Platforma průmyslové spolupráce

CZ.1.07/2.4.00/17.0041

Název

Automatic pull request integration

Popis a využití

- nástroj pro jednodušší hodnocení oprav domácích úkolů v jazyce Java
- výuka: pokročilá Java

Jazyk textu

- anglický

Autor (autoři)

- Jan Brázdil

Oficiální stránka projektu:

- http://lasaris.fi.muni.cz/pps

Dostupnost výukových materiálů a nástrojů online:

- http://lasaris.fi.muni.cz/pps/study-materials-and-tools

Contents

Chapter 1

Introduction

In software development it is good to know if all parts of a project work as desired. It is also good to know if all the parts work together. When many developers are collaborating on a project there is usually a need for code review, especially in open source projects. The review process can be complicated and time consuming.

Goal of this work is to facilitate this process by development of a tool that can reduce the time needed to review code changes. When new changes are automatically tested a reviewer does not have to spend his time on the wrong changes. And if the test feedback is automatically given to the author of the changes he can fix all issues immediately without the need to wait for the review.

This thesis examines tools for code review and existing tools for automatic testing of proposed code changes. It also describes development of a tool for automatic testing of GitHub Pull Requests in Jenkins CI and for reporting the results back to GitHub. The tool is in a form of Jenkins plugin and is named *GitHub Pull Request Builder*.

As this work is done in cooperation with Red Hat the focus was on open source technologies and the resulting tool is released under open source license.

1.1 Structure of This Thesis

The thesis is divided into five chapters.

This is *Chapter 1*. There is introduction to this thesis, its goals and structure.

In *Chapter 2* important technologies are introduced and basics of Continuous Integration are explained. In this chapter tools for code review and existing tools for automatic Pull Request testing are pre-

sented.

In *Chapter 3* requirements for a new tool for automatic Pull Request testing are described and existing JBoss internal tool is analyzed. There is also deeper analysis of GitHub and Jenkins CI in this chapter.

Chapter 4 describes implementation of the GitHub Pull Request Builder. Library for the GitHub API that was used for communicating with GitHub is described. Also specifics of the Jenkins CI plugin development are described. Lastly architecture and function of the GitHub Pull Request Builder are presented.

In the last *Chapter 5,* there results of this thesis are summarized.

Chapter 2

Tools and Technologies

Technologies and tools that are important for this work are described in this chapter. First of all a source code management system called Git is introduced. Continuous Integration is explained afterwards and Jenkins CI, one of the systems for Continuous Integration, is introduced. After that code review tools for git are described. Existing tools for the preintegration testing are presented in the end of this chapter.

2.1 Git

Git is a version and source code management system. Unlike other SCM systems (e.g. CVS, Subversion and others), git is distributed and supports heavy branching. Git was designed for open source projects by Linus Torvalds.

One of main advantages of git is in its branching model. Creating, merging and working with branches is fast and easy. Branches can be used for easy switching of context, for example in situations where developer works on a new feature and is asked to quickly fix newly discovered bug. Separation of development and stable branches and keeping branches with older but still supported version of project is also possible. Developing in multiple feature branches not interfering with each other and merging only finished work and other techniques can be done using branches too.[1] [1]

The fact that git is distributed has also many advantages. Every git user clones the whole repository instead of checking out just the current state. This can be seen as disadvantage as initial clone can

1. Example of git branching model is on Figure C.1

take longer then when using e.g. SVN, however this operation is usually done only once. Also the cloned repository is slightly bigger in size but it is effectively backed up at every developer's computer and as most of the operations are done locally, they are really fast. [2] Also the distributed nature of git enables projects to use various kinds of workflows.

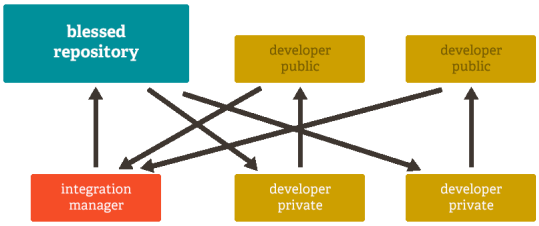

Figure 2.1: Git workflow with integration manager. [3]

A centralized workflow can be used when every developer has rights to contribute to a project directly. In world of open source other workflows are common. In one such workflow (Figure 2.1) every developer has his own repository and their changes are put together by *integration manager* who pulls the changes from developers' repositories and integrate them in to one canonical repository (also called *blessed repository*). The developers then pull all changes from this repository.

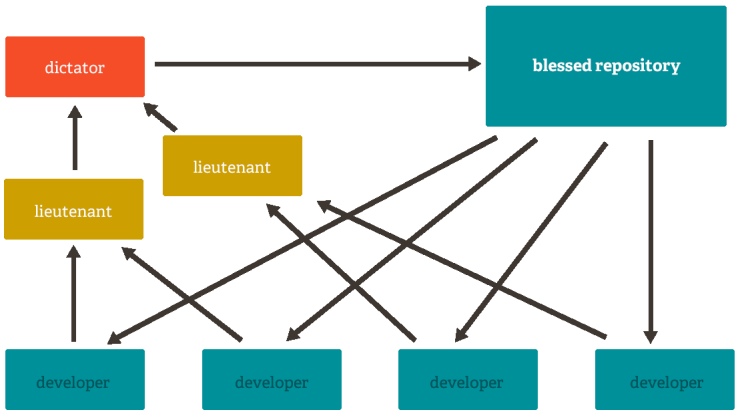

Figure 2.2: Git workflow in Linux kernel. [3]

Linux kernel is using another workflow (Figure 2.2) where people participating on the project make a hierarchy of responsibility. There are people called *lieutenants* who are responsible for their part of project and they gather relevant changes from developers. Person called a *dictator* then gathers changes from the lieutenants and pushes them in the canonical repository. [3]

Consistency checks are another git advantage. Git is using SHA-1 hashing to mark its objects like data blobs or commits. Every commit's information like author, timestamp, changes and commit's parent(s) hashes are hashed together and the resulting hash is identifier of the commit. If you know the commit hash is correct you can verify your whole git history if it is correct. [4]

Git branches are identified by a *refspec*. For example refspec of master branch is `refs/heads/master`. Refspec can define namespace. For example QA team can have branches in refspec:

```
refs/heads/qa/*
```

The QA team can then specify that they want to download only branches in their namespace. [5]

2.2 Continuous integration

Process of software integration is problematic. As long as a project is small (e.g. only one person works on it) integration is not big problem. However as the project grows, more people are working on it and it is becoming more complex, integration becomes essential. Common behavior in software development is that integration of individual components of the project happens late in the development process. That brings a lot of disadvantages.

The project is often in broken state because nobody is forced to test that everything works together. Only separate parts are tested. Issues that will be discovered during an integration are often in the project for long time and fixing of these issues can be time consuming and expensive. The process of integration itself is not tested and can be error-prone and bring another issues. Estimation of time needed for integration and deployment of the project is very difficult.

