Code Review Process

Albert Ghukasyan

Table of Contents

1 Summary

1.1 Document Scope

This document contains ITDs and Implementation details for the new Code Review Process.

1.2 Update History

Every Update to this document must have the corresponding row in a grid below.

Version	Architecture Update	Update Date	Status	Owner
v 1.0	Initial Document	22 Sep 2020	In Progress	Albert Ghukasyan

1.3 Abbreviations / Acronyms

Below is the table with abbreviations and acronyms used in this document

Abbr.	Description
MR	Merge Request (Gitlab)
PR	Pull Request (Github)
ITD	Important Technical Decision
UAT	User Acceptance Testing

2 Important Technical Decisions

2.1 ITD : Branching Model

ITD 1 – Use Gitflow branching model	
THE PROBLEM	For version controlling purposes need to choose git workflow to use..
OPTIONS CONSIDERED (Decision in bold)	**1. Gitflow** 2. Gitlab flow
REASONING	Main reason is that gitflow allows us to continuously support previous versions in production while developing the next version.
DETAILS	See details here.

2.2 ITD : Task Management/Ticketing system

ITD 2 – Use Jira as a Task management/Ticketing system.	
THE PROBLEM	Choose the Task Management system.
OPTIONS CONSIDERED (Decisions in bold)	**1. Jira**
REASONING	Earthlink already is using Jira as a Task management system,the team is familiar and Jira is the leading ticket management system in the market and is able to accomplish requirements.
DETAILS	See details here.

2.3 ITD : Code Changes

ITD 3 – Track all the code changes in the Jira.	
THE PROBLEM	How to track changes to the codebase.
OPTIONS CONSIDERED (Decision in bold)	**1. Track Changes in the Jira.**
REASONING	Each MR(or PR) must be tracked in the Jira. This way we can keep track of what is changed in the code base and prevent invisible pushes and commits to the code.
DETAILS	See details here.

ITD 4 – Use ONE-TO-ONE Mapping for ‘Code Change->Jira Ticket’	
THE PROBLEM	What is the mapping between code change and jira.
OPTIONS CONSIDERED (Decision in bold)	**1. ONE-TO-ONE** 2. MANY-TO-ONE 3. ONE-TO-MANY
REASONING	We should not allow multiple MR(or PR) per one Jira ticket and one MR(or PR) for multiple tickets. Single Bug or Single feature must be pushed to the repository via single MR.
DETAILS	See details here.

ITD 5 – No Ticket -> No Change	
THE PROBLEM	Allow or not changes in the codebase without corresponding Jira tickets.
OPTIONS CONSIDERED (Decision in bold)	1. Allow 2. **Reject**
REASONING	All the changes in the codebase must be tracked in the Jira,there **should not be** any change in the code-base main branches without a Jira ticket.
DETAILS	See details here.

ITD 6 – Use multiple types of tickets for code changes.	
THE PROBLEM	Identify common types of tickets for code changes.
OPTIONS CONSIDERED (Decision in bold)	1. **Multiple types of tickets** 2. Single type of ticket
REASONING	Code changes can be done for different types of problems. Each type of problem may be escalated for review to a different person. To have flexibility for these types of scenarios, a decision was made to support different types of tickets. 1. **Bug** -> Bug in the code. 2. **Feature** -> New feature or enhancement to the code. 3. **Optimization** -> Optimize/fast-up already working code. 4. **QE Review** -> Quality Enforcement ticket (this is a specific ticket type which will be created for reviewers). (list can be changed)
DETAILS	See details here.

3 Implementation Details

3.1 Gitflow Branching Model

The simplified version of the gitflow is explained below(for the detailed explanation please check this).

Each repository will contain 4 types of branches.

1. Main branches
2. Feature branches
3. Hotfix branches.
4. Release branches.

3.1.1 Main Branches

There are two Main branches, **master** and **develop**.

Master branch is the branch containing codes for the **current production** version. This means that master is the ONLY branch from where we are deploying to the Production.

Develop branch is the most recently updated branch and contains the updates done after last release.

