SEP Project
Jira Guidelines

CodeCatalyst UG33
Team Documentation

Version 1.0

August 30, 2025

Contents

1 Project Epics

This section outlines the main Epics for our SEP project. Choose the appropriate Epic when creating tickets to ensure proper organization and tracking.

Epic	Description
Optimising Speed Performance	Multithreading, parallel processing, profiling, GPU acceleration
Codebase Quality & Refactoring	Code cleanup, comments, standards, maintainability improvements
Optimising Compression Performance	Algorithm improvements, block detection, compression maximization
Documentation	User guides, algorithm explanations, setup instructions
Assignment & Administrative Tasks	Submissions, reports, retrospectives, project management

Table 1: Project Epics Quick Reference

2 Ticket Types

This section defines the different types of tickets used in our Jira workflow and their specific purposes.

Type	Purpose
Epic	Large feature or goal that spans multiple tasks/sprints
Story	Feature or user requirement that contributes to an Epic
Task	Smaller, discrete work items (e.g., code refactoring, documentation)
Bug	Issues found in the code that need fixing
Chore	Maintenance or non-feature work (e.g., dependency updates)

Table 2: Jira Ticket Types and Their Purposes

3 Ticket Statuses & Transitions

3.1 Default Workflow

The standard workflow follows this progression:

```
Backlog     To Do     In Progress     In Review     Done
```

3.2 Status Descriptions

3.3 Transition Rules & Best Practices

- **Cannot move to "In Progress" without an assignee**
- **Only one reviewer per PR** (if applicable)
- **Tickets must always belong to an Epic**
- **Ticket should be linked to PRs and commits** using branch naming convention:

```
UG33-<ticket-number>-<short-description>
```

Status	Description
Backlog	Ticket is created but not yet planned for a sprint
To Do	Ticket is planned for the current or upcoming sprint
In Progress	Work has started; ticket has an assignee
In Review	Work completed and awaiting peer review
Done	Work completed, reviewed, and merged/implemented
Won't Do	Ticket will not be implemented (optional for cleanup)

Table 3: Ticket Status Definitions

4 Ticket Creation Guidelines

Follow these steps when creating new tickets:

1. **Create tickets in the correct type** (Epic, Story, Task, Bug, Chore)
2. **Provide a clear title and description**, including acceptance criteria
3. **Assign ticket to yourself** only when actively working
4. **Link to Epic** (if applicable)
5. **Add labels** for priority, component, or sprint

4.1 Writing Effective Tickets

Writing clear, actionable tickets is crucial for effective project management. This section provides examples and templates to help create high-quality tickets that follow the workflow described in Section 3.

4.1.1 Good Example

Good Example

Title: Implement multi-threading to improve speed

Description:
Introduce multi-threading to the compression pipeline to enhance processing speed while preserving correctness. Ensure that block model slices are processed in parallel without exceeding memory constraints and that the output remains fully valid for the verification service.

Acceptance Criteria:
- Compression algorithm correctly handles concurrent processing of slices or parent blocks
- No label mismatches or missing blocks in output
- Performance improvement measurable against single-threaded baseline
- Threading implementation is safe and avoids race conditions or deadlocks

Why it's effective:
- Clear **what** needs to be done (introduce multi-threading)
- Explains **why** (enhance processing speed)
- Defines **scope and constraints** (memory limits, output validity)
- Provides **concrete acceptance criteria** that define success

4.1.2 Poor Example

Poor Example

Title: Make the algorithm better

Description: (empty)

Problems with this approach:
- Vague title that doesn't specify **how** improvement will be achieved
- Empty description provides no context, constraints, or goals
- Missing acceptance criteria makes verification impossible
- Lacks clear actionable steps for the assignee

4.1.3 Ticket Template

Template

Title Format:

```
<Action Verb> <Component/Feature> to <Goal/Outcome>
```

Examples:
- *Implement caching to reduce memory usage*
- *Refactor authentication to improve security*