Continuous Integration is a practice that started as part of extreme programming. Continuous Integration works on presumption

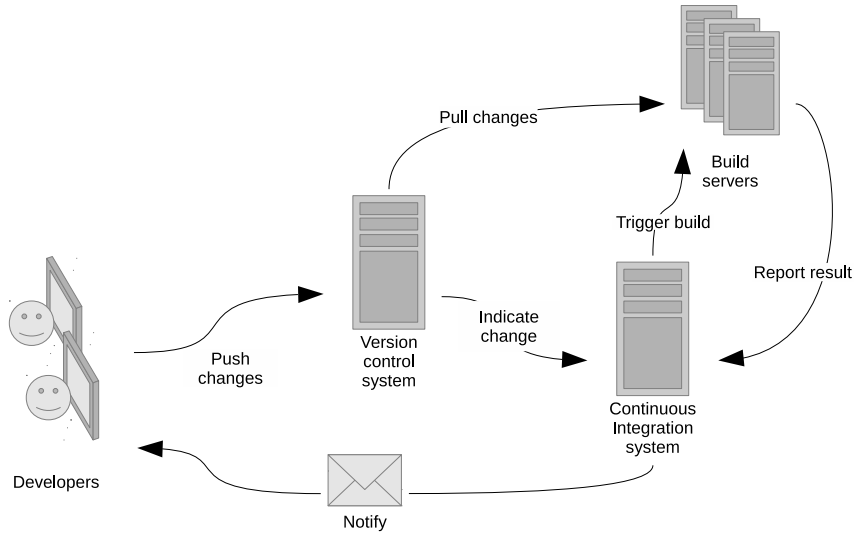

Figure 2.3: Continuous Integration workflow.

that if it is needed to prevent difficulties which rise from late integration, integration must be done often. In Continuous Integration that means integration should be done continuously – with every new change.

To be able to achieve this, specific requirements have to be met. The source codes need to be in a version control repository and developers must commit small changes regularly. All integration processes must be automated. When a new commit is pushed into repository, automated build of the sources must start. Then every part of the project must be tested by unit tests and integration tests. Good code and functionality test coverage is a must. Whenever a part of the process fails, it must be immediately reported to developers and they have to fix the issue promptly.

When all the requirements are met, Continuous Integration brings benefits. Developers have regular feedback about quality of the code, the project is in runnable state most of the time and issues are discovered soon and are easier to locate. [6]

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integrations per day. Each integration is

verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly." [7]

When there are many people contributing to open source project, there is usually need for validating patches sent by community. The patches usually need to complain to specific rules (e.g. style rules) and must not break the application. Human validation is usually long and risk-prone process and in the end when the patch is merged to the project it can still break the project. It is a good idea to automate this process as well and use Continuous Integration even on the patches. The same process that is used for Continuous Integration can be performed on the source base with the patch applied. If the process passes, developer can then review the patch.

There are numerous tools and systems for Continuous Integration and they are easy to install and use. [8] There are a lot of commercial and several open source systems for Continuous Integration. Most common open source are *Jenkins CI* and *CruiseControl*. Also *Travis CI* is a free and open source service for Continuous Integration.

2.3 Jenkins CI

Jenkins is an open source system for Continuous Integration. It supports automated building and testing of projects. Jenkins, in the past known as Hudson, was created by Kohsuke Kawaguchi who also created a number of plugins and libraries around the project. Jenkins is easily extensible with plugins. Currently[2] there is over 800 plugins for many various tasks like source code management, build triggering, notifying and reporting, slave launching and controlling, uploading artifacts, integration to other sites and tools and a lot more.

Jenkins is written in Java but it also supports other languages and technologies including Ruby, PHP, .NET, Groovy etc. Jenkins is easy to start using in a few minutes. Jenkins is also very well supported and has big community. The development is fast and new releases are coming out weekly.

2. 27. 2. 2013

More information about Jenkins architecture is in section 3.4 and creation of Jenkins plugin is described in section 4.2.

2.4 Code Review Tools For Git

Tools for code review that work with git are presented in this section. The possibilities of integration of these tools with systems for Continuous Integration are examined.

2.4.1 Gerrit

Gerrit is an open source tool for hosting repositories and for code review. Developers can push code changes into special branches when using Gerrit. The change is then shown in web UI waiting for a review. A reviewer then checks the change and can accept the change or abandon it.

Gerrit is a fork of *Rietveld*, another tool for code review. Both of them are good for centralized repositories and Gerrit supports access control lists. Gerrit is written in Java and provides API for plugin development. There are Jenkins plugins for integration with Gerrit making automatic testing of changes possible. [9]

2.4.2 GitHub

GitHub is a service for hosting git repositories. GitHub has a web code browser and allows commenting on the code. Users can easily create their own copy of project repository by using fork button. Then they can modify the code in the forked repository and promote the changes to the original project repository as a Pull Request. Then project maintainer can verify the change and merge the Pull Request into the project code. GitHub also supports issue tracking and wiki pages.

GitHub is not open source but open source projects can be hosted there for free. For other projects GitHub provides paid private repositories and offers enterprise GitHub for in-house installations. GitHub is very popular – it has over 3 millions of users. [10]

GitHub provides good REST API allowing even more operations

then which can be done in web UI. There are tools allowing testing of Pull Requests like BuildHive and Travis CI.

2.4.3 GitLab

GitLab is a code browser and a code review tool inspired by GitHub. It is an open source project with a lot[3] of contributors. It is in rapid development – new versions are released monthly. GitLab used *Gitolite* for repository hosting but has switched to their own system called GitLab-shell.[4] [11, 12]

GitLab is aiming for company use cases. It does not have forks and Pull Requests but provides functionality called Merge Requests – when a development branch is ready for integration a developer can create Merge Request indicating that it is ready for review and merge into the master branch. Reviewers can comment and vote on the Merge Request and if merge can be done without conflicts it can be done by clicking on one button.

GitLab is written in Ruby. It provides REST API for basic operations. There is a Jenkins plugin for integration with GitLab but it supports only testing new commits in specified branch not testing merge requests. [13]

2.4.4 Other Tools For Code Review

Barkeep is a simple and easy to use open source code review system with support for inline comments. It provides simple API that can be used for reporting from Continuous Integration systems. [14]

Phabricator is a suite of web application with support for code review, code browsing, issue tracking and many other features. It is open source and it provides API that can be used for integration with Continuous Integration systems. [15]

Review Board is an open source tool for code review with web API. There is also Jenkins plugin for integration with Review Board. [16]

3. From 26. 3. 2013 to 26. 4. 2013 there were 67 contributors
4. At 22nd Mar 2013 GitLab version 5.0 was released without Gitolite support [11]

2.5 Tools for Automatic Pull Request Testing

Existing tools for Continuous Integration which supports GitHub and automatic testing of Pull Requests are presented in this section. These tools are *Travis CI* and *BuildHive*.

2.5.1 Travis CI

Travis CI is a hosted service for Continuous Integration. It aims to enable building of projects hosted on GitHub. Travis supports numerous different programming languages like C, Haskell, Java, PHP, Ruby etc. Travis is free for Continuous Integration of public GitHub repositories. It also supports building and testing of Pull Requests.

