Git Integration for Legacy Software

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Harrison Li

Spring, 2022

Technical Project Team Members

Harrison Li

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Harrison Li

Technical advisor:	Daniel Graham PhD, Department of Computer Science
Technical Writing Advisor:	Rosanne Vrugtman PhD, Department of Computer Science

Git Integration for Legacy Software

CS4991 Capstone Report, Fall 2021

Harrison Li
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
HL5NE@virginia.edu

Abstract:
As a software engineering intern at a private defense contractor, I was assigned the task of improving the old file system for a warship simulator with versioning history. The features I implemented allow the customer to retrieve and restore any past version of any configuration file. With a full-fledged version history feature, the software improves in both recoverability and user-intuition. I utilized a Java Git API to create a utility class which handles all file tracking. The feature is integrated but extensive system testing is required due to potentially mission critical software.

1 Introduction

Over Summer 2021, my goals as an intern were simple: learn from mentors and create valuable software. The opportunity to create valuable software arose when I was to redesign the old filesystem for a warship simulator so the client can view the history of their simulation files and revert to any older version of any file. Such a feature serves two purposes: it introduces system recovery and provides peace of mind to the user.

Background:

Old Filesystem:

The old file system was stateless much like the default Microsoft Windows File Explorer. The system would not remember the files that were uploaded. Accidentally deleting a file on the user side meant the file would be gone forever. This statelessness can be dangerous in software because humans make mistakes and often change their minds. I felt responsibility as a software engineer considering the consequences of losing a simulation file.

Project Motivation:

The data recording software that I was to improve was written before my time at the company. The lead developer for this project quickly put together a barebones file input/output system for the simulator that is not up to professional standards. Improving the simulator file system was feature work saved for a later time. I got assigned to design and implement improvements to the file system.

2 Related Work

A valuable resource that aided my design and coding is the JGit-Cookbook by GitHub user Centic9: Dominik Stadler [1]. The JGit-

Cookbook is a public GitHub repository that serves as a JGit tutorial for developers who are looking to integrate Git into their own applications using Java [1]. The README file provides clear examples showing readers how to clone and test drive the functionalities of the JGit-Cookbook [1].

The JGit-Cookbook README also links to common examples of JGit Porcelain and Plumbing commands. Git Porcelain is the high-level functionalities [2]. Plumbing in Git can be seen as the low-level functionalities that power the Porcelain [2]. Porcelain commands in JGit provide the developer convenience while the Plumbing commands provide customizability. The JGit-Cookbook provided me invaluable examples of JGit code snippets: enough for me to start implementation.

3 System Design Process

I spent two weeks brainstorming and designing a new file system for the simulation tool. Much of the design process was learning how to format documentation effectively for future developers to reference. All design was documented in a master document. I refined the problem statement so that it was clear what parts of the software needed to be addressed by the design.

Requirements:
Next came creating the technical requirements of the new system. Requirements must be succinct and describe a single function of the system. Example requirement: The system shall show the version history of an uploaded file.

Use Cases:
Each requirement is broken into procedural use cases. The use cases describe how the user will interact with the respective system requirement. Use cases for the above requirement: click simulation tab, select file, click view history, etc.

Flow Diagrams:
Flow diagrams serve to organize the use cases in a digestible manner. Following a standard convention, a flow diagram is created for each requirement to organize data/logic movement. Use cases are ordered and used to build the flow diagram. Flow diagram paths may split at a decision point and merge at a common use case.

Document Refinements:
After defining requirements, use cases, and creating flow diagrams. I reviewed my design portfolio with my senior mentor. The greatest piece of advice I received from my mentor is to waste no words and ~~to~~ not reinvent the wheel. I revised my design documentation to be clear and concise and made the effort to include original functionality in the design.

Design Review:
After addressing my mentor's feedback, I scheduled a design review meeting with the project lead. The review consisted of a presentation of the design documentation and an honest critique. Once given the green light, I was clear to proceed with implementation.

3.1 Git and Online Repositories

Git is a technology that optimizes project file management workflow. It is currently the standard project management version control system used by software engineering teams. Web services like GitHub and GitLab are businesses that utilize Git to store and serve project files for projects of all calibers.

