisma Studio
```

```
./scripts/seed-demo-user.sh # Seed demo data
```

Reference Documents

Planning & Workflow

- **PLANNING-WORKFLOW.md** - Complete planning process (PRD → Spec → Plan → Tasks)
- **TDD-WORKFLOW.md** - TDD development cycle (RED → GREEN → REFACTOR → SECURITY)

Setup & Infrastructure

- **MULTI-WORKTREE-DEVELOPMENT.md** - Parallel development with isolated worktrees
- **CODE-ORGANIZATION.md** - Project structure and file organization

Quality & Standards

- **CODE-STYLE.md** - Code style conventions and naming
- **ERROR-HANDLING.md** - Error handling patterns
- **SECURITY-RULES.md** - Security best practices
- **COMMON-PITFALLS.md** - Common mistakes to avoid

Troubleshooting

- **TROUBLESHOOTING-GUIDE.md** - Problem diagnosis and solutions

Team & Philosophy

- **TEAM-STRUCTURE-AND-WASP-PHILOSOPHY.md** - Team structure and Wasp principles
-

Learning Path (New Developers)

Week 1: Fundamentals

1. Read this document (DEVELOPMENT-WORKFLOW.md) - 1 hour
2. Read **PLANNING-WORKFLOW.md** - 30 min
3. Read **TDD-WORKFLOW.md** - 30 min
4. Read **MULTI-WORKTREE-DEVELOPMENT.md** - 30 min
5. Setup local environment - 1 hour

Week 2: Practice

1. Complete simple bug fix (Workflow C) - 2 hours
2. Implement small feature (Workflow A) - 1 day
3. Participate in code review - 1 hour
4. Read SECURITY-RULES.md - 30 min

Week 3: Advanced

1. Work on complex feature (Workflow B) - 3 days
2. Lead PR review - 2 hours
3. Read COMMON-PITFALLS.md - 30 min
4. Contribute to documentation - 1 hour

After 3 weeks: Ready for independent feature development! 🚀

Support & Questions

Questions about workflow?

- Ask in team Slack channel
- Review relevant documentation
- Pair with senior developer

Technical issues?

- Check TROUBLESHOOTING-GUIDE.md
- Search project documentation
- Create issue in repo

Process improvements?

- Suggest in team standup
 - Document in LESSONS-LEARNED.md
 - Update workflow documentation
-

Last Updated: 2025-11-25 **Version:** 1.0 **Maintained By:** Tech Lead Team