Development, Peer Review, and Acceptance Process

Summary	Assignee
NGSS Sprint Peer Reviewer	Matthew Braatz

The scope of a Peer Review includes verifying the deployment readiness checks, ensuring integration test coverage, and providing feedback on readability, documentation and judicious use of language. The goal is that you clearly understand the change set and how it changes the operation of the component.

The Peer Review Process starts with the assignment of a Review subtask, and ends with the handover to the Product Owner for approval and merge. The review subtask should have the same priority as the story.

Developer

Once you have completed the work on your story and have had a successful build in Jenkins, you will need to create a pull request and a Jira subtask following these steps:

Create a PR

- In CodeRepo go to pull request tab
    - create pull request
    - select which branch you worked on and request it be merged to main
        - There are instances where we are merging to a different branch other than main, but most of the time it's main
    - [continue]
    - Fill out the following:
        - title: CKM-parent ticket # and a short description of the changes
        - Description: Optional. More detail can be added here.
        - Reviewers: list both the peer review AND the Product Owner for the component
    - [create]

Submit for Peer / Acceptance Review

- In Jira you will then need to create a subtask for the Peer Review
    - While this is possible to create manually, we have made a shortcut that standardizes the naming convention as well as many of the other necessary pieces of information for this sub-task.
        - On the issue for which you need a Peer Review, click on the `More` dropdown. (Alternatively: press the `.` or `,` keyboard shortcut while the Jira Issue is on screen)
        - Scroll to the bottom of the list and select `Dev: Create Review/Acceptance Sub-Task` (Alternatively: If using the keyboard shortcut, begin typing the aforementioned command)
    - This will create a sub-task that has the following appropriately set:
        - A summary / title of the subtask in the form: “Peer / Acceptance Review: *component name*” ex: “Peer / Acceptance Review: s3-file-service”
        - A priority equivalent to that of the parent issue.
        - The correct component, matching that of the parent issue.
        - The current Peer Reviewer assigned to it.
            - This is done by using the assignee of a static Jira task to keep track of the peer reviewer as it changes as shown at the top of this document.

    - As a developer, you will need to **add one final piece of information to this sub-task**
        - In the description, add the link to the pull request
    - Put the CKM parent ticket into “waiting” while leaving the subtask as "open"

Address Items from Review

- If there are items that need to be addressed from the peer/acceptance task:
    - You will be notified of any comments made by the peer reviewer/PO via email
        - Make sure your Crowd account email address has been updated to your Apothesource email (instead of your VA email)
    - The subtask for the peer review will be put in [waiting] by the Peer Reviewer
    - When you go to address these concerns the parent ticket should be put back to [in progress]
    - Once you have finished addressing the concerns you should put the parent ticket back in to [waiting] and change the subtask to [open]
    - Comment on the subtask that it is ready for re-review
    - Continue with this Jira workflow and comments until the peer review has been completed and the branch has been merged (Pull Request has been closed) and the parent ticket is closed by the Product Owner

<u>Note:</u>

- SRVDD reviews only need to go through acceptance review so the subtask should be assigned directly to the Product Owner (listed as the reporter on the story)

How to tell when your story is done and ready for review

In addition to meeting all of the acceptance criteria for a story, the following tasks must be completed for any story requiring code changes before the story should be considered ready for review (unless the story or the component's product owner explicitly states otherwise):