To start using Travis, it is needed to log in with GitHub account and select project repository. This will activate GitHub hooks on the repository. These steps are easily done by only a few clicks. Next step is to add `.travis.yml` file with Travis build configuration to the repository. If the file is missing Travis will assume that the project is in Ruby and will use default configuration.

However Travis has limitations: the build time is limited [17] and build environment is only Linux (Ubuntu) or Mac (for objective-C).

Travis CI is open source[5] but final version was not yet released and developers do not recommend using it for in-house testing. [18]

2.5.2 BuildHive

BuildHive is a hosted Continuous Integration service built on Jenkins CI. BuildHive is used for Continuous Integration and for testing Pull Requests on GitHub. For open source projects BuildHive is free but for private repositories it is paid. BuildHive also provides paid Jenkins plugin for testing Pull Requests.

When user logs into BuildHive using GitHub account and selects a repository for building, BuildHive will try to automatically determine what kind of project (Maven, Ant, Bundler etc.[6]) it is and then configure the build accordingly. However the project type can not be changed and more complex job configuration is not possible.

5. From 26. 3. 2013 to 26. 4. 2013 there were around 13 contributors
6. At 26. 4. 2013 BuildHive supports 7 types of projects [19]

Chapter 3

Analysis

Because this work was done in cooperation with JBoss and Red Hat, requirements were primarily gathered there and existing approaches to preintegration testing in JBoss were analyzed. Apart from JBoss, requirements from community were also collected. All the requirements are described in first section of this chapter. Existing JBoss internal tool is described in the next section. GitHub and Jenkins CI are examined in the remaining sections.

3.1 Requirements

The most important requirement for this work is that it should comply with JBoss workflow. It is needed to analyze existing approaches for code reviewing and integration testing. An access control is one of main requirements. It is necessary to ensure that no unauthorized person could run a potentially dangerous code on corporate machines. User whitelist is used in existing solution. Changes made by users on the whitelist can be automatically tested. One of the requirements is to extend this with functionality to test a change without the need of adding the author of the change to the whitelist. In case where users in the whitelist are members of a GitHub organization, the maintenance of the whitelist can be a duplicate job to the maintenance of users in the GitHub organizations so an ability to whitelist whole GitHub organization is required. Administration of the whitelist should be easy and adding new user to whitelist should be possible to do from GitHub. On the other hand if the Continuous Integration system is secured (e.g. tests are running in a sandbox environment) there is no need for access control.

Next important requirement is to automate the process of test-

ing the changes and to run the tests as swiftly as possible. Besides other things it is needed to rerun the tests when the changes are updated. Another requirement is to report the results of the test back to GitHub using the commit status feature. Also the Continuous Integration system can be inaccessible from the internet (e.g. behind firewall) or access to the Continuous Integration system can be protected and the result of this work needs to be able to handle both cases. Related to this is that private Continuous Integration system instance can have public mirror where results of a build are published and URL to this public results should be reported to GitHub too.

Another requirement is to support different project policies regarding the severity of failed tests (e.g. if only failed build is considered as failure and failed tests are ok or if failed tests are also considered as failure) and to support automatic rejection of the changes when the testing of them failed. Also the build needs to get proper information about the changes like Pull Request number and target branch for the changes.

It is also required to support enterprise GitHub in addition to github.com. Last but not least the result should be well documented to facilitate easy setup and use.

3.2 JBoss status

Jenkins CI is commonly used for testing and Continuous Integration in JBoss. Some projects like WildFly Application Server have their own approach to preintegration testing. WildFly source codes are hosted on GitHub and Pull Requests testing is done on internal Jenkins server called Lightning. Lightning is using Java application called *Pull Player* for triggering builds of Pull Requests and results are then published on public Jenkins server.

3.2.1 Pull Player

Pull Player is a standalone Java application that checks GitHub repository for Pull Requests and schedule building and testing of these Pull Requests on Jenkins server. Pull Player is controlled by Pull Re-

quest comments and is sending reports of its progress also as the Pull Request comment. Pull Player divides GitHub users to three categories: admins, whitelisted and others. Building of Pull Request starts only when author of the Pull Request is whitelisted. Whitelisted users can also restart the build by commenting to the Pull Request with "retest this please". Admins are allowed to add authors of Pull Request to whitelist by commenting with "ok to test". The Pull Player process is executed regularly by using cron.

When the Pull Player is executed, it checks for open Pull Requests on specified GitHub repository. For every Pull Request Pull Player iterate over the Pull Request's comments and updates internal information about the Pull Request. These information represent state of the Pull Request – whether building of the Pull Request should be started, already was or if Pull Player is waiting for user input. After all comments are read and if all conditions are met new build in Jenkins is triggered.

Pull player is communicating with Jenkins via web API. It finds information about a build by retrieving information about all builds and searching for the requested one.

Information about completed jobs are stored in files.

3.3 GitHub

GitHub is introduced more closely in this section. At first there is brief insight into GitHub users and organizations and then Forks and Pull Requests are described. Then GitHub hooks are presented in the end GitHub API is described.

GitHub recognizes two types of users. First one is regular user and organization is the second one. Both of them can have public and private repositories. Regular users can add collaborators to their repository. Collaborators are allowed to push directly into the repository and in case of private repository being a collaborator is the only option how to get access to it. Organizations can create teams from regular users. The teams can have Pull, Pull & Push or Pull, Push & Administrator rights for repository owned by the organization.

3.3.1 Forks and Pull Requests

Figure 3.1: Buttons on GitHub for creating Pull Request and Fork. [20]

When someone wants to contribute to a GitHub project, he can use *Fork*. Forking a repository creates a copy of that repository available for modification. Changes in the forked repository (e.g. branch with new functionality or bug fix) can be then proposed to the original project for merge. To facilitate this, GitHub provides *Pull Requests*. Pull Request indicates an intention to merge the head branch to the chosen base branch. The Pull Request creator must select source (head) and target (base) repositories and corresponding branches. After the Pull Request is created owner of the target repository is notified about a new Pull Request.

Figure 3.2: GitHub dialog for creating of Pull Request. [20]

Opened Pull Requests for a repository can be browsed on GitHub. Pull Requests can be commented, merged and closed. If the Pull Request can be merged without conflicts it can be done even from GitHub web interface. If there are conflicts, they can be resolved and merged with standard git merging process and when pushed to GitHub the Pull Request is closed automatically.