Description Structure:
1. **What:** Describe the specific work to be done
2. **Why:** Explain the business or technical rationale
3. **Constraints:** Note any limitations, dependencies, or requirements

Acceptance Criteria Guidelines:
- Use concrete, verifiable outcomes
- Include performance metrics where applicable
- Address edge cases and error handling
- Specify compatibility requirements

4.1.4 Template Application Example

Template Application Example

Title: Refactor compression output validation

Description:
Refactor the validation module to ensure all parent blocks are correctly processed and no tags are missing in the output. This refactoring is needed to address current validation gaps that cause downstream errors in the verification service. The solution must maintain compatibility with both single-threaded and multi-threaded execution modes.

Acceptance Criteria:
- All blocks are correctly processed and tagged without data loss
- Solution works seamlessly in both single-threaded and multi-threaded runs
- No missing blocks or incorrect labels in validation output
- Maintains or improves current validation performance

5 Definition of Done & Acceptance Criteria

This section defines the explicit criteria for when a ticket can be moved to "Done" status, ensuring consistent quality and completion standards across all work items.

5.1 General Definition of Done

A ticket can only be moved to "Done" when **ALL** of the following criteria are met:

- **Acceptance criteria fulfilled** - All acceptance criteria listed in the ticket description are satisfied
- **Code reviewed and approved** - At least one team member has reviewed and approved the work
- **Testing completed** - Appropriate tests have been written and are passing
- **Documentation updated** - Relevant documentation has been created or updated
- **No blocking issues** - All identified bugs or blockers have been resolved
- **Code merged** - Changes have been successfully merged into the main branch

5.2 Epic-Specific Completion Criteria

Different Epic types may have additional specific requirements:

5.3 Reviewer Checklist

Before approving a ticket for "Done" status, reviewers should verify:

Epic Type	Additional Requirements
Speed Performance	Performance benchmarks completed, no regression in correctness
Code Quality	Code formatting applied, comments added, no increased complexity
Compression Performance	Compression ratio measured, algorithm correctness verified
Testing & Validation	Test coverage maintained/improved, edge cases addressed
Documentation	Documentation reviewed for clarity, examples provided
Administrative	Deliverables submitted, deadlines met, stakeholders notified

Table 4: Epic-Specific Completion Requirements

Reviewer Checklist

Code Quality:

- Code follows project conventions and standards
- No obvious bugs or logical errors
- Appropriate error handling implemented
- Code is readable and well-commented

Functionality:

- All acceptance criteria have been met
- Feature works as expected in different scenarios
- No regression in existing functionality
- Performance impact is acceptable

Testing & Documentation:

- Adequate test coverage provided
- Tests are passing and meaningful
- Documentation is accurate and complete
- Changes are properly logged/documented

6 Ticket Updating

Maintain tickets throughout their lifecycle:

- Move ticket to **In Progress** when work begins
- Update **status, assignee, and description** as needed
- Add **comments** for blockers, notes, or updates

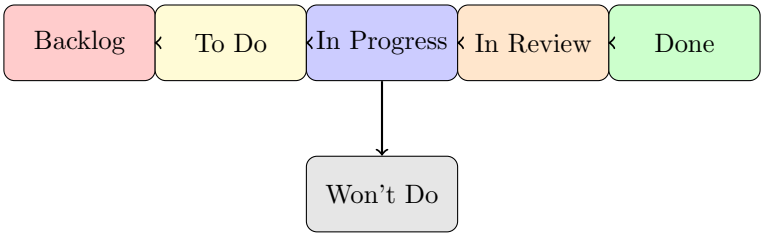
7 Ticket Closing

Properly close tickets following these steps:

- Move ticket to **In Review** when work is completed
- Ensure PR is approved and linked to ticket

- Move ticket to **Done** once merged and verified
- Tickets marked **Won't Do** should include a reason in comments

8 Visual Workflow Diagram



9 Storage & Access

- Store the full documentation in the team **shared drive** or **project wiki**
- Ensure all team members can access and reference the guidelines