

Team Sokka

Integrated Development Tools Project

Alex Udodik, Jacob Huber, Malachi Potts, Shawn Poole

Mentor: Brandon Busby

Instructor: Bolong Zeng

CptS 423 - Software Design Project II

April 27, 2021

Table of Contents	Page #
Executive Summary	3
Introduction & Background	4
Final Project Design	5, 6
Project Management	7, 8
Results	9 - 12
Broader Impacts	13
Limitations or Recommendations for Future Work	13
Summary / Conclusions	15
Acknowledgements	15
References	15
Appendices	16

Executive Summary

In this report, the project is introduced based on the initial requirements described in the project proposal poster versus the final requirements derived over the course of the two semesters and the progress made on those requirements.

The functionality for the visual studio code extension is described by its ability to automatically stage changes to be committed, and the functionality for the codebase view graphical user interface is described by its use of git metadata parsed from the output of git log and stored in either a local or online database.

The contributions made to the project by each team member are listed. The differences between the initial and final design for the project is described specifically in the results section. The unit testing results for the project and the test validation results are analyzed in detail in the testing section below that. The broader impacts of the project are discussed based primarily on the implications of storing git metadata and the types of useful information that can be derived from what the application captures. The limitations and recommendations for future work are enumerated specifically by problems the team was unable to solve in the time allotted for the project.

The report is concluded by discussing the way the goals of the project changed over time and the potential it has for future development.

Introduction and Background

Integrated Development Tools is a suite of tools that allows software developers to increase their productivity. The original project requirements that were introduced:

- Design and develop a Visual Studio plugin to automatically update version control on changes to project-mapped files
- Design and develop a notification system which notifies relevant parties of pull request updates, code check in, automated test failure, and work item updates
- Design and develop a tool for searching through a codebase which integrates with version control, showing changes to files, who checked in the change, at what time, etc.
- Design and instrument an API to collect health and usage data from the software that was created. Display the health and usage data of the software in a dashboard for easy monitoring.

Throughout the development of the project, the project changed significantly. The notification system was not fully implemented into the visual studio extension, it now displays messages based on what changes were made to a file (creating a file, deleting a file, editing files, etc.), it also launches the codebase search tool as one application.

Another thing that was added to the project was using a database to store all of the commits information (author, commit hash, author email, commit message, and the changes that were made to the code).

The codebase view became a huge part of the whole project, instead of just showing the changes to the code, check in, and what time, the code base view executes queries which can now filter commits based on date/time ranges, author names, and commit hash. New features were also added to the codebase view throughout the semester.

The codebase view was given new features for users such as: adding new repositories to the view, switching between repositories and branches, switching from where you want the data to come from (data is being store on Postgres and AWS), and giving developers grades based on how much they commit. Since the codebase view took most of the time of development, creating and implementing the health and usage data API ended up not happening.

Final Project Design

The final project design includes a visual studio code extension written in Typescript, that automatically tracks changes made to be committed. This extension can also launch the Codebase View application with a button it adds to visual studio code.

The Codebase View application allows a user to view a repository, select from the branches of that repository and view all of the commits for that branch in that repository. They can also filter those commits by the specific commit hash, the authors of that repo, a date range, a specific directory or file.

The application parses git metadata from the raw string output of the “git log --all” command. It then inserts this parsed data into either the local or online database. The application queries the data directly from that database. The application also has a button that can open a separate view intended to be a view that allows for a more sophisticated AI to grade the contributions of the user.

- **Visual Studio Extension:**
 - Tracks changes made to files (edits, new created files, deleted files)
 - A button which runs the codebase view along with extension
- **Codebase View:**
 - Displays commits from different users
 - Adds new repositories to the view
 - Can switch to different repositories and branches
 - Filters commits from different date ranges, commit hashes, authors
 - Shows commit information, (message, author, date created, ID)
 - Shows changes that were made to the code what code was added, or removed
 - Switches between seeing commits on the postgres database or the AWS database
 - Can view grades for developers that give them a grade base the number of commits they have
- **Databases:**
 - PostgreSQL database which stores all the data
 - Schema that contains information on: Commits, files, branches, repositories, commits mapped to different repositories and branches
 - AWS database which also can store all the data