Git Internals:
The fundamental data structure of Git is a directed acyclic graph [3]. Graphs are used in Git because of the complex parent-sibling relationships that arise from branching and merging. The internal objects of Git are Blobs, Trees, Commits, and Branches [3]. A blob stores the encrypted contents of a file [3]. Trees represent file directories because they point to multiple blobs [3]. Commit objects represent a snapshot of the project file

system [3]. Commit objects point to tree objects [3]. Branches represent a version of the whole filesystem and point to the most recent commit within the branch [3].

A common misconception is that Git is synonymous with GitHub and GitLab. To reiterate, Git is the underlying theory and logic behind keeping project files organized and easily maintainable. GitHub and Gitlab are web services that are powered by Git to host project files in their databases. Because Git and Git services are separate entities, Git may be integrated with personal and professional projects.

4 Results

New File System:

Using an open-source Java Git API called JGit, I introduced versioning to the simulator's file system. I worked directly from the design documentation and implemented the file system requirement by requirement. The application's front-end is written in pure HTML, CSS, and JavaScript. The application's back-end is written in Java. I created a utility class called SimGitUtility.java that handled the versioning logic involving Git. I also created API endpoints for the software to interface with the SimGitUtility functionalities.

If a user were to upload an untracked file, SimGitUtility would check if the local repository contains the file and see no such file. SimGitUtility would then perform a Git Add and Git Commit to make note of the newly uploaded file.

By the end of the summer internship, the system user can add and retrieve files, view the entire history of every added file, restore past versions of a file, and compare differences between two versions of the same file.

I encountered a number of barriers while developing the JGit Integration feature. First, the codebase I worked on had tech debt. Tech debt is accrued when a developer does not thoroughly implement a feature in the interest of time/energy savings. I spent valuable time debugging errors that previously existed in the system.

Second, answers to JGit questions online were limited, so learning how to use JGit took trial and error. Working with JGit was difficult due to it being a black box plugin. The source code was not accessible from my IDE, so I looked online for the source code.

During implementation, the JGit GitHub repository that hosted the source code (not to be confused with JGit-Cookbook) was removed from the site along with its useful README file. Fortunately, a backup JGit repository is hosted on the Eclipse webserver, and I was able to pull the files from that server.

5 Conclusions

Revamping the simulator's file system with JGit was a rewarding experience. I worked closely with experienced developers to design and create a useful feature for a software in use. If the feature is fully tested and integrated into the end product, the client would have peace of mind using the simulation functionality because of the addition of file recovery.

6 Future Work

I did not get the time to thoroughly test the new file system with coverage criteria. Should the feature be used in future versions of the application, extensive system and regression testing must be done. This is largely due to the mission critical nature of the software the file system is written for.

7 UVA Program Evaluation

Of the course offerings from the UVA CS Department, Advanced Software Development

(CS 3240) and Human Computer Interaction (CS 3205) stand out as being applicable on the job.

Advanced Software Development UVA CS3240:
In my experience, Advanced Software Development at the University of Virginia is the most beneficial class for workforce development. The course teaches project management, design, Agile coding in a team, DevOps, Full stack development, frameworks, REST, and testing. Students are put into teams of four for a semester-long project, in which they play a role. The roles include Scrum Master, DevOps Manager, Testing Manager, and Requirements Manager. Everything I learned in CS3240 has been directly applied at my internship.

Human Computer Interaction UVA CS3205:
Human Computer Interaction teaches the psychology aspect of software design. The focus of the course is the design process for an elegant application. CS3205 takes students through a semester-long project emphasizing the user experience process. Students work in groups and create an application for a client. Artifact gathering, requirements drawing, requirement analysis, and client interviews all are important skills that come from CS3205. The course instills user-centric design.

Suggested Improvements to UVA CS:
It would be beneficial for UVA CS undergraduates to have predefined paths within the CS curriculum for desired specializations. If a student knows they want to pursue a career in software engineering, then a set of CS electives that are more geared towards development should be available.

References:

[1] Dominik Stadler. 2021. jgit-cookbook. In *Proceedings of Dominik Stadler* https://github.com/centic9/jgit-cookbook

[2] Scott Chacon, Ben Straub. 2014. Pro Git: Everything you need to know about Git. In *Proceedings of Apress. 1 page.* https://git-scm.com/book/en/v2/Git-Internals-Plumbing-and-Porcelain

[3] Omer Rosenbaum. 2020. A Visual Guide to Git Internals: Objects, Branches, and How to Create a Repo From Scratch. In *Proceedings of FreeCodeCamp.* https://www.freecodecamp.org/news/git-internals-objects-branches-create-repo/