- Update any dependency version and/or plugin version that is explicitly declared in the project to the latest version
    - The exception to this rule is for patch releases. Patch releases need to remain stable and versions should only be updated if explicitly stated in the story
    - If a dependency version update causes a breaking change, the version should be updated to the latest non-breaking version, a comment should be added stating that the declared version is the latest non-breaking version, and the PO should be notified so that a new story can be created to upgrade the dependency to the latest version
    - If a parent project version is updated and the new parent version includes management of a dependency that is explicitly declared in the child project, the explicit version declaration in the child project should be removed (assuming the change is non-breaking)
    - See How do we make sure that dependency versions are up-to-date in our projects?for more info
- Create and/or update tests to verify the functionality of all new code
    - any error responses documented in swagger
    - security (with and without VA Mobile JWT, JWTs with and without authorized roles/resources/scopes, etc.)
    - any expected health indicator status checks
    - For services, integration tests are preferred whenever possible (client-based testing with live or mocked service dependencies). At a minimum, unit tests should be used to fill gaps where integration tests can't test certain branches or exception handling in the code.
    - Tests should cover as many happy path/positive and error response/negative scenarios as is reasonable, including (but not limited to):
- Build the project successfully (no build errors, test errors/failures, or static analysis errors/warnings without commenting/disabling tests and static analysis)
    - The project should build both locally and in Sandbox Jenkins on the feature branch after all changes are committed
- Run Fortify and commit the generated files after all code changes are completed (including running new scans after any non-test code changes for review feedback)
    - Unaudited findings must be fixed in the code or audited in Workbench as "Not an issue" with an appropriate reason if the finding is truly not an issue
    - "Log Forging" findings must be fixed in the code, not audited (typically Sanitizer.sanitize() on unchecked logging input will fix this finding)
- Run ready-check after Fortify scans are complete
    - Any warnings or errors should be addressed, either with changes or to verify that the finding is a false positive for the component
    - Note: many of the checks performed by ready-check are only relevant for Java/JSP-based service components
- Update the dibr.md (deployment document) with any Consul and/or Vault variable changes
    - For most projects, the DIBR is auto-generated and is updated by modifying the entries under the "documentation" section of the project's metadata.yaml file and rebuilding the project
- Update the CHANGELOG with the story id and a brief description of changes
    - Include any Consul and/or Vault variable changes
    - Include any major framework dependency version changes (JSP, HMP, etc.)
- Update the README
    - Update any existing content that is no longer accurate due to the current story
    - If necessary, add new content for any important information that developers or clients might need to know based on the current story
- Check with the component's product owner to determine if any SRVDD or other documentation updates are necessary

Keep in mind that this list may not be fully comprehensive and required tasks may differ slightly between components and/or change over time. That is why communication with the product owner is critical to understanding the intended outcome of a story. At any point after picking up a story, contact the product owner directly with any questions or if there is a need to clarify anything on a story.

<u>**Peer Reviewer**</u>

During Sprint Planning, the upcoming Peer Reviewer should not take any Major priority Jira issues. At most, they should take one to two Standard priority Jira Issues to be done in the absence of PRs. Discuss with the PM as early as possible within the Sprint if it seems like these Standard issues are not able to be completed so work may be redistributed and rebalanced.

You will be notified of all peer review tasks through Jira. There are many ways to filter but the sprint execution board will show you all your assigned tasks and subtasks or you can access the two dimensional filter statistic: CKM Peer Reviewer on the CKM Sprint Planning Dashboard. Peer reviews should be completed in order by highest priority.

Prioritizing Peer Reviews

- Based on the highest priority ticket select the subtask
    - Put the subtask [in progress]
    - Go to the parent ticket and review the JIRA issue before going to the open pull request. You should be familiar with what the JIRA issue is attempting to solve and the accompanying Acceptance Criteria.
    - Open the pull Request in CodeRepo
        - The Pull Request should be automatically linked to the parent story if the PR was titled properly within CodeRepo.

PR Verification

- Verify the Pull Request (PR):
    - There are three checks on all pull request – no open tasks, no needs work, and has a successful build on the current commit. These three checks must be satisfied before a pull request can be approved or merged.
    - Confirm the PR is a functional/dependency change; if not, then it should be a non-release commit.
    - Confirm the PR includes minimum updates needed for a functional change (e.g. fortify, changelog, and SRVDD).
    - Confirm that the Changelog correctly describes the change within the scope of the Release Type. Check that the changelog follows the guidelines below.
    - Check for "There are merge conflicts". In most cases, send back to the issue owner to resolve conflicts.
    - Check if code review branch is significantly behind main. In cases where there are code, configuration, or environment changes (i.e., not documentation-only changes), send back to issue owner to update.
    - Check that the last main build was successful in Jenkins. If not, work with story owner or Product Owner to resolve the main build failure.
    - Assign code review to self if not already assigned

