

# SDLC REDUX

- Overview
- Guidelines
- Product Development
  - Created
  - Review Ready
  - Backlog
  - READY - READY
  - Points
  - Sizing
- Ticket Workflow
  - Open
  - Data Impact Check
  - In Progress
  - Code Review
  - MR
  - Ready for QA
  - QA Bypass
  - UAT
  - DONE-DONE
  - Ready to Deploy
  - Deployed
- Failed Deployment
- Backlog Review
- Engineering Priorities

## Overview

How to work a bug or feature. This process is iterated on and improved by all members that are using it.

## Guidelines

A feature or bug is not considered complete until it is fully developed, fully tested, deployed to production and confirmed working. The steps should be automated wherever possible.

Getting a ticket ready to start requires the ticket to be in a state of READY-READY. This is from a Product - Engineering perspective, both teams should agree that the ticket is ready.

Getting a ticket ready to ship to production requires the ticket to be DONE-DONE. This is from an Engineering - Stakeholder perspective, both teams should agree that the ticket is done. Stakeholder will generally be Product but could be any other team/person.

## Product Development

This is the planning part, which includes the PRD being written.

### Created

Product writes requirements, in conjunction with engineering and design in the Jira project for the appropriate team. When the requirements are complete from product and design perspective, the ticket is moved to Review Ready

### Review Ready

Product, design and engineering review the ticket together. Engineering gets clarifications on any questions. If it is deemed ready, it will be added to the engineering backlog.

- **Questions to be answered**
  - Why are we doing this?
  - Does this affect anyone else? Will there be cross team collaboration?

- Does this make sense to me?
- Scope is written by Product
- UI design is completed
- Have reviewed the code to understand the process - time boxed to an hour or two
  - Technical scope defined
  - Any hurdles defined
  - Used to determine size/points

## Backlog

The backlog is sorted by Product and Engineering. We use sprints to define what we will be working on in the next couple of weeks.

## READY - READY

- READY for implementation and completable in ONE sprint
- The requirements are complete and fully understood
- Acceptance criteria is complete and understood
- Monitors and metrics defined
- Bugs
  - Include the stack trace if applicable (from Datadog, container logs, etc)
- Often a Product written ticket needs to be broken into engineering tickets
- UI design reviewed by engineering
  - This might cause greater amounts of refactoring before started
  - Potentially break out in phases/multiple phases of deployment
- The API and/or interactions between UI and backend services
- Ready from both a development and testing perspective

## Points

**Use non-consecutive point values.** Sufficiently separate the values to make each estimate more clear-cut. Don't split hairs by discussing whether a story is a 9 or a 10! Popular patterns are Fibonacci series (1, 2, 3, 5, 8, 13, 21, etc.) and powers of 2 (1, 2, 4, 8, 16, etc.). Anything estimated above a 16 should be broken into multiple stories and re-estimated.

**Stabilize the team.** Don't make important decisions using estimates calculated during the early forming stages of a team's evolution or while members are coming and going. Until a team's membership, estimation process, and velocity stabilize, the estimates will not be accurate.

**Get inputs from everyone** (business-layer developers, front-end developers, database developers, testers, etc.). Only the development team (not the Product Owner) can estimate the points but it is important that estimates reflect the perspective of everyone involved in completing the story.

**Make informed estimates.** The Product Owner should present the story and acceptance criteria to the development team and answer questions during an interactive backlog refinement meeting. The team should capture any details they need to remember when implementing the story.

**Iterate and improve.** Review the team's performance during retrospectives and adjust as needed. Keep the reference stories current and recalibrate stories on the backlog regularly. You can guard against "point inflation" by comparing new estimates with stories estimated in the past.

**Don't overthink it.** The team should arrive at a consensus on the points for a story in a reasonable time. Don't try to convert hours to points or vice versa when estimating. However, once story points are assigned, let development team members estimate their tasks in hours.

Ideal pointing:

Pointing done by Fibonacci scale:

- 1 ~ an hour
- 2 ~ a day
- 3 ~ 2-3 days
- 5 ~ a week
- 8 ~ a sprint

There is no above 8. If you have a ticket above 8, it needs to be made into multiple tickets.

Each dev should have 8 or less points, per sprint.

Pointing is done based on the task, not who is doing the task.

Points must also include QA time. A one line change does not always mean an easy testing stage.

