Code Review Process: Guidelines for Continuous Improvement and Collaboration

A good review process requires that pull requests get addressed as soon as possible in order to prevent the project from being impeded. Ideally, pull requests are reviewed within ***two hours of their submission.***

Version Control

Version	Description	Author
Version 1.0	Initial version of PR Review process	@Rashmi Wickramasinghe

Get feedback with pull requests

Pull requests support reviewing and merging code collaboratively. When a developer adds a feature or fixes a bug, they create a pull request to start merging changes into the upstream branch. Team members review and approve the code before finalizing it. Use pull requests to review work in progress and get early feedback. Owners can abandon a pull request at any time without committing to merge.

Get code reviewed

Code review during a pull request aims beyond identifying obvious bugs, that is the role of tests. Effective code review detects subtle issues that could cause costly problems later.

There is no such thing as “perfect” code but there is only *better* code. Reviewers should not require the author to polish every tiny piece of a change list before granting approval. Rather, the reviewer should balance out the need to make forward progress compared to the importance of the changes they are suggesting. Instead of seeking perfection, what a reviewer should seek is ***continuous improvement*.**

Code review is a way to have a conversation about the code where participants will:

- **Improve code quality** by identifying and removing defects before they can be introduced into shared code branches.
- **Learn and grow** by having others review the code, we get exposed to unfamiliar design patterns or languages among other topics, and even break some bad habits.
- **Shared understanding** between the developers over the project's code. Code reviews encourage and strengthen collaboration and communication between developers. The team gains a clear history of all changes between the main branch and feature branches.

Pre-Review Quality Requirements

Before submitting a Pull Request (PR) for review, ensure the following criteria are met:

1) Traceability

- Every PR **must be linked to a Jira issue/story**.

2) Branch Naming

- The branch name **must follow the agreed naming conventions**.

2.1. Branch Naming Convention:

To maintain clarity and traceability, all development branches must follow the naming pattern below:

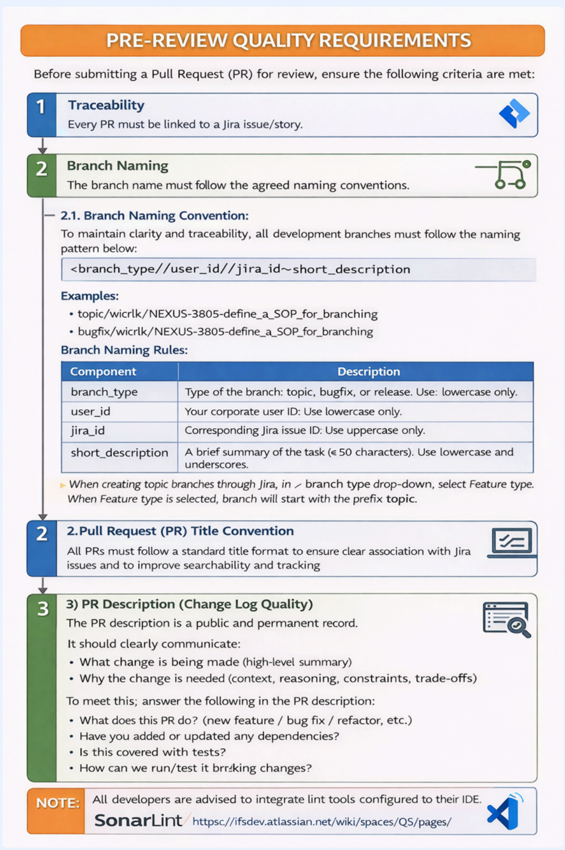
```
<branch_type>/<user_id>/<jira_id>-<short_description>
```

Examples:

```
topic/wicrlk/NEXUS-3805-define_a_SOP_for_branching
bugfix/wicrlk/NEXUS-3805-define_a_SOP_for_branching
```

Branch Naming Rules:

Component	Description
`branch_type`	Type of the branch: `topic`, `bugfix`, or `release`. Use lowercase only.
`user_id`	Your corporate user ID. Use lowercase only.
`jira_id`	Corresponding Jira issue ID. Use uppercase only.



PRE-REVIEW QUALITY REQUIREMENTS

Before submitting a Pull Request (PR) for review, ensure the following criteria are met:

1 **Traceability**
Every PR must be linked to a Jira issue/story.

2 **Branch Naming**
The branch name must follow the agreed naming conventions.