Local Checks

- Pull branch in git

    - If it is behind the main branch (git log —-no-merges main ^*ckm-xxx-dev-branch*), pull in commits from main (git pull origin main)
    - Build locally (e.g. mvn clean verify)
        - Verify all tests passed.
        - Did the local build produce any changes that are not included in the MR? *(note: Jenkinsfiles now include randomly generated 5-character build hashes which change on each build, so these can be safely ignored)*
    - Run ready-check; if any errors, notify developer and consider whether the rule should be altered or ignored.
    - Optional: run the service locally and verify health check
    - Optional: additional testing locally.
    - Optional: build and test in Jenkins.

Additional Items to Review

- Review

    - Check for new configurations and corresponding documentation.
    - Check for javadocs. Any new or substantially modified public method must have a Javadoc. Trivial methods (e.g., getters/setters) and trivial changes (e.g., whitespace) do not require a Javadoc added.
    - If changes require a new release, ensure that changelog has correct next release version (i.e. major vs. minor vs. patch release).
        - Ensure that changelog reflects functional changes, dependency updates (e.g. library upgrades), and new environment variables (if applicable).
        - Ensure that version changes are complete in all areas of the code (e.g. pom, Jenkins file, k8s yaml, etc.).
    - Optional: Apply Code Review Checklist.

Discussion and Approval

- If there are items that need to be addressed:
    - Add discussion points in code review. Set expectations. State which resolutions are required for merge.
        - The reviewer will add a task (using PR create task button) for any comment that requires some action by the developer. All tasks must be completed before the PR can be closed.
        - For any comments that do not require a task the developer should add the "thumbs up" emoji to acknowledge that the comment has been read.
    - The subtask for the peer review should be put in [waiting]
    - A comment should be made on the subtask saying that there are comments to address

- Re-review as needed

    - When you go to re-review put the sub-task should be put back to [in progress]
    - Re-review as described above to ensure all concerns were addressed
- Approve
    - mark your approval on the code review
    - comment in the subtask "approved"
    - Change the status on the subtask back to [open]
    - assign the review subtask to the Product Owner.
    - After discussions have been resolved,

Acceptance Review (Product Owners only):

You will be notified of all Peer / Acceptance Review tasks through Jira. There are many ways to filter but the sprint execution board will show you all your assigned tasks and subtasks or you can access the two dimensional filter statistic: CKM Peer Reviewer on the CKM Sprint Planning Dashboard. Peer reviews should be completed in order by highest priority.

- Based on the highest priority ticket select the subtask
    - Put the subtask in progress
    - Go to the parent ticket and review the story before going to the open pull request
- Perform the checks outlined above for the peer reviewer
    - if there are issues to be addressed follow the same protocol as the peer reviewer for tracking Jira tickets and commenting on tickets
- Once approved merge the Pull Request
- close the subtask
- close the parent ticket
- Follow the Release process if this new change will be pushed to release

Note:

- SRVDD reviews only need to go through acceptance review so the subtask should be assigned directly to the PO (listed as the reporter on the story)
    - once SRVDD tasks are closed notify Brittany Fowler to submit changes to VAMFAT

CHANGELOG Guidelines (based on GL guidelines)

A good changelog entry is descriptive and concise. It focuses on functional changes, not implementation details. It is written for an audience of release managers and consuming services, not for the PO or reviewer. It clearly communicates and does not include unnecessary information.

- Any new configuration variables **must** have a changelog entry.
- Security fixes **must** have a changelog entry.
- Any new service dependencies (including mocks) **must** have a changelog entry. Example: "Add service dependency to mobile-facility-service"
- Any user-facing change **should** have a changelog entry. Example: "provider-context-selection-web now sorts the site list"
- Any significant library updates **should** have a changelog entry. Example: "Update user-session-service-client library to 3.29.0 to support the /vars/all endpoint"
- Performance improvements **should** have a changelog entry.
- Any docs-only changes **should** **not** have a changelog entry.
- Any developer-facing change (e.g., refactoring, technical debt remediation, test suite changes) **should** **not** have a changelog entry. Example of what not to do: “Update helm dependency for user-service”
- If a release is required only for development workflow (e.g., to create a helm artifact for updating transitive chart dependencies), then the changelog entry **should** clearly state the release's non-functional purpose. Example: "Update vista dependencies in helm chart."