## Sizing

This is the risk/complexity. Increases the more integration required with other teams or services. How much is unknown - exploratory. But mostly should be covered in READY-READY

- Use case complexity - this drives code, automated tests, and QA
- Cyclomatic complexity - this drives the amount of refactoring needed
- Integration complexity
- Affected Components
- Amount of refactoring before the new requirements can be implemented
- Development and testing complexity
- Include the amount of automated tests to be created and or fixed
- Not included: Who is completing the ticket. This is just velocity metric

## Ticket Workflow

### Open

1. Choose an open ticket at the top of the sprint or Kanban backlog and assign it to yourself
2. Read through the ticket and make sure you understand what is being asked. If you don't understand, reach out to the team lead, design, engineering manager or Product
3. Carefully read through the description and acceptance criteria
4. Review any UI designs
5. The ticket should be written in such a way that it can be coded and shipped, independently, in a 2 week sprint
6. If the ticket is dependent on other tickets we may need merge them, particularly if they are not QA-able by themselves
7. If the ticket cannot be done in two weeks, we may need to split it into multiple tickets
8. Confirm with the Product Manager and Design if this is a ticket that should be UAT'd
9. Engage QA and go over what is required for the ticket. Work out the basic design and test cases to be covered. For a bug, this is less important. How to reproduce can be covered in QA notes or ticket comments
10. Write a high level design in the description of the Jira and review with a lead on the team
11. If the ticket is good for development move it into Data Impact Check
12. Set Planned End Date - this is for Project Management to set.

### Data Impact Check

Review the requirements and conduct a [Data Impact Check](#)

Once the Data Impact is completed (this may be just letting the Data team know and adding them to a ticket), move the ticket to In Progress

### In Progress

1. Create your feature branch following guidelines here: [Branches, Commits, and Merge Requests](#)
2. Complete your development
  - a. We look for 80% code coverage as a general rule and when it makes sense
3. Avoid Drive By Coding
4. Bump all-the-things!
  - a. If your code changes a library, api or client that is used by other applications, you must bump the versions in the other applications
  - b. Use GitLab search to confirm you have not missed any dependencies
5. Smoke test your code to make sure it works!
6. IF you are developing and the ticket turns into something much larger/more complicated than expected, we may need to go back a previous step: tech design, product req, UI design, etc, etc
7. Update Planned End Date!!!!
8. Confirm you have coded the right thing, i.e. the implementation meets the description in the ticket. Especially that it meets any and all acceptance criteria and design.

## 9. Send to Code Review

### Code Review

1. Used for getting eyes on. Also used for getting Design or other stakeholder early approval.
  - a. This is Design QA for FE changes
2. Put a WIP tag (gitlab) on it so people know you're still working on it
3. Once you have Design or stakeholder early approval. Move to PR
  - a. Remove your WIP tags before going to PR!

### MR

1. It is your responsibility to get the MR it's approvals
  - a. The longer it sits, the more code you have to rebase / merge from master
  - b. Publish the link to the MR in one of the Slack channels for code reviews
  - c. Ask for approvals in stand up
  - d. Ask for approvals in our squad Slack channels. If you are planning to use QA Bypass for this, please identify in the title of the PR /MR
2. You will need to get the minimum MR approvals. There are code owners for every repo - you will need minimum from both code owners and the total minimum reviewers. See [Branches, Commits, and Pull Requests](#)
3. Address all comments on the MR promptly. If you agree with the change you can simply update the code. If you disagree with the change, explain why or ask for more information. This is often easier in Slack or in person
  - a. If there are disagreements, the minimum quorum is (Code Owners - 1)
  - b. Must fix
    - i. These must be fixed OR you need to have a good reason not to be done. This should be a discussion
    - ii. Please be cognizant of trying not to "that's the not way I would do it" if it's still a valid solution - this is really hard
    - iii. If a code review spawns a new ticket. LINK the tickets and include in the code review.
      1. This is not an easy get out of refactoring jail free card.
  - c. Nit
    - i. Whitespace and style issues
    - ii. Fix if this is not an emergency
4. Once you have addressed all the comments on the PR you can get your code in a good state for QA
  - a. Rebase/merge from master - get your code up to date
    - i. Update any dependencies bump-all-the-things!
    - ii. Make sure you have a working build
  - b. QA Notes
    - i. You MUST write QA notes before it is ready for QA. The engineer that tests the ticket if it is not the engineer who moved the ticket into MR will send it back if it is not obvious what is to be tested
    - ii. Good things to put in QA notes are:
      1. For bugs - how to reproduce, expected results, actual results (pre-fix)
      2. Features or configurations required
      3. Data setup required in databases, files. Provide scripts if possible
      4. Mention any functionality that was updated that is not obvious from the description of the Jira. Ex. refactoring to common methods may have unexpected impact to other parts of the system. Or for Operator, there is code used by multiple app
      5. If files are generated or needed, provide instructions on where to find them or how to create them. Templates are great.