Content of a Pull Request is stored in GitHub in special refspec:

```
remotes/origin/pr/*/head
```

Where `*` is Pull Request id. If the Pull Request can be automatically merged the merged state is stored in refspec:

```
remotes/origin/pr/*/merge
```

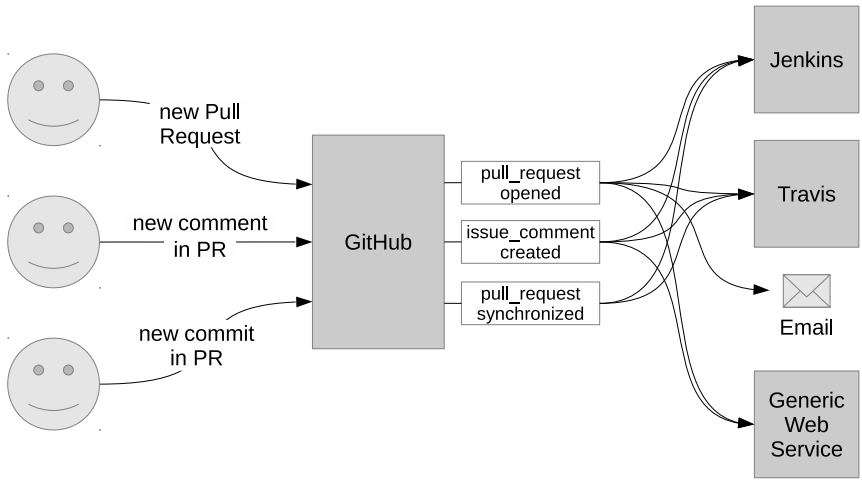

Figure 3.3: GitHub hooks.

In September 2012 GitHub introduced *Commit Status API* [21]. Commit status can be used for commit evaluation with external tools. External tool (e.g. Jenkins) can checkout the commit run tests on it or do an analysis with it and then report the results back to GitHub. After that it is easy to see if the evaluation was successful or not in Pull Request view.

3.3.2 Hooks

GitHub hooks are used for communication with other services to let them know when an important event happens in a GitHub repository. GitHub supports many different services[1] including general web service where it is able to send `POST` request with information about the event to specified URL address.

Among the events GitHub hooks can handle are `issue_comment` which fires when somebody create comment on issue or Pull Request and `pull_request` which fires when Pull Request is opened, closed or a new commit occurs in it. [23]

1. 112 services at 2013-03-23 [22]

3.3.3 API

GitHub uses REST API for making its functionality available for external tools. GitHub API is now in version 3. API is accessible using HTTPS protocol on address `https://api.github.com/`. Pagination with default 30 results per page is used when accessing list of entities. [24]

GitHub provides anonymous access to all publicly available data, however unauthenticated requests are limited to 60 requests per hour. [25] There exist three ways of authentication in GitHub API: using basic HTTP authentication with username and password, using OAuth2 access token sent either int request header or as a parameter or using OAuth2 secret client ID and key.

Part of the API can be used for testing Pull Requests and reporting results back to GitHub.

To list all Pull Requests on a given repository, following `GET` request can be used:

```
/repos/:owner/:repo/pulls
```

Where `:owner` is username or name of organization and `:repo` is name of repository.

To retrieve a specific Pull Request by id, following `GET` request can be used:

```
/repos/:owner/:repo/pulls/:number
```

Where `:number` is the id of the Pull Request. To close a Pull Request, `PATCH` request is sent to the address with body containing: [26]

```
{ "state": "closed" }
```

Comments for given Pull Request are located on address

```
/repos/:owner/:repo/issues/:number/comments
```

To list all comments, `GET` request is used and to create a new comment, `POST` request is used. Its body contains [27]

```
{ "body": "a new comment" }
```

To update a commit status, following `POST` request can be used:

`/repos/:owner/:repo/statuses/:sha`

Where `:sha` is SHA hash of given commit. In the body of this request there must be field `state` (which can have one value of `pending`, `success`, `error` or `failure`) and optional fields are `target_url` and `description`. [28]

To create a new hook, following `POST` request is used:

`/repos/:owner/:repo/hooks`

with body containing name of service, array of events on which will the hook be informing and hook configuration.

3.4 Jenkins CI

Jenkins is "*a set of Java classes that model the concepts of a build system*" [29]. In the root there is class `Jenkins` which contains a tree structure of `Item`s. Items can represent various things like jobs, projects, builds and so on. Jenkins is using Stapler for mapping these Items to URL. Jenkins uses serialization into XML files to store its data.

Jenkins job can represent a particular task or step in build process. It can compile a sources and run unit tests or build a project and do related tasks like running integration testsuite, measure code quality, generate documentation etc. [30]

Jenkins provides a range of classes and interfaces for modeling every part of a build process like source code management, triggers, builders, mailers etc. These build parts can be extended and modified by using plugins. [30]

3.4.1 Stapler

Stapler is a Java library that maps objects to URLs creating an URL hierarchy according the object model. In application that uses Stapler root object of the application is mapped to URL `/`. An object accessible with `root.getProject("myProject")` is assigned to URL `/project/myProject` and method `doStuff()` on this object will be assigned to URL `/project/myProject/stuff`.[2]

2. More information about Stapler can be found on `http://stapler.kohsuke.org/`

3.4.2 Plugins for integration with git

For integration of Jenkins with GitHub, I investigated existing plugins. I am using three: *Git Plugin*, *GitHub Plugin* and *GitHub API Plugin*.

Git Plugin allows using of git as a build SCM in Jenkins. It provides functionality for all work related to bare git: repository cloning, change fetching, branch merging and others. This plugin is essential for all git-related work.

GitHub Plugin integrates GitHub with Jenkins. It creates hyperlink in Jenkins Job view to GitHub and can trigger a job when changes are pushed into repository.

GitHub API Plugin is essential for communicating with GitHub via REST API.

Chapter 4

Implementation

At first GitHub API plugin is analyzed and parts specific for development of GitHub Pull Request Builder are described and also work on improving this GitHub API plugin is presented. Process of developing Jenkins CI plugin is described after that. Then the architecture of GitHub Pull Request Builder is explained. Usage of GitHub Pull Request Builder is covered in the end.

4.1 GitHub API

GitHub API library and plugin by Kohsuke Kawaguchi is Java implementation of GitHub REST API. It provides simple programmatic interface for communication with GitHub.

There is class `GitHub` with static methods `connect` providing various ways of authentication. When user is authenticated, `GitHub` instance can be used for getting data like users, organizations and repositories. These can be used to operate with GitHub API. GitHub entities are represented by relevant classes `GHUser`, `GHRepository`, `GHPullRequest` and others.

4.1.1 Architecture

Usage of the GitHub API library was not perfectly documented and as mentioned later, the functionality was not perfect either. So, it was needed to investigate the source codes for better comprehension.

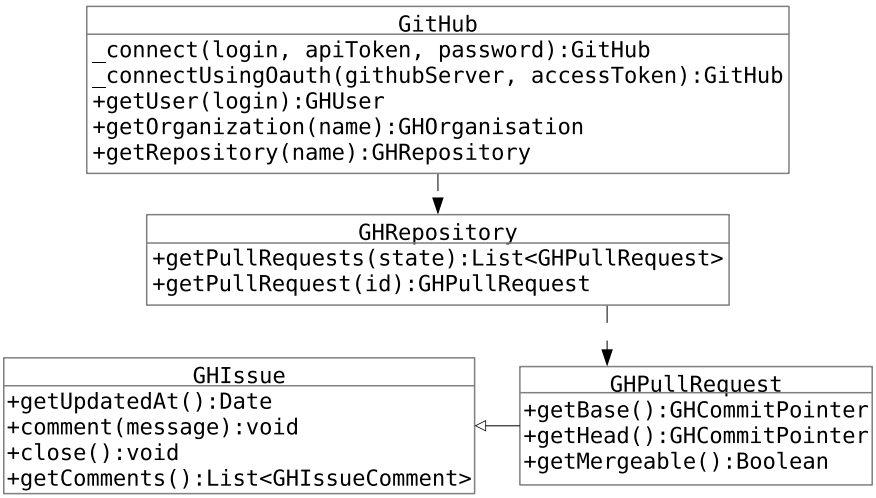

Figure 4.1: GitHub API library.