Supporting Documentation

Code Review Template

Code Review Template .docx

Peer Reviewer Rotation Roster (CKM)

- Robin Raju (Sprint 55) - Kevin O: First Week of Sprint.
- Unknown User (krampt) (Sprint 56)
- Mark Remi (Sprint 57) wont finish out this sprint due to FAM work
- Ben Griner (Sprint 58)
- Dallas Vaughan (Sprint 59)
- Edward Felch (Sprint 60)
- Cody Nelson (Sprint 61)
- Chuck McCormick (Sprint 62)
- Unknown User (gaineys) (Sprint 63)
- @Drew Connelly (Sprint 64)
- Unknown User (jonasw) (Sprint 65)
- Michael Ferraro (Sprint 66)
- Unknown User (hallbl) (Sprint 67)
- Eugene Shvartsman (Sprint 68)
- Ben Griner (Sprint 69)
- Scott Heimmer(Sprint 70)
- Randy Nolen (Sprint 71)
- Edward Felch (Sprint 72)
- Robin Raju (sprint 73)
- Scott Brawner (sprint 74)*
- Cody Nelson (sprint 75)
- Joseph Damuth (Sprint 76)
- Brian Poploskie (Sprint 77)
- Laith Al Samir (Sprint 78)
- Scott Thompson (Sprint 79)
- Dallas Vaughan (Sprint 80)
- Drew Connelly (Sprint 81)
- Chuck McCormick (Sprint 82)
- Eugene Shvartsman (Sprint 83)
- Sanjeev Pillutla (Sprint 84)
- Michael Ferraro (Sprint 85)
- Ben Griner (sprint 86)
- Randy Nolen (sprint 87)
- Dallas Vaughan (Sprint 88)
- Todd Dunagan (sprint 89)
- Sandeep Pillutla (sprint 90)
- Robin Raju (sprint 91)
- Laith Al Samir (Sprint 92) - Matt to fill in 9/1, 9/2, 9/6
- Matthew Braatz (sprint 93)
- Drew Connelly (Sprint 94)
- Joseph Damuth (Sprint 95)
- Cesar Agustin Garcia Vazquez & Chuck McCormick (Sprint 96)
- Dallas Vaughan (sprint 97)
- Yari Yakup (sprint 98)
- Edward Felch (Sprint 99)
- Brian Poploskie (Sprint 100)
- Jason Wilson (Sprint 101)
- Eugene Shvartsman (sprint 102)
- Ben Griner (sprint 103)
- Cody Nelson (Sprint 104)
- Eugene Shvartsman (sprint 105)
- ~~Steve Gazzo~~ Todd Dunagan to cover last week(Sprint 106)
- Damien Harsany (Sprint 107)

- Rees Byars (Sprint 108)
- Chuck McCormick (Sprint 109)
- Drew Connelly (Sprint 110)
- Yari Yakup (Sprint 111)
- Todd Dunagan (Sprint 112)
- Robin Raju (Sprint 113)
- Matthew Braatz (Sprint 114)
- Edward Felch (Sprint 115)
- Jason Wilson (Sprint 116)
- Eugene Shvartsman (Sprint 117)
- Ben Griner (Sprint 118)
- Cody Nelson (Sprint 119)
- Damien Harsany (Sprint 120)
- Rees Byars (Sprint 121)
- Chuck McCormick (Sprint 122)
- Yari Yakup (Sprint 123)
- Brian Poploskie (Sprint 124)
- Todd Dunagan (Sprint 125)

If your name is not tagged, then the line item is tentative. We will be making newer devs peer reviewer as they get their credentials.

*not in standard rotation

Peer Reviewer Rotation Roster for VAOS