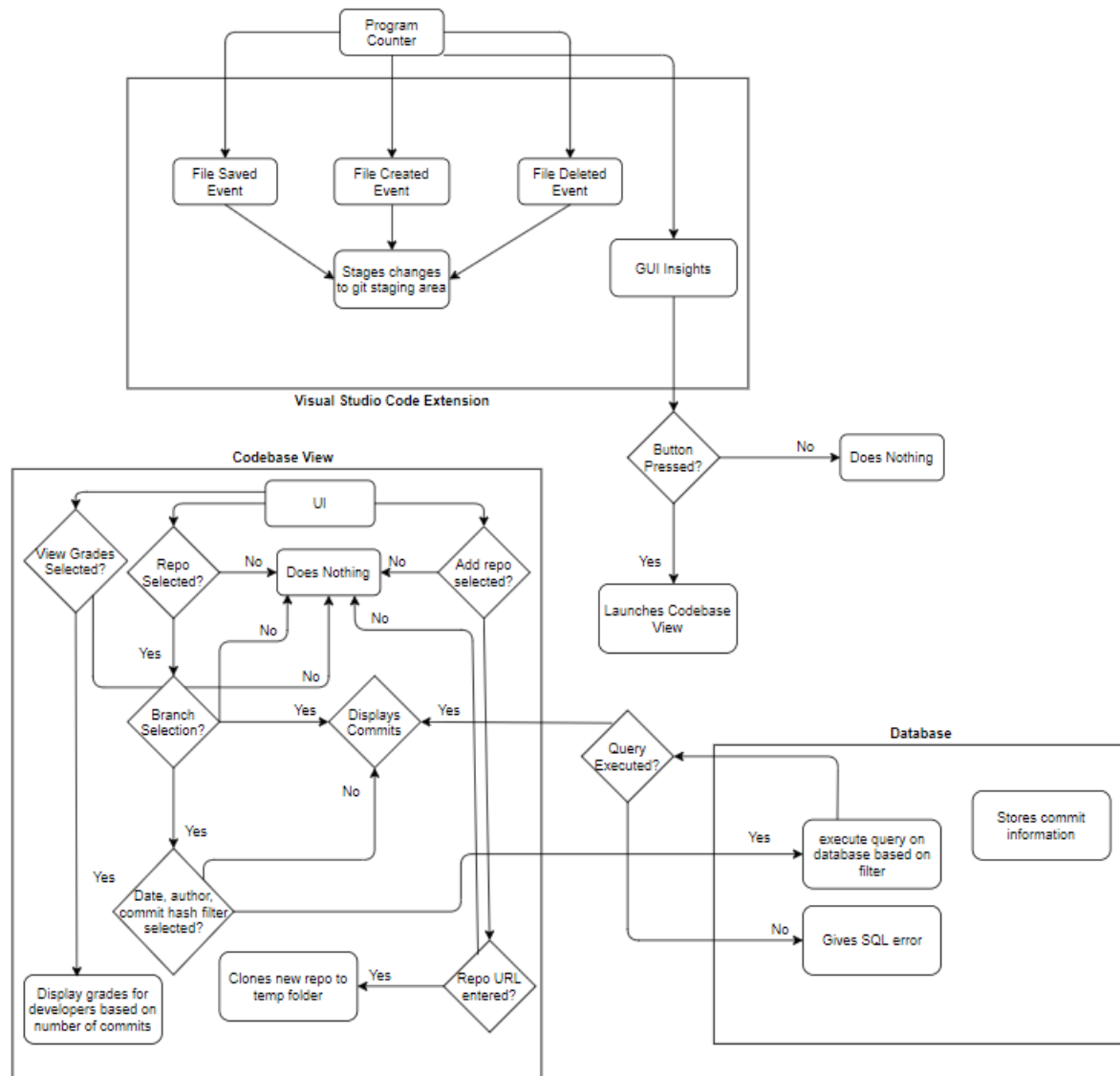


Figure 1.0: A flowchart showing how the visual studio code extension works with the codebase view, and how the codebase view works with the database.

Project Management

Team Member Contributions:

- **Alex Udodik:**
 - Contributed to building up the database schema.
 - Small additions to the SQL select query builder
 - Finalized the CodeBase View UI design.
 - Developed the CodeBase View logic for:
 - UI Navigation
 - Cloning a repository
 - Writing/reading registry keys for cloned repositories
 - Switching between local and remote database
 - Selecting files and directories when filtering by file and/or directory
 - Updating repos in the TEMP folder
 - Displaying the git diff of a commit in the code/changes richtextbox
 - Right-click menu for copying a commit hash from the list of commits
 - Displaying commit hash, date, and message in the datagridview
 - Implemented the query building for all filtering options
 - Diagnosed and solved a connection issue with the AWS database
 - Retrieved responses from Git's REST API, but was abandoned as time was short
 - Wrote the first two monthly reports
 - Contributed to the writing reports during the semester
- **Jacob Huber:**
 - Contributed to designing database schema.
 - Parsing the git log output.
 - Running git commands in C#.
 - Running git commands in TypeScript.
 - Added a button and command to the visual studio code extension to launch the GUI application.
 - Debugging database querying.
 - Investigated github and gitlab Rest APIs.
 - Made the SelectQuery and InsertQuery builders.
 - Wrote the code to insert the parsed git data into the database.
 - Team Liaison
 - Contributed to writing reports during the semester.
- **Malachi Potts:**
 - Contributed to designing the Codebase View UI
 - Contributed to designing database schema.
 - Testing the project
 - Created various unit tests
 - Ran through user scenarios to complete acceptance testing
 - Created WiX installer for the Codebase View
 - Contributed to the writing assignments during the semester
- **Shawn Poole:**

- Codebase View:
 - Created add repository form, which allows users to enter in git URLs into text box for cloning
 - Added repositories from the cloned git URL to the codebase view
 - Added all branches from the repository to the codebase view
 - Added separate view for viewing grades of each developer
 - Added filters for branches, and commit hashes
 - Contributed to final design of Codebase view
 - Added update queries for updating the database with more repositories and branches
- Database Schema:
 - Added data table for repository which keeps track of repository data
 - Added data table for branches which keeps track of branch data
 - Added data table for mapping different commits to branches
- Contributed to all writing assignments, adding more tasks to the gantt chart, as well as final presentation, and creating most of the final poster.