Class GitHub

This class is the central class of this library. It provides various ways of authentication using static builder methods.

Method `connect` returns `GitHub` instance which is connecting to GitHub using username and password. This method is not actually communicating with GitHub when executed, it only stores credentials for later communication.

Method `connectUsingOAuth` returns `GitHub` instance which connects to enterprise GitHub using access token. This method communicates with GitHub when executed because it tries to determine login name.

There are also other methods for connecting but they were not important for my work.

When the instance of `GitHub` class is obtained, it can be used for accessing some of the parts of GitHub API directly and others indirectly.

Using the methods `getUser` and `getOrganization` can be retrieved objects representing GitHub user or organization and using the method `getRepository` can be retrieved object representing repository.

GHRepository

This class has numerous methods to access and modify a repository in GitHub. There are two important methods For my work.

First `getPullRequests` retrieves list of all Pull Request given their state (e.g. open Pull Requests) and second `getPullRequest` retrieves single Pull Request by its id.

GHPullRequest and GHIssue

Class GHPullRequest is a representation of a Pull Request in GitHub. Because a Pull Request is basically an issue GHPullRequest is subclass of GHIssue.

These methods are important for my work: `getUpdatedAt` is used for checking if a Pull Request was changed, `comment` is used for commenting on a Pull Request, `close` is used to close a Pull Request when the build fails, `getBase` and `getHead` are used to get a commit SHA for determining base point for merging and checking if a new commit occurred in the Pull Request. Method `getMergeable` is used for checking if commit can be automatically merged and method `getComments` is used for obtaining list of comments.

4.1.2 Improvements

When I started working on my Pull Request Builder plugin, GitHub API plugin was not up to date with actual GitHub API v3. In the code there were still references to GitHub API v2, there were issues with pagination and shallow entities and parts of GitHub API were not covered.

As I needed proper functionality I have written number of patches solving these issues. Some of them were accepted by Kohsuke or at last inspired him to implement this by his own. [31]

My contribution specifically consisted of following.

I have implemented a support for pagination when retrieving issue comments and fixed referencing to author of the comment (see appendix D.1).

In API v3 GitHub came with *shallow entities*. For example in Pull Request entity there is a reference to user who made that Pull Re-

quest. But in previous version of GitHub API there were only two ways how to do this reference – either by an identificator like username or by using whole user entity. Shallow entities allow usage of reference which provides more relevant information about user but without the need to supply also the irrelevant information. So user shallow entity contains only username, GitHub URL and avatar URL. When other information about the user are required, the whole user entity has to be retrieved.

I have implemented one way of handling shallow references by using a class hierarchy. However Kohsuke though that it would be easier for client to have only one class for each entity and missing data would be populated lazily. So he used my patch and updated it accordingly. [31] I have also removed old GitHub API v2 data fields and replaced them with new GitHub API v3 data fields (see appendix D.2).

After my Pull Request Builder plugin was released, I found out there are other issues in GitHub API plugin. One of them was that authentication header was not sent sometimes when it was needed. After discussion with Kohsuke I implemented a solution in which the authentication header is being sent in every request, not only when needed (see appendix D.4).

In order to add more functionality to my Pull Request Builder plugin like generation of access token and support for GitHub hooks, I needed to implement parts of GitHub API that was not covered yet in Kohsuke's GitHub API plugin.

One of them was method for creating OAuth2 access token and related classes for representing entities returned via the GitHub API (see appendix D.6).

The other was implementation of class representing data that GitHub send in its hook (see appendix D.7).

And I have also made other smaller changes (see appendix D.3, D.5 and D.8).

During the time I was working on my plugin and contributing to Kohsuke's GitHub API library I earned his trust and he allowed me to push my changes directly into his repository.

4.2 Jenkins plugin

4.2.1 Plugin API

Jenkins provides *extension points* for extension by plugins. Extension points are Java interfaces or abstract classes. In Jenkins core there is over 100 extension points for plugins to use and plugins can define their own extension points. By implementing these extension points

Implemented extension points can be annotated with annotation `@Extension`. This annotation marks annotated class for automated localization by Jenkins.

4.2.2 Descriptors and Describable

For storing metadata about a configurable (`Describable`) instance Jenkins is using `Descriptor`s. `Describable` is interface used for marking that a class is described by `Descriptor`. `Descriptor` is abstract class and each `Describable` class has a singleton implementation of `Descriptor`.

`Descriptor` is used to store system-wide configuration and to help to render the configuration form. It is serialized to XML and can persist any data by storing them in fields.

4.2.3 Extension points for build triggering

Jenkins has a lot of extension points, however only few of them are needed for triggering a build.

Most important class is `Trigger` that represents a build trigger. A cron-like string can be associated to it for periodical execution of its method `run()`. `Trigger` is configurable by job and can store job specific configuration.

Class `Cause` class keeps information about a reason why a build was started. It also renders the cause in the UI and can be used for providing useful information.

`RunListener` is Jenkins wide listener that is notified about all builds that happens in Jenkins. It is notified when build is started, completed, finalized (everything related to the build is persisted) and before a build is deleted.

Stapler is mapping `UnprotectedRootAction` class to Jenkins root URL according the `getUrlName()` method. If the URL name is `someName`, the instance of `UnprotectedRootAction` is mapped to URL like:

```
http://jenkins.server.com/someName
```

Difference against `RootAction` is that `UnprotectedRootAction` can be accessed without authentication.

4.2.4 Project and Build

Class `Project` is representation of a buildable Jenkins job. It holds all information about the job and can be used schedule a new build of the job.

Class `Build` is representation of a build. It holds all information about the build such as the `Cause` and results of the build.

4.2.5 Plugin Release

Jenkins supports an easy way of releasing plugins via maven. Developers are encouraged to host their plugins in Jenkins community repositories on GitHub. When plugin is ready for release and changes are pushed to the GitHub repository it can be published by the following command:

```
mvn release:prepare release:perform
```

This will ask developer what version it is and what version will be next. Then build, tests and packaging will be started and if it all passes the package will be uploaded to Jenkins download center. After some time the new version (or new plugin) will appear in Jenkins wiki on Plugins page[1] and in the Jenkins web console in update Center. [32]

1. `https://wiki.jenkins-ci.org/display/JENKINS/Plugins`

4.3 GitHub Pull Request Builder

4.3.1 Cron and Hooks

One of the requirements was to start the builds as soon as possible after a new change in Pull Requests occurred. When using GitHub hook, it is possible to react to a change in Pull Request in seconds. GitHub hooks also provide currently modified data so the amount of needed communication with GitHub via API is reduced.

