

Definition of Done

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-03

Table des matières

1	Definition of Done	1
1.1	Methodology-Specific Adaptations ..	1
1.1.a	Scrum	1
1.1.b	Kanban	1
1.1.c	XP (Extreme Programming) . . .	1
1.2	Scope of Application	2
1.3	Epic Level Definition of Done	2
1.4	Feature Level Definition of Done	2
1.5	User Story Level Definition of Done .2	
1.5.a	1. Code Quality	2
1.5.b	2. Testing Requirements	2
1.5.c	3. Documentation Requirements	2
1.5.d	4. Non-Functional Requirements	2
1.5.e	5. User Experience	3
1.5.f	6. Acceptance Criteria	3
1.6	Sprint Level Definition of Done	3
1.7	Release Level Definition of Done	3
1.8	Enforcement Policy	3
1.9	Evolution of Definition of Done	3
1.10	Quality Gates	3
1.10.a	Code Quality Gates	3
1.10.b	Build Quality Gates	3
1.10.c	Deployment Quality Gates . . .	4
1.10.d	SonarQube Quality Gates	4

1 Definition of Done

This document establishes the criteria that must be met for any work item to be consi-

dered « Done ». Adherence to these standards ensures consistent quality across our application.

1.1 Methodology-Specific Adaptations

1.1.a Scrum

- All definitions apply as written
- Sprint and Release level definitions are mandatory
- Story points must be validated against velocity
- Sprint ceremonies must be completed

1.1.b Kanban

- Focus is on User Story and Feature level definitions
- Sprint level definition is not applicable
- Release happens when WIP limits and quality gates are satisfied
- Continuous flow metrics replace sprint metrics
- Lead time and cycle time measurements must be captured

1.1.c XP (Extreme Programming)

- Additional technical requirements:
 - ☑ Pair programming sessions completed
 - ☑ Test-Driven Development (TDD) practiced
 - ☑ Continuous Integration passes
 - ☑ Code refactoring completed

- Shorter cycles with immediate customer feedback
- User stories must have automated acceptance tests
- No technical debt allowed without immediate resolution

1.2 Scope of Application

The Definition of Done applies to different levels of work items:

- **Epics:** Large bodies of work that span multiple sprints
- **Features:** Deliverable functionality that may consist of multiple user stories
- **User Stories:** Complete functionality from user's perspective
- **Tasks:** Individual components of user stories

1.3 Epic Level Definition of Done

An epic is considered « Done » when:

- ☑ All constituent features and stories are completed
- ☑ Overall acceptance criteria for the epic are met
- ☑ System-wide integration testing is completed
- ☑ Epic-level documentation is finalized
- ☑ Business metrics for success are measured
- ☑ Stakeholder sign-off is obtained

1.4 Feature Level Definition of Done

A feature is considered « Done » when:

- ☑ All related user stories are completed
- ☑ Feature-level integration testing is completed
- ☑ Feature documentation is complete
- ☑ Feature metrics are defined and measured
- ☑ Product owner has approved the feature
- ☑ Feature demo is completed with stakeholders

1.5 User Story Level Definition of Done

A user story is considered « Done » when:

1.5.a 1. Code Quality

- ☑ Code follows our coding standards
- ☑ All code is peer-reviewed and approved by at least one team member
- ☑ Code is free of any « TODO » comments related to the implementation
- ☑ Technical debt is documented if incurred (with justification)
- ☑ Code is checked into the correct branch with proper commit messages

1.5.b 2. Testing Requirements

- ☑ Unit tests cover at least 80% of new code
- ☑ Integration tests for key user flows are implemented
- ☑ UI tests for new screens or modified UI components are added
- ☑ All tests (unit, integration, UI) pass
- ☑ Manual testing completed according to test cases
- ☑ User acceptance testing completed and signed off (for user-facing features)

1.5.c 3. Documentation Requirements

- ☑ Code is documented according to project standards
- ☑ Technical documentation is updated (if applicable)
- ☑ User-facing documentation is updated (if applicable)
- ☑ Release notes are drafted (if feature will be included in next release)

1.5.d 4. Non-Functional Requirements

- ☑ Performance meets defined benchmarks
- ☑ Feature works on all supported device
- ☑ Accessibility requirements are met
- ☑ Security requirements are implemented and verified

1.5.e 5. User Experience

- ☑ Implementation matches approved designs
- ☑ All UI strings are externalized for localization
- ☑ UI is responsive across supported device
- ☑ Error states are handled appropriately

1.5.f 6. Acceptance Criteria

- ☑ All acceptance criteria explicitly defined in the user story are met
- ☑ Product Owner has reviewed and approved the implementation
- ☑ Demo has been prepared for sprint review

1.6 Sprint Level Definition of Done

A sprint is considered « Done » when:

- ☑ All user stories meet the User Story Definition of Done
- ☑ Sprint goal is achieved
- ☑ Product increment is potentially releasable
- ☑ All automated tests are passing
- ☑ Sprint review is completed with stakeholders
- ☑ Sprint retrospective is completed and action items identified
- ☑ Metrics are captured (velocity, bugs found, technical debt incurred)
- ☑ Product backlog is refined for next sprint

1.7 Release Level Definition of Done

A release is considered « Done » when:

- ☑ All planned features are implemented
- ☑ Release notes are complete and approved
- ☑ Full regression testing is completed
- ☑ Performance testing shows acceptable results
- ☑ Security audit is completed
- ☑ User documentation is updated
- ☑ Marketing materials are prepared (if applicable)
- ☑ Legal review is completed (if needed)

- ☑ App store metadata is prepared and approved
- ☑ Release branch is created and tagged

1.8 Enforcement Policy

- Stories that do not meet the Definition of Done cannot be marked as completed
- Unfinished stories should be brought back to the backlog with lessons learned
- Exceptions to the Definition of Done must be documented and approved by both the Tech Lead and Product Owner

1.9 Evolution of Definition of Done

This Definition of Done is a living document and should be reviewed and updated at least quarterly. Any team member can propose changes, which will be discussed during a sprint retrospective before being adopted.

1.10 Quality Gates

Quality gates are specific checkpoints in our development process that ensure work meets predefined quality standards before moving forward. These gates act as formal review points where quality is measured against established criteria.

1.10.a Code Quality Gates

- Static code analysis metrics meet thresholds
 - Code complexity below defined limits
 - Duplicate code < 3%
 - Code coverage > 80%
- No critical or high-severity issues in security scan
- No known vulnerabilities in dependencies
- Successful peer review completion

1.10.b Build Quality Gates

- All unit tests pass
- Integration tests pass
- No build-breaking warnings
- Documentation generated successfully
- Code signing verification complete

1.10.c Deployment Quality Gates

- Performance testing results within acceptable ranges
- Load testing shows no degradation
- Security scanning complete
- Infrastructure as Code validation
- Configuration validation
- Backup verification

1.10.d SonarQube Quality Gates

SonarQube serves as our primary code quality monitoring tool with the following quality gates:

1.10.d.a Reliability

- Zero critical or blocker bugs
- Bug rating: A or B
- Zero known vulnerabilities
- Security rating: A

1.10.d.b Maintainability

- Technical debt ratio < 5%
- Code smells rating: A or B
- Coverage on new code > 80%
- Duplicate lines < 3%

1.10.d.c Security

- Zero security hotspots
- Security review rating: A
- OWASP dependency check: No critical issues
- No backdoor secrets or hardcoded credentials

1.10.d.d Code Organization

- Cognitive complexity per function < 15
- File lines of code < 750
- Classes per file < 30
- Functions per class < 20

These SonarQube metrics are automatically checked during each build, and failures block the deployment pipeline.