Software Citation Tools
Overview of Software Development Process and Methodology

Latest Revision: 2016-10-20

Team ForCite:
Eric Lee
Rob Lowe
Sam Mosher
Colin O'Neill

Methodology

This project will develop a suite of related tools using an iterative and incremental development methodology with evolutionary prototyping. Each full process iteration will result in the creation of a stable deliverable. This deliverable may be an initial working prototype of a tool, to be built upon in future iterations, or a new, stable version of an existing tool with additional features. By utilizing this evolutionary prototyping approach, the team hopes to get early and frequent feedback from the community in order to develop a suite of tools that provides the highest degree of value.

Each process iteration will consist of four phases: requirements gathering, analysis and design, development, and deployment. These phases will exist as general guidelines driving the development of the software. They will not be considered discrete or immutable: some overlap of phases may occur, and even past iterations may be revisited. In particular, the requirements gathering and development phases shall be largely ongoing across multiple iterations.

The decision to treat these phases as ongoing comes from the project's central goal of fostering a community of collaborators and contributors. The team wishes to reduce the barrier to contribute to this project as much as possible. By allowing outside collaborators to provide feedback and insight into the requirements at any point, and by allowing potential contributors to develop code for any past or present iteration, the team hopes to engage these external contributors early and encourage their long-term involvement with the project, rather than asking them to withhold their potential contributions until the proper development phase, and risk losing their interest in the project. The development process was chosen with consideration of this asynchronous methodology.

Requirements Gathering

During the requirements gathering phase, the team will elicit requirements from the community using open questions, interviews, discussions, bug reports, and feature requests. In early iterations, the focus will be on gathering requirements via open questions, interviews, and arranged discussions. In later increments, focus will switch to gathering requirements via more passive, unsolicited channels: bug reports, feature requests, and spontaneous discussion within the community.

Analysis and Design

During the analysis and design phase, the team will analyse the information gathered from the community, sort it into more structured requirements, and determine which requirements are relative to the release for the current iteration. Requirements that are determined not to be part of the current increment will still be added to the backlog with low priority, allowing external contributors to contribute to those features at any time.

Development

During the development phase, the team will implement features in small increments. These will add up to a stable version of the tool with the features determined in the previous phase. The development process is described in greater detail in the "Development Process" section below.

Deployment

During the deployment phase, the addition of features to the tool will be halted (development efforts may continue concurrently, but the merging of new features will be temporarily put on hold. The team will perform dynamic black box testing on the current version of the code base. Any defects found and determined to be unacceptable for the deliverable will be fixed. When the deliverable is approved by all team members, it will be released.

Development Process

The development process will utilize Git and GitHub to track work, allow multiple developers, including external contributors, to work concurrently, and organize development and released versions of the tools.

Work Tickets

Work tickets will be tracked as GitHub Issues. Each work ticket will include a description of the feature to be added or problem to be solved, blockers or dependencies (e.g. other tickets), potential risks (if applicable), and a numeric estimate of the ticket's complexity in "story points". Each ticket will be labeled to indicate its current status.

Ticket Creation

New tickets can be created by the team at any time. These new tickets should have a "Not Ready" label placed on them, indicating they have not been reviewed and approved by the team. Other contributors can also propose tickets as either bug reports or feature requests, and will also be marked "Not Ready" until the team formalizes and approves them. Other members of the team peer review the tickets, discussing whether the description adequately describes the expected functionality, whether all blockers and risks are accounted for, and whether the complexity of the ticket is appropriate. When a team member believes the ticket is ready, they approve it. Once two team members approve the current version of a ticket, it may be labeled "Ready". Tickets that have blockers can be labeled "Ready", but should also be labeled "Blocked", indicating that they will be ready as soon as their blockers are resolved.

Development

The highest priority unblocked tickets should be developed first. When a developer wishes to work on a ticket, they change the ticket's label from "Ready" to "In Development" and assign themselves to the ticket. When work is complete, the developer should issue a pull request against the development branch of the project. Developers are responsible for resolving merge conflicts before issuing a pull request, and pull requests should therefore always be fast forwards. The easiest way to achieve this is to develop in a feature branch and rebase the changes onto an up-to-date development branch before issuing the pull request (see "Git Workflow" below). Static white box testing in the form of peer review will take place. Other developers may discuss the changes and ask questions, and the developer issuing the pull request may tweak the pull request in response to this discussion. When the same version of the pull request receives approval from two other team members, it may be merged into the development branch of the project. The ticket should then be labeled "Done".