Results

Final Prototype Description

The integrated development tools project is a suite of tools that will allow software developers to increase their productivity. These tools consist of a Visual Studio Code extension that can automatically handle version control while notifying users changes that were made to any files, and a code search tool that handles user commits from selected repositories and branches. The code search tool shows relevant data from the repository such as: who made the change, the date the change happened, what time, showing changes that were made to the code.

Comparison of the final product vs original product requirements

- **Original Product Requirements:**
 - Develop visual studio plugin that automatically updates version control on changes to project mapped files
 - Design and develop a notification system which notifies relevant parties of pull request updates, code check in, automated test failure, and work item updates
 - Design and develop a tool for searching through a codebase which integrates with version control, showing changes to files, who checked in the changes, at what time, etc.
 - Design and instrument an API to collect health and usage data from the software that is created. Display the health and usage data of your software in a dashboard for easy monitoring
- **Final Product Requirements:**
 - Develop visual studio extension that works together with codebase search tool, which tracks changes made to files (edited, created, deleted)
 - Codebase view that shows commits from different authors, dates, viewing commits from different repositories and branches, and changes that were made to the code
 - Giving developers a grade based on the number of commits they have
 - Moving data from PostgreSQL onto Amazon Web Services

Comparing the original product requirements and final product requirements, the visual studio plugin executes git executables which brings over the changed files to the git staging area, we did not finish the notification system, instead anytime a file does get changed (edits, created, deleted) it shows a message box that says what the change was. The codebase search tool was finished, it shows all the commits as well as changes that were made to the code. Comparing the final product to the original product, the final codebase view design has more functionality than the original design. For the final product, we are able to filter commits from different date ranges, author names, commit hashes, being able to add new repositories to the codebase view, switching from different repositories and branches, and having a view that shows the number of commits each developer has and giving them a grade based on the amount of commits they have. When the original product design only needed to show changes to files, and who did what change. Adding all the functionality to the codebase view took up most of the time which did not allow us to finish the health and usage data API for the project.

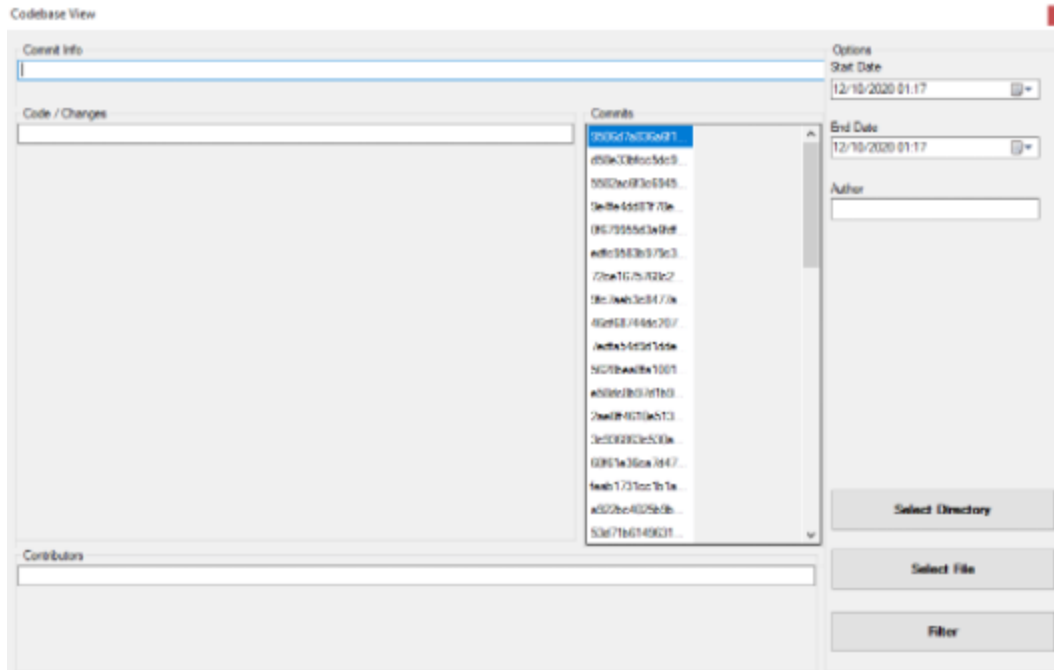


Figure 2.0: The codebase view design from the alpha prototype, only had features for showing commