Real-time Conflict Awareness for Distributed Version Control Systems

Project Plan

Project type:	Master thesis
Project period:	31.10.2014 — 30.04.2015
Student:	Fabian Gremper
Status:	5th master semester
Email:	fgremper@student.ethz.ch
Supervisor:	Martin Nordio, H.-Christian Estler, Prof. Bertrand Meyer

1 Project description

1.1 Overview

In this day and age, most computer programs are written by a team of more than one person. In order to handle concurrent editing of source code, we require smart mechanisms and systems to deal with the changes of individual users.

Conventional version control provides a means to collaborate on writing programs, even different subtasks, and merge the changes later. Merging changes can however produce conflicts. To avoid big conflicts, we want to implement a real-time awareness system to inform programmers at the time of writing if another programmer is currently editing parts of the code that may be conflicting with what they are about to do.

For a previous master thesis [2], a proprietary version control system was used to alert the user of possible conflicts within the CloudStudio IDE. This thesis focuses on creating real-time awareness functionality for existing Git repositories, which significantly decreases the entry barrier for new users and allows existing Git projects to make use of this functionality.

Also, because programmers like to use their own personal development environment, we want to provide a service API that may be e.g. used in combination with plugins for IDEs or other development tools.

1.2 Scope of the work

This projects focuses on creating a useful mechanism for users of a distributed version control system to detect possible conflicts early on.

In order to achieve this, several components will need to be implemented.

First, a local plugin that runs on each developer's machine will gather information about the current status of the local file and repository status and use an API to communicate with a new, central CloudStudio server.

Second, the CloudStudio server will then use the data from all the users running the plugin, as well as the data from a centralised repository (e.g. on GitHub) to detect possible merge conflicts that

may occur later when two or more parties attempt to push their changes. The CloudStudio server will then provide an API that allows other programs or tools to retrieve this awareness information.

Third, a basic tool to access, display and inspect the output of the CloudStudio API. This part can later be replaced with e.g. an IDE plugin that directly integrates the information into a desired IDE, but this is outside of the scope of this project.

1.3 Intended results

- Provide tools that can indicate possible merge conflicts early on.
- Come up with mechanics that ensure useful and reliable awareness information.
- Make it easier for users to work on a project using a distributed version control system document.
- Documentation of work and design choices in the project report.

2 Background material

2.1 Reading list

[1] Git documentation — http://git-scm.com/documentation
[2] Automatic Version Control System for Distributed Software Development. Sandra Weber, ETH Zürich, 2012
[3] Extending CloudStudio with a collaborative remote debugger. Rand Nezha and Mert Tufekci, ETH Zürich, 2012
[4] Collaborative Debugging. H.-Christian Estler, Martin Nordio, Carlo A. Furia, Bertrand Meyer, In 8th International Conference on Global Software Engineering (ICGSE), IEEE, 2013
[5] Unifying Configuration Management with Awareness Systems and Merge Conflict Detection. H.-Christian Estler, Martin Nordio, Carlo A. Furia, Bertrand Meyer, In 22nd Australasian Software Engineering Conference (ASWEC), IEEE, 2013

3 Project management

3.1 Objectives and priorities

- Implement a real-time awareness system for users working together on a project using a Git repository.
- Document work and results in the report.

3.2 Criteria for success

The minimal quality requirements are the following:

- Implement a local plugin that communicates with the CloudStudio server via an incoming API and notifies it of local changes that are relevant to possible conflicts.

- Implement functionality for the CloudStudio server to analyse this incoming data together with the central repository data and find possible conflicts. CloudStudio will then provide an outgoing API to provide this information back to the users.
- Possible conflicts are detected on a repository and file basis.
- Works with simple commit structures that can only contain different versions of master branches for each user.
- Test coverage of the code is at least 50 percent.

Expected requirements:

- Possible conflicts are detected on a repository, file and line basis.
- Works with basic branching scenarios and structures.
- Implement a (web-based) system that accesses the outgoing CloudStudio API and displays the results provided by CloudStudio.
- Test coverage of the code is at least 70 percent.

Requirements for a result that significantly exceeds expectations:

- Explore possible problems and limitations with more complex branch structures on the repository.
- Explore and implement different algorithms to test for conflicts and whether the merge will be successful.
- Test coverage of the code is at least 80 percent.

3.3 Method of work

See project steps below.

3.4 Documentation and validation steps

- The master thesis is documented in the report.
- The current status will be discussed in a (bi-)weekly meeting.
- Test cases will be written to ensure the quality of the code.
- Theoretical scenarios are run continuously during development to test the functionality.

4 Plan with milestones

4.1 Project steps

Milestones

1) Study literature of Git and version control system functionality.
2) Design of the system and API.
3) Implement local plugin.
4) Implement incoming CloudStudio API to receive data and store it in a database or filesystem.
5) Implement CloudStudio functionality to detect possible repository and file conflicts.
6) Implement CloudStudio functionality to detect possible line conflicts.
7) Implement outgoing CloudStudio API.
8) Implement system to display the output of the outgoing CloudServer API.
9) Expand functionality for more complex scenarios.

10) Write test cases.
11) Work on tasks described in the “exceeds requirements” section if time permits.
12) Write the master thesis report.

4.2 Deadline

The deadline is April 30, 2015.

4.3 Tentative schedule