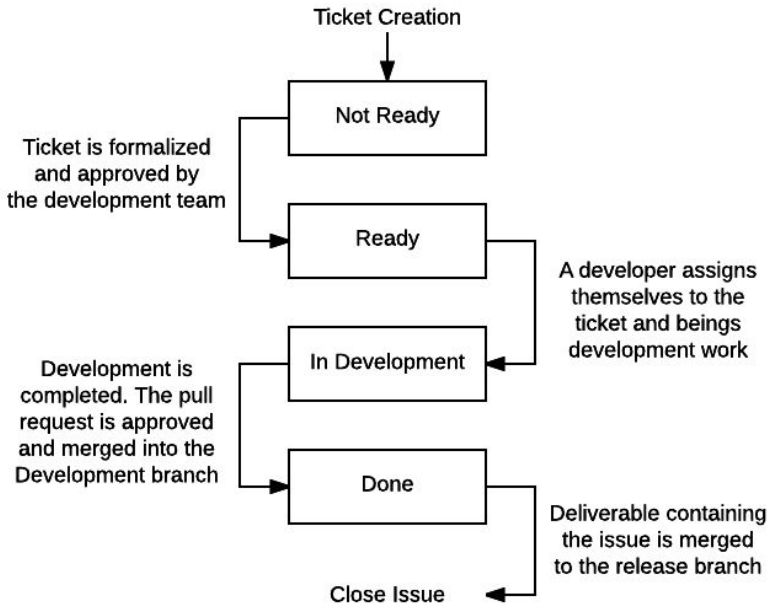

Figure 1: Ticket Lifecycle

Deployment

After all planned features for a release are completed, the development branch is frozen, that is, no pull requests are merged (pull requests may still be issued, and reviewed, but should not be merged). The team shall perform dynamic black box testing to verify the release. Once the team is satisfied that no unacceptable defects are contained in the codebase, the development branch will be merged to the release branch, the release will be tagged, and the development freeze will be lifted.

Git Workflow

The following is the recommended Git workflow for working with this project.

Preparing the dev environment:

1. Fork the Development branch of the project to your personal GitHub
2. Clone the Development branch from your personal GitHub to your local machine
3. Branch the development branch on your local machine to a feature branch

Developing:

1. Perform development work on the feature branch
2. Periodically pull from the project's development branch to your local development branch and rebase your feature branch onto the up-to-date development branch
3. Push your feature branch to your personal GitHub to synchronise development between machines, publically discuss implementation with colleagues, etc.

Merging:

1. Make sure you have rebased your feature branch onto an up-to-date development branch (step 2 of Developing)
2. Squash commits related to this feature into one "feature commit" (if you would like to keep granular changes, branch your feature branch before squashing)
3. Push your feature branch to your personal GitHub
4. Issue a pull request from your personal GitHub's feature branch to the project's development branch
5. If changes are necessary to get the pull request accepted, make them on your local feature branch, and push them to your personal GitHub (remember to squash your commits). GitHub will automatically update your pull request.

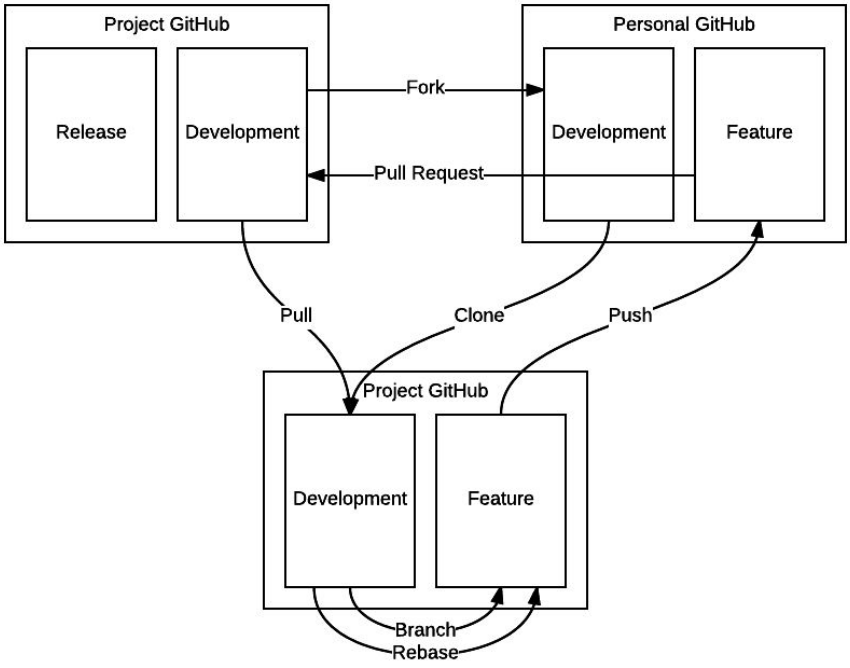

Figure 2: Git Workflow