Unfortunately GitHub hooks can not be used in cases were Jenkins is not accessible from internet (e.g. hidden behind firewall). Hooks can be also created only by an owner (or user with administrator rights) of the repository. Therefore I started with second option and implemented support for GitHub hooks.

The second option is to use regular polling. Jenkins class `Trigger` supports timed running of its method `run()` by using cron-like string to configure the timer. When the timer is fired GitHub is polled for changes. However this approach brings problem how often to configure the polling. On one side, frequent polling can increase load on the Jenkins server and the GitHub API, on the other side there could be undesirable delays. With regards to this I added configuration option to set the frequency for every job separately.

The cron activated polling is used also when the usage of GitHub hooks is enabled. However this check is only performed once for Jenkins uptime and is used for fetching changes that occurred during the time when the hooks were not listened to. Those are for example cases when new job is created for repository that already contains opened Pull Requests or when the Jenkins was shut down.

4.3.2 Architecture

Class `GhprbTrigger` is a connecting point between Jenkins and the rest of the plugin. It extends Jenkins extension point `Trigger` which is `Describable` so it holds `Descriptor` with plugin's global configuration. The Descriptor is also used to store information about Pull Requests. `GhprbTrigger` stores job specific configuration, have method for starting a new build and according to cron configuration periodically notify class `Ghprb`.

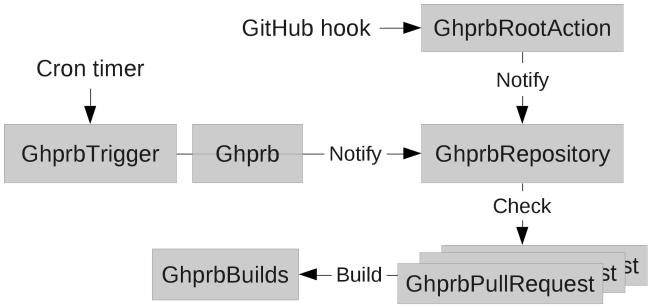

Figure 4.2: GitHub Pull Request Builder workflow.

Main purpose of `Ghprb` class is to make communication between individual parts of the plugin easier. It has builder that is used in `GhprbTrigger` for easier creation of the `Ghprb` class and ensuring that configuration is ok. The builder also sets all properties of the `Ghprb` class so it does not need to have methods that are not used by other objects. `Ghprb` provides information methods that are used for flow control. It also contains method `run()` that is periodically called by `GhprbTrigger` and which determines whether check if the repository should be initiated.

`GhprbRepository` is representation of GitHub repository in the meaning of the plugin. It has list of `GhprbPullRequest` objects which represents opened Pull Requests. It can check all opened Pull Requests in GitHub repository for changes creating new entries in the list or removing closed one. Class `GhprbRepository` is also used for communicating with GitHub repository such as creating new commit status, adding comments, closing a Pull Request or creating new hooks. It has also methods that react to incoming hook messages.

Class `GhprbPullRequest` represents GitHub Pull Request and it contains logic that determines whether new build should be started or not. It remembers state of the Pull Request and when asked to check if the build should start, it compares new events that occurred in the meantime from last check. If a new build should start, object of `GhprbBuilds` class is notified.

`GhprbBuilds` class handles starting new builds and notification about build progress. When a new build is about to start new in-

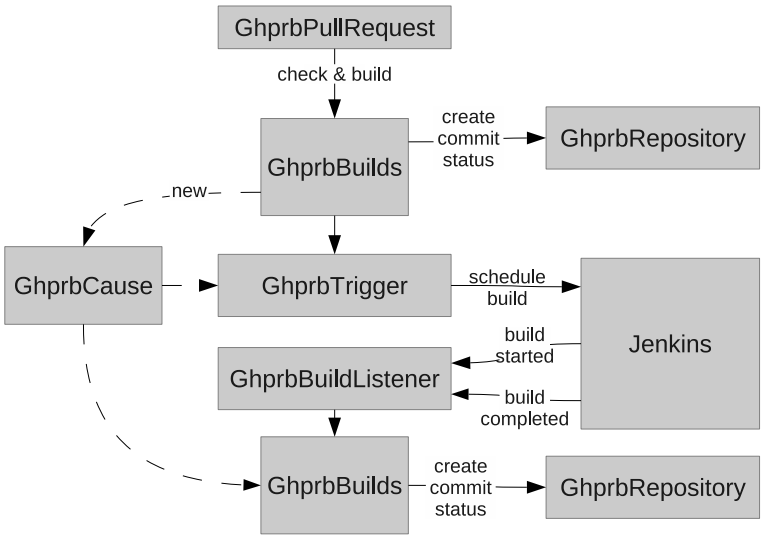

Figure 4.3: Build triggering and reporting.

stance of `GhprbCause` is created and handled to `GhprbTrigger`. `GhprbCause` is an extension of Jenkins class `Cause` and holds information about the Pull Request needed for the build. `GhprbTrigger` takes data from the `GhprbCause` and schedules a new build in Jenkins.

When a build is started or completed, `GhprbBuildListener` is notified and it checks if the build is started by `GhprbBuilds`. If so, `GhprbBuilds` retrieves the `GhprbCause`, check status of the build and reports progress or results to the GitHub.

Class `GhprbGitHub` is wrapper around GitHub API class `GitHub`. It ensures connection to GitHub. Because `GitHub` builder method can throw an exception when there is a problem with connection, this class tries to build the `GitHub` every time it is asked for it and when successful it stores it and reuses it in following interactions. `GhprbGitHub` also can check whether a user is a member of specified organization.

Class `GhprbRootAction` is listening for GitHub hooks. This class is mapped using Stapler to a web address like:

`http://jenkins.server.com/`**`ghprbhook`**

When HTTP request arrives it is parsed and then `GhprbRootAction`

determines which job should be notified.

4.4 Usage

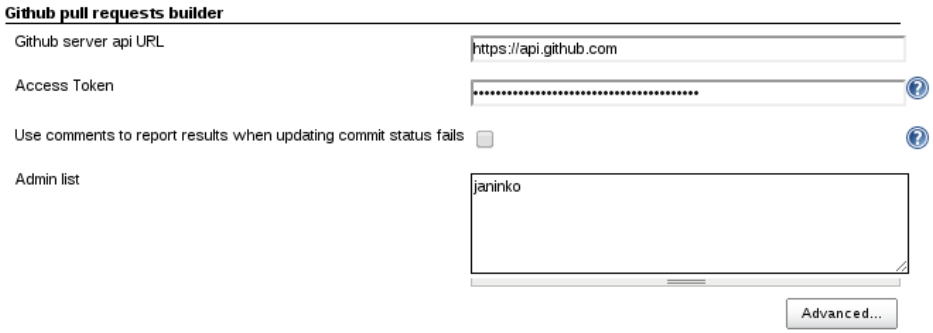

Figure 4.4: GitHub Pull Request Builder global Jenkins configuration.

When a new Pull Request is opened in the GitHub project and the author of the Pull Request is not in whitelist, GitHub Pull Request Builder will add comment to the Pull Request asking "Can one of the admins verify this patch?". One of the admins can comment "ok to test" to accept this Pull Request for testing, "test this please" for one time test run and "add to whitelist" to add the author to the whitelist. If an author of a Pull Request is whitelisted, adding a new Pull Request or a new commit to an existing pull request will start a new build automatically and commenting on the Pull Request with "retest this please" will rerun the build.

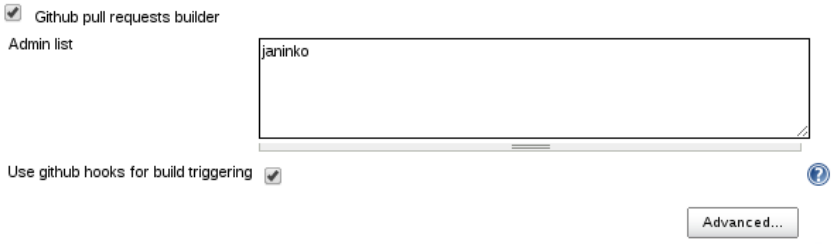

Figure 4.5: GitHub Pull Request Builder Jenkins job configuration.

A new build can also be started with a comment: “retest this please”.

When a building of Pull Request is finished the result is reported to GitHub and can be seen on the Pull Request page as shown of Figure 4.6.

Figure 4.6: Report of successful build in GitHub. [20]

Chapter 5

Conclusion

Goal of this work was to develop a tool that integrates Jenkins CI with code review and collaboration tool operating on top of git. I have developed a plugin for Jenkins CI which automatically builds GitHub Pull Requests named GitHub Pull Request Builder. The plugin has over 1000 installations[33] and is also used by JBoss.org projects like Hibernate and GateIn.

The plugin was compared to existing tools for building Pull Request. GitHub Pull Request Builder and BuildHive was very similar in time needed to trigger a build and the build started in seconds. However BuildHive did not use commit status to report the results but instead added comment to the Pull Request. Also other disadvantages of BuildHive are described in section 2.5.2. In some cases Travis CI build started after almost three hours and building bigger projects like WildFly Application Server was impossible because of timeouts.

Git tools for code review were introduced in section 2.4. Jenkins CI architecture is described in section 3.4 and creation of Jenkins plugin is described in section 4.2. Existing Jenkins plugins for integration with git and GitHub were identified in section 3.4.2. In section 3.2 current JBoss approaches to integration of development branches were analyzed.

During the development of the GitHub Pull Request Builder it was needed to contribute to other open source project – GitHub API plugin. Communication with the owner of the GitHub API was efficient and all contributions were accepted.

The GitHub Pull Request Builder is released under open source MIT license on GitHub[1] and documentation is available both in the

1. `https://github.com/janinko/ghprb`

GitHub repository and on Jenkins wiki[2].

Further improvements were requested by the community including implementation of blacklist and ability to rerun tests when target branch of Pull Request is updated.

I believe I have completed all the points that were requested in this thesis and I hope my plugin will continue to be helpful to the software development community.

2. https://wiki.jenkins-ci.org/display/JENKINS/GitHub+pull+request+builder+plugin

Bibliography

[1] *Git - Branching and Merging*. 2013. [visited 05/07/2013] available from: http://git-scm.com/about/branching-and-merging.

[2] *Git - Small and Fast*. 2013. [visited 05/07/2013] available from: http://git-scm.com/about/small-and-fast.

[3] *Git - Distributed*. 2013. [visited 05/07/2013] available from: http://git-scm.com/about/distributed.

[4] TORVALDS, Linus. *Linus Torvalds on git*. May 2007. [visited 05/08/2013] available from: https://git.wiki.kernel.org/index.php/LinusTalk200705Transcript.

[5] CHACON, Scott. *Pro Git*. Praha: CZ.NIC, 2009, p. 263. ISBN: 978-80-904248-1-4.

[6] DUVALL, Paul, MATYAS, Steve, GLOVER, Andrew. *Continuous Integration - Improving Software Quality and Reducing Risk*. Upper Saddle River, NJ: Addison-Wesley, 2007, p. 283. ISBN: 03-213-3638-0.

[7] FOWLER, Martin. *Continuous Integration*. May 2006. [visited 04/17/2013] available from: http://www.martinfowler.com/articles/continuousIntegration.html.

[8] HUMBLE, Jez, FARLEY, David. *Continuous Delivery*. first. Upper Saddle River, NJ: Addison-Wesley, 2010, p. 463. ISBN: 978-032-1601-919.

[9] *Gerrit - Background - The history behind Gerrit Code Review*. Nov. 2011. [visited 08/14/2013] available from: https://code.google.com/p/gerrit/wiki/Background.

[10] SANHEIM, Rob. *GitHub - Three Million Users*. Jan. 2013. [visited 05/08/2013] available from: https://github.com/blog/1382-three-million-users.

[11] ZAPOROZHETS, Dmitriy. *GitLab 5.0 Release, Standing on Its Own Two Feet*. Mar. 2013. [visited 04/08/2013] available from: http://blog.gitlab.org/gitlab-5-dot-0-has-been-released/.

[12] ZAPOROZHETS, Dmitriy. *GitLab Without Gitolite*. Feb. 2013. [visited 03/15/2013] available from: http://blog.gitlab.org/gitlab-without-gitolite/.

[13] *Jenkins - Gitlab Hook Plugin*. Nov. 2012. [visited 08/14/2013] available from: https://wiki.jenkins-ci.org/display/JENKINS/Gitlab+Hook+Plugin.

[14] *Barkeep - the friendly code review system*. 2013. [visited 05/08/2013] available from: http://getbarkeep.org/.

[15] *Phabricator*. 2013. [visited 05/08/2013] available from: http://phabricator.org/.

[16] *Review Board - Take the pain out of code review*. 2013. [visited 05/08/2013] available from: http://www.reviewboard.org/.

[17] *Travis CI - Configuring your Travis CI build with .travis.yml. Build-Timeouts*. 2013. [visited 03/18/2013] available from: http://about.travis-ci.org/docs/user/build-configuration/#Build-Timeouts.

[18] *What is Travis CI. We Are Not Done Yet*. 2013. [visited 05/08/2013] available from: https://github.com/travis-ci/travis-ci/tree/3cc74a80e4df1b070e9290ae633d44efee3d255f#we-are-not-done-yet.

[19] *BuildHive templates*. 2013. [visited 04/26/2013] available from: https://buildhive.cloudbees.com/template/.

[20] *GitHub*. 2013. [visited 04/08/2013] available from: https://github.com/.

[21] SANHEIM, Rob. *GitHub - Commit Status API*. Sept. 2012. [visited 03/23/2013] available from: https://github.com/blog/1227-commit-status-api.

[22] *GitHub services*. 2013. [visited 03/23/2013] available from: https://github.com/github/github-services/tree/4b52db951868827f1d2b6ddc49481ef966011058/lib/services.