2.1. Branch Naming Convention:
To maintain clarity and traceability, all development branches must follow the naming pattern below:

```
<branch_type//user_id//jira_id~short_description
```

Examples:
- topic/wicrlk/NEXUS-3805-define_a_SOP_for_branching
- bugfix/wicrlk/NEXUS-3805-define_a_SOP_for_branching

Branch Naming Rules:

Component	Description
branch_type	Type of the branch: topic, bugfix, or release. Use: lowercase only.
user_id	Your corporate user ID: Use lowercase only.
jira_id	Corresponding Jira issue ID: Use uppercase only.
short_description	A brief summary of the task (≤ 50 characters). Use lowercase and underscores.

▸ *When creating topic branches through Jira, in ⸝* branch type *drop-down, select Feature type. When Feature type is selected, branch will start with the prefix* **topic**.

2 **2.Pull Request (PR) Title Convention**
All PRs must follow a standard title format to ensure clear association with Jira issues and to improve searchability and tracking

3 **3) PR Description (Change Log Quality)**
The PR description is a public and permanent record.

It should clearly communicate:
- What change is being made (high-level summary)
- Why the change is needed (context, reasoning, constraints, trade-offs)

To meet this; answer the following in the PR description:
- What does this PR do? (new feature / bug fix / refactor, etc.)
- Have you added or updated any dependencies?
- Is this covered with tests?
- How can we run/test it brr:king changes?

NOTE: All developers are advised to integrate lint tools configured to their IDE.
SonarLint httpsc://ifsdev.atlassian.net/wiki/spaces/QS/pages/

`short_description`	A brief summary of the task (≤ 50 characters). Use lowercase and underscores.

When creating *topic branches* through Jira, in *branch type* drop-down, select *Feature* type. When *Feature* type is selected, branch will start with the prefix *topic*.

2.2 Pull Request (PR) Title Convention

All PRs must follow a standard title format to ensure clear association with Jira issues and to improve searchability and tracking.

Format:

`[jira_id] <short_description>`

Example:

`[NEXUS-3805] code promotion to dev platform`

PR Title Rules:

Component	Description
`jira_id`	The relevant Jira issue ID. Must be in uppercase.
`short_description`	A concise summary of the work (≤ 50 characters).

> ⚠️ **Note:** Including the Jira ID is mandatory as it links the PR directly to the Jira issue and Bitbucket (BB), enabling automated traceability and efficient tracking.

3) PR Description (Change Log Quality)

The PR description is a **public and permanent record**. It should clearly communicate:

- **What** change is being made (high-level summary)
- **Why** the change is needed (context, reasoning, constraints, trade-offs)

To meet this, answer the following in the PR description:

- What does this PR do? (new feature / bug fix / refactor, etc.)
- Have you added or updated any dependencies?
- Is this covered with tests?
- How can we run/test this change?
- Does it introduce any breaking changes?

4) Code Coverage

- Code coverage expectations must be met.
- Any coverage-related issues must be resolved **before raising/submitting the PR**.

5) Static Code Analysis

- New code **must not introduce new violations** compared to the existing baseline.

6) Initial Build

- The initial build / CI checks must pass successfully **before submitting the PR for review**.
- Atleast 2 latest builds should be passed

7) Jira Readiness for Review

- The Jira issue must be moved to **PR Review** state.
- The Jira issue must be **assigned to the first reviewer(s)**

8) PR Submission

Once all the above are satisfied and CI checks have passed, the PR can be submitted for review.

NOTE:

All developers are advised to integrate lint tools configured to their IDE. Following are recommended tools.

- SONAR Lint Intergration to VS Code

Code Review Process

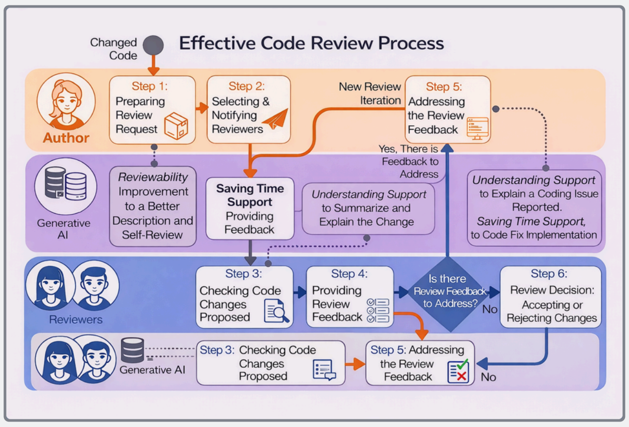
To ensure pull requests (PRs) are reviewed consistently without slowing down delivery, this process uses a **two-phase review approach**. It starts with a **distributed team review** where PRs are individually assigned and reviewed asynchronously, and only escalates to the **architect** when architectural input is required. This helps the team catch most issues early, keeps reviews predictable, and reserves architect time for the changes that truly need it.