Main branches will have branch protection rules and the only way to change them will be MRs (or PRs).No direct push to master and develop branches will be allowed and MRs will be approved ONLY by a pre-configured list of members.

3.1.2 Feature Branches

Feature branches correspond to the single unit of work. Feature branch is created from the develop branch. This means whenever you have to fix some bug or add a new feature, you must check out the most recent version of **develop** branch, create a branch from it and name it **feature/JIRA-KEY.**

As an example, you have to fix the bug which is tracked in the Jira with the key SOC-1234, you must create a branch and name it feature/SOC-1234, work on it and whenever you finished your work, create a MR (or PR in case if using GitHub, not GitLab) into the develop branch.

3.1.3 Hotfix Branches

Hotfix branches correspond to the single unit of work. Hotfix branch is created from the master branch. This means whenever you have to do hotfix on an already released to production version, you must check out the most recent version of **master** branch, create a branch from it and name it **hotfix/JIRA-KEY.**

As an example, you have to do a hotfix which is tracked in the Jira with the key SOC-1234, you must create a branch and name it hotfix/SOC-1234, work on it and whenever you finished your work, create a MR (or PR in case if using GitHub, not GitLab) into the **develop and master** branches.

3.1.4 Release Branches

Release branches correspond to the single **release**. Release branch is created from the develop branch. Whenever you need to freeze the branch for the new release,you must check out the most recent version of **develop** branch, create a branch from it and name it **release/SOME_KEY_HERE.**

SOME_KEY_HERE is defined per project,it's based on a release schedule.

As an example, you have a product that you are releasing monthly,your release branches can have the name in a format YYMM(e.g. release/2009,this means Sep 2020 release). If your product is released weekly ,format can be YYWW ,where WW is WeekNumber from 1 to 52).

Once the release branch is created ,it is deployed to staging and/or UAT environments.

If testing found issues that need to be fixed before production deployment you must create a **feature** branch from release branch,fix it and create a MR to the release branch.

After testing is done,the **release** branch is merged to both **develop** and **master** branches and the **master** branch is deployed to production.

3.2 Jira Ticketing Model

3.2.1 ITD details (3,4,5)

ITDs 3,4 and 5 stating

1. Track all the code changes in the Jira
2. Use ONE-TO-ONE Mapping for 'Code Change->Jira Ticket'
3. No Ticket -> No Change

This means,all the changes must have corresponding jira tickets. There should not be any change in the codebase without a corresponding jira ticket (reviewer MUST reject the Merge Request if there is no ticket specified for the MR).Each ticket must have only ONE MR associated with it and ONE MR must contain only fix for ONE ticket(reviewer MUST reject the MR it it contains fixes for 2+ jira tickets).

This is done to

1. Prevent untracked commits to main branches.
2. To be able to rollback a single fix without touching other fixes.
3. To be able to get reports on what was done during specific releases and how.

3.2.2 ITD details (6)

ITD 6 is stating that we must support multiple types of tickets for the same product.

Currently identified 4 types of the tickets

1. **Bug** -> Bug in the code.
2. **Feature** -> New feature or enhancement to the code.
3. **Optimization** -> Optimize/fast-up already working code.
4. **QE Review** -> Quality Enforcement ticket (this is a specific ticket type which will be created for reviewers).

First 3 types of tickets have the same workflow and are created for developers to implement the changes.