[23] *GitHub API - Repo Hooks API*. 2013. [visited 03/23/2013] available from: http://developer.github.com/v3/repos/hooks/.

[24] *GitHub API - API v3. Pagination*. 2013. [visited 03/23/2013] available from: http://developer.github.com/v3/#pagination.

[25] *GitHub API - API v3. Rate Limiting*. 2013. [visited 03/23/2013] available from: http://developer.github.com/v3/#rate-limiting.

[26] *GitHub API - Pull Request API*. 2013. [visited 03/23/2013] available from: http://developer.github.com/v3/pulls/.

[27] *GitHub API - Issue Comments API*. 2013. [visited 03/23/2013] available from: http://developer.github.com/v3/issues/comments/.

[28] *GitHub API - Repo Statuses API*. 2013. [visited 03/23/2013] available from: http://developer.github.com/v3/repos/statuses/.

[29] *Jenkins Wiki - Architecture*. Oct. 2012. [visited 05/08/2013] available from: https://wiki.jenkins-ci.org/display/JENKINS/Architecture.

[30] SMART, John Ferguson. *Jenkins - the Definitive Guide*. first. Sebastopol, Calif: O'Reilly Media, 2011, p. 380. ISBN: 978-144-9305-352.

[31] KAWAGUCHI, Kohsuke. Sept. 2012. [visited 05/08/2013] available from: https://github.com/kohsuke/github-api/pull/17#issuecomment-8320566.

[32] *Jenkins Wiki - Hosting Plugins*. May 2013. [visited 05/08/2013] available from: https://wiki.jenkins-ci.org/display/JENKINS/Hosting+Plugins.

[33] *Jenkins - GitHub pull request builder plugin*. May 2013. [visited 08/13/2013] available from: https://wiki.jenkins-ci.org/display/JENKINS/GitHub+pull+request+builder+plugin.

[34] DRIESSEN, Vincent. *A successful Git branching model*. Jan. 2010. [visited 05/08/2013] available from: http://nvie.com/posts/a-successful-git-branching-model/.

Appendix A

Dictionary

Cron – Cron is tool for periodical execution of tasks.
GitHub – GitHub is a hosting service for git repositories.
GitHub hooks – GitHub hooks automatically informs other services about changes in GitHub repository.
Gitolite – Tool for hosting git repositories.
JBoss – JBoss is a division of Red Hat that specializes in open source middleware software.
OAuth2 – OAuth2 is an open standard for authorization of client application.
Pull Request – GitHub way how to propose code changes to repository.
Red Hat – World's leader in open source software development and support.
Refspec – Identifier of git branch.
REST – Representational State Transfer is machine to machine communication protocol commonly used by web based servers.
SCM – Source code manager.
SHA-1 – Secure Hash Algorithm. SHA-1 is hash function which produces 160 bit hash. In git it is used as checksum for data objects.

Appendix B

Archive content

Archive name: `automatic-pull-request-integration.zip`

Folder `ghprb` contains source codes of GitHub Pull Request Builder.
Documentation is in file `ghprb/README.md`.
License is in file `ghprb/LICENCE`.
Folder `github-api` contains patches for GitHub API plugin.
`ghprb-1.8.hpi` is Jenkins plugin package.

Appendix C

Git branching model

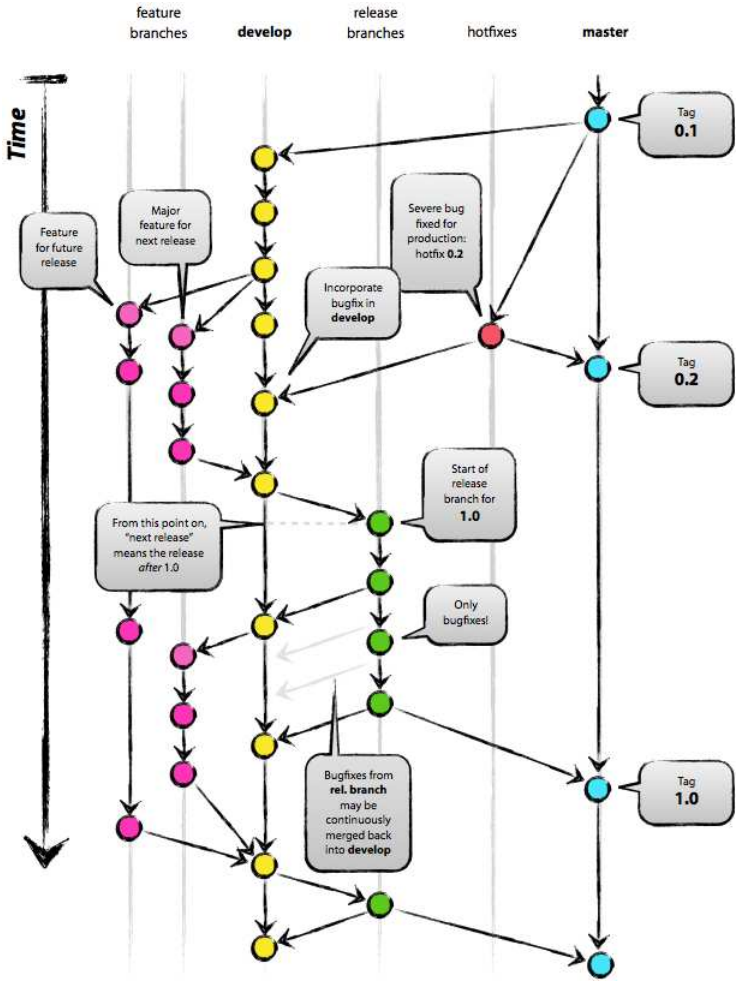

Figure C.1: Git branching model. [34]

Appendix D

List of patches for github-api plugin

D.1 Fixes for github api v3

Location in archive: `github-api/16.patch`
GitHub Pull Request URL: `https://github.com/kohsuke/github-api/pull/16`

D.2 Issues pull requests apiv3

Location in archive: `github-api/17.patch`
GitHub Pull Request URL: `https://github.com/kohsuke/github-api/pull/17`

D.3 When using lazy population, this is not deprecated

Location in archive: `github-api/18.patch`
GitHub Pull Request URL: `https://github.com/kohsuke/github-api/pull/18`

D.4 PagedIterable dosn't use authentication

Location in archive: `github-api/19.patch`
GitHub Pull Request URL: `https://github.com/kohsuke/github-api/pull/19`

D.5 Retrieve repository directly

Location in archive: `github-api/22.patch`
GitHub Pull Request URL: `https://github.com/kohsuke/github-api/pull/22`

D.6 Implement retrieving of access token

Location in archive: `github-api/31.patch`
GitHub Pull Request URL: `https://github.com/kohsuke/github-api/pull/31`

D.7 Implement GHEventPayload.IssueComment

Location in archive: `github-api/32.patch`
GitHub Pull Request URL: `https://github.com/kohsuke/github-api/pull/32`

D.8 Add repository to Pull Request payload and wrap the PR with the repository

Location in archive: `github-api/38.patch`
GitHub Pull Request URL: `https://github.com/kohsuke/github-api/pull/38`