Phase 1: Assigned Individual Review (Team)

- A PR becomes ready for review (partial submissions allowed).
- The PR link is shared in the review chat in advance.
- Instead of a scheduled meeting, **each team member spends about 1 hour reviewing assigned PRs**.
- Review feedback includes comments, suggestions, and questions.
- Outcome: the team decides if the PR requires architectural review.

Decision Gate: “Architect revisit needed?”

- **No:** The author addresses comments and finalizes the PR normally.
- **Yes:** The PR advances to Phase 2.


Changed Code
Effective Code Review Process
Author
Step 1: Preparing Review Request
Step 2: Selecting & Notifying Reviewers
New Review Iteration
Step 5: Addressing the Review Feedback
Yes, There is Feedback to Address
Generative AI
Reviewability Improvement to a Better Description and Self-Review
Saving Time Support Providing Feedback
Understanding Support to Summarize and Explain the Change
Understanding Support to Explain a Coding Issue Reported. Saving Time Support, to Code Fix Implementation
Reviewers
Step 3: Checking Code Changes Proposed
Step 4: Providing Review Feedback
Is there Review Feedback to Address?
No
Step 6: Review Decision: Accepting or Rejecting Changes
Generative AI
Step 3: Checking Code Changes Proposed
Step 5: Addressing the Review Feedback
No

Code Review Process

Phase 2: Architect Review (Scheduled) • The PR link is added to the architect review meeting chat, and the architect is notified. • The architect reviews it during a **recurring 1-hour review slot** (meeting time may shift based on availability). • The architect provides feedback, then the PR returns to the author for finalization.	

Code Review Guidelines (for the use reviewer)

- The review should include inline comments for each significant change to facilitate tracking and future learning.
- **Reference on style guides**
  - Google Style Guides : This contains the style guidance for following languages and many more.
    - Go Style Guide
    - JavaScript Style Guide
    - Java Style Guide
    - JSON Style Guide
    - Markdown Style Guide
    - Python Style Guide
    - Shell Style Guide
  - Formatting conventions : Formatting conventions for Tekton
  - Build, test, and deploy with Pipelines | Bitbucket Cloud | Atlassian Support
    Formatting conventions for pipelines
  - Introduction - The Rust Style Guide : RUST Style Guide

- If the PR review is completed reviewer should approve the PR.

How to Handle Reviewer Comments (Instructions for change log owner)

When a change log is submitted for review, it is common for the reviewer to provide feedback through various comments. Understanding how to effectively manage and address reviewer comments is essential.

Don’t take it personally

- The objective of a code review is to uphold the quality of our codebase and products. When a reviewer offers feedback on your code, consider it as their effort to assist you, enhance the codebase, and benefit Build and Deployment, rather than as a personal criticism of you or your skills.
- **Never respond in anger to code review comments.** It is a serious breach of professional etiquette that will remain in the code review tool indefinitely. If you find yourself too angry or irritated to respond kindly, it is advisable to step away from your computer for a while or focus on another task until you regain the composure needed to reply politely.

Fix the code

- If a reviewer expresses confusion about any aspect of your code, your initial action should involve clarifying the code itself. If the code remains unclear, consider inserting a code comment to elucidate the purpose of the code. Only when a comment appears unnecessary should you resort to providing an explanation within the code review tool.
- When a reviewer encounters difficulty comprehending a section of your code, it is probable that other individuals reviewing the code in the future will face similar challenges. Composing a response within the code review tool does not benefit future code readers; however, enhancing code clarity or incorporating code comments significantly aids their understanding.

Think collaboratively

- Writing a change log can be a demanding task that requires significant effort. It is indeed gratifying to submit one for review, feel a sense of completion, and believe that no further adjustments are necessary. However, it can be disheartening to receive feedback requesting changes, particularly if you do not share the same perspective.
- During such instances, it is crucial to pause and reflect on whether the reviewer's input offers valuable insights that can benefit the codebase and Build and Deployment team. Your initial inquiry should always revolve around understanding the reviewer's requests.
- If you find yourself unable to comprehend the feedback, do not hesitate to seek clarification from the reviewer.
- Furthermore, if you grasp the comments but hold a different viewpoint, it is essential to approach the situation with a collaborative mindset rather than a combative or defensive one.

Resolve conflicts

The initial approach to conflict resolution should prioritize reaching a consensus with your reviewer. If reaching a consensus proves challenging, refer to The Standard of Code Review for guiding principles in handling such scenarios.

PR Merge

- Please always delete the branch after merging the PR as a best practice

Resources

- Best Kept Secrets of Peer Code Review : Copyright © 2013 by SmartBear Software best-kept-secrets-of-peer-code-review_redirected.pdf