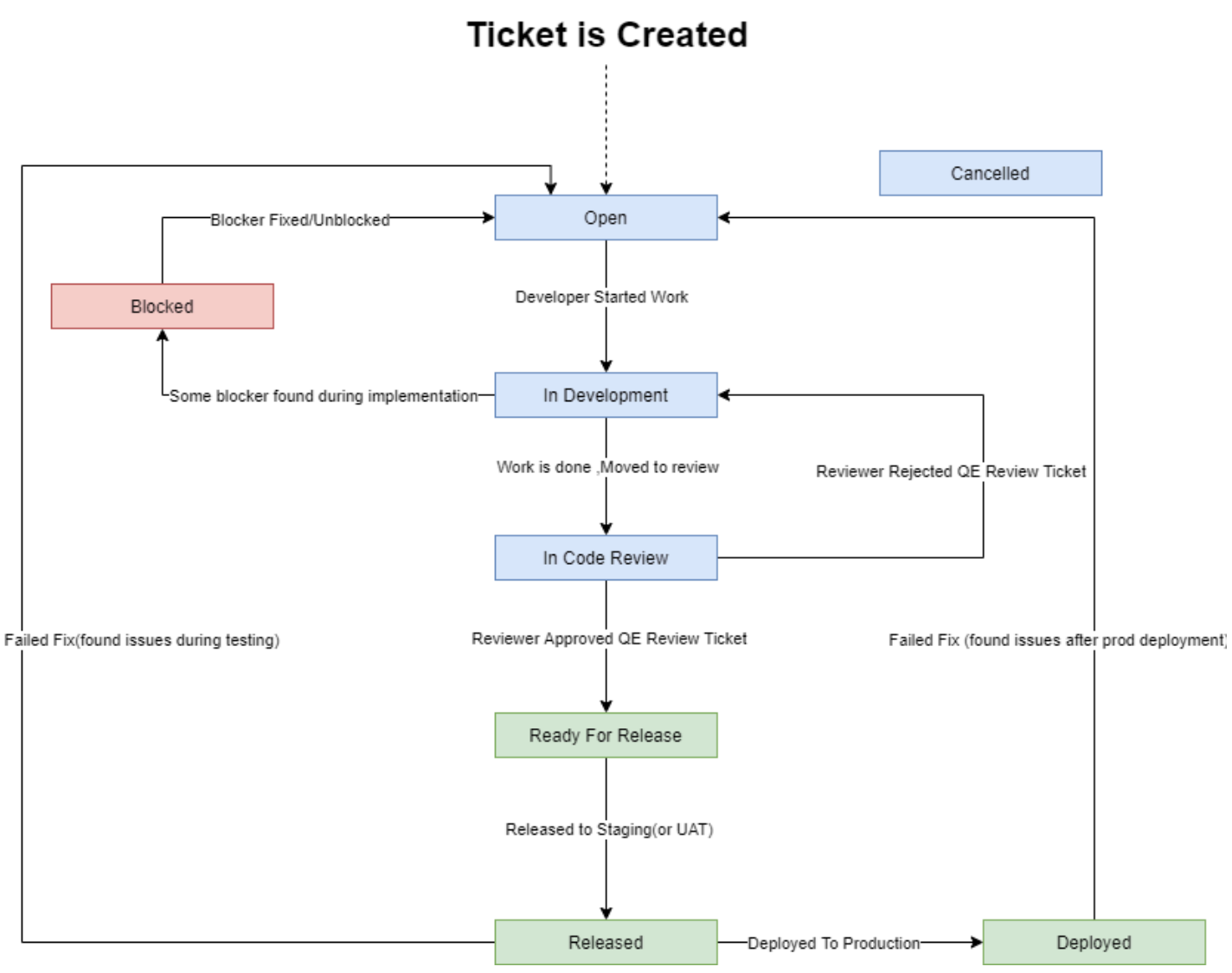
4th type (QE Review) is only for reviewers and it's workflow is simple.

To accomplish requirements we need to have 3 Custom fields defined for each Project in the Jira. Fields are below

1. Bug ticket reviewer
2. Feature ticket reviewer
3. Optimization ticket reviewer

Using values from these fields Jira automation will identify to whom QE review tickets must be assigned.

The workflow for 3 types of tickets are below



The ticket must have 3 additional fields(except common ones like description,assignee,etc).

1. MR link -> Link to Merge Request
2. Fix Version -> this will show the release number (e.g. Release-2009).
3. Release Scheduled date -> Shows when is the release date.

The workflow works in a way explained below.

Step 1 : Ticket Created -> It is moved to status **Open**.

Step 2 : Developer picks the ticket from the queue ,assigns to himself/herself.

Step 3 : Developer starts work and if he/she finds some blocker ,moves ticket to **Blocked** state with corresponding comment.If no blocker,ticket is moved to **In Development** status.

Step 4 : Work is done,MR is created. Developer moves the ticket into **'In code review'** status and puts the MR Link field (URL to Merge request).