### Ready for QA

An engineer will review and make sure you have all the pieces in place for them to test 2.

Your ticket must meet [Ready for QA Expectations](#)

Your ticket has details of the acceptance criteria for testing so another engineer can test the ticket **QA**

### QA Bypass

If you are requesting QA Bypass, **in addition** to any items listed above you must also:

1. You must have completed smoke testing

2. Include near 100% automated test coverage
3. A comment on the ticket with justification for why your changes do not need to be tested
4. Add QA Bypass to the title of the Jira
5. QA approval
  - a. Could include a demonstration to QA
  - b. This should include a review that all the use cases were covered
  - c. Manager approval

## UAT

Not required for most bug fixes, technical tasks or tech debt. Work with your engineering manager to determine if a ticket needs UAT and who should be in that group.

 We often skip this step but occasionally either Product or Design wish to conduct User Acceptance Testing before the ticket ships.

## DONE-DONE

Does your ticket meet all the requirements for Definition of Done. The Done-Done is because it needs to be both coded and tested (QA and UAT).

The definition of done:

1. Code complete
2. Unit tests as close to 100% as possible
3. Integration tests as close to 100% as possible
4. Smoke tested
5. MR Approvals
6. Manual QA (if necessary) with comments on what was tested, screenshots
7. Functional tests (Transitioning to Cypress IO from Nightwatch JS)
7. Ticket is shipped and verified to be working in Production without errors

## Ready to Deploy

1. You are responsible for successfully getting your ticket into production.
2. You must be available throughout the whole process until your ticket is successfully deployed and confirmed working
3. Have a rollback/revert plan
4. Resolve any merge conflicts in your code. The Release Manager (RM) will not merge conflicted code
5. The RM will contact you when your ticket is at the top of the list for deployment.
6. When you confirm your availability, the release manager will start merging your code into master.
7. If there are dependencies, the RM will merge the source first. You are responsible for updating the dependencies in the other branches. The RM will provide you with the master build names - or you can be following the process and pro-actively get the numbers.


## Deployed

1. Once your code is deployed, monitor logs for errors in any of the services you changed.
2. Once Deployed To Prod you are responsible for updating the ticket status in Jira

## Failed Deployment

1. First, mistakes happen. Things are missed in development and in QA. We move so fast and our systems are so large, sometimes deployments go sideways. The key is to learn from the experience and don't make the same mistake next time. You will be working with the Incident manager to monitor, report and diagnose the issue. TODO: Link incident management
2. Second, **ROLLBACK**
3. If there have been multiple releases, they will have to rollback multiple releases. It's a good idea to know the stable build that pre dated your release.
4. The most important thing is to get the broken code out of production to stop the bleed. Even if you are not entirely sure if your release is the culprit, there is ZERO harm in rolling back.
5. It is not always possible to rollback. DB changes are difficult if not impossible to rollback.

6. Diagnose the issue and fix your code. You may possibly use the same ticket number. You will need to go through a quick version of the SDLC. In extreme cases, we may be able to immediately deploy a hotfix.
7. Tell the RM when you are ready to deploy again.
8. Third, **REVERT**
  - a. If you have not solved (including QA re-review!) the issue within an hour or so. Revert your code so the rest of the company can continue working
  - b. This is also the case if your "small" fix needs regression testing again. The worst case is to continuously be deploying half working code
9. Incident Report (if applicable)
  - a. Be prepared to write and present the incident report

 If you are on vacation or sick or otherwise not working, your tickets will likely get stuck. Please be sure to hand off your tickets before vacation.

## Backlog Review

TODO: Elaborate here

1. Design review
2. Product review

## Engineering Priorities

Priorities are set in reverse of the Jira board. Priorities in order are:

1. Ready to Deploy & Deployed QA
2. PR
  - a. Conduct reviews on other people's code
  - b. Respond to comments on your own PRs and make improvements
3. In Progress
4. Only when you have no other tickets in the above to you take a new ticket off the top of the queue!