Step 5: Once the ticket is moved to **In Code Review** status,Jira automation is creating a new Ticket as a Sub-Task to the main ticket.The new ticket type is **QE Review** and is assigned to the corresponding user(Using the project custom fields).

Step 6: Reviewer picks the **QE Review** ticket and cheks the MR from the main ticket.

Step 7: If the reviewer approves the QE Review ticket,he must put the review Rate(some number later used for reports),then the main ticket will be auto-transitioned into the **Ready For Release** status.If reviewer is Rejecting the ticket ,the main ticket will be auto-transitioned into the **In Development** status**.**

Step 8: When the ticket is in **Ready For Release** status, TPM for that product must put **Fix Version** and **Release Scheduled date** fields of that ticket(these fields can be filled before).

Step 9: Once the release is deployed to the staging(or UAT) environment,TPM or the responsible person is moving the ticket to **Released** status.

Step 10: Once it's deployed to the production environment,TPM or the responsible person is moving the ticket to **Deployed** status(the last status in the workflow).

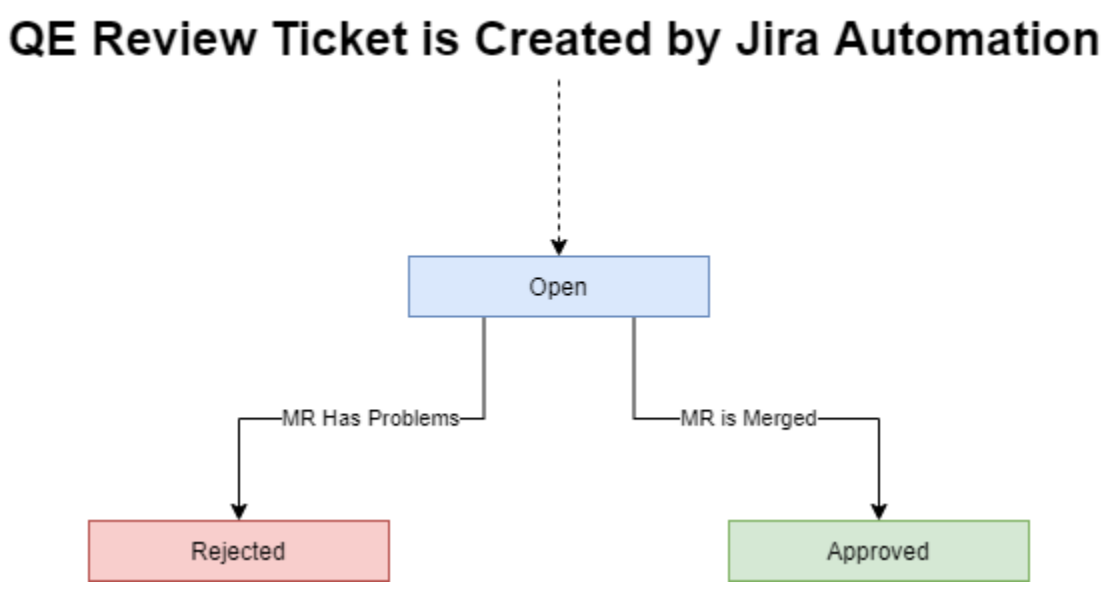
There are some exceptions for this workflow.

Exception 1 : Ticket can be transitioned to the **Cancelled** status anytime.

Exception 2 : Ticket can be transitioned to the **Open** status from **Released** status if issues found after Staging(or UAT) release.

Exception 3 : Ticket can be transitioned to the **Open** status from **Deployed** status if issues found after production deployment.

The workflow for the QE review tickets is below



The workflow works in a way explained below.

Step 1 : Sub-task added to the Main ticket,ticket is created -> It is moved to status **Open** and assigned to corresponding user.

Step 2: MR is good,Reviewer approves the QE Review ticket,QE Review ticket moved to Approved status.The main ticket auto-transitioned into the **Ready For Release** status.

Step 3: MR has issues,Reviewer Rejects the QE Review ticket,QE Review ticket moved to **Rejected** status.The main ticket auto-transitioned into the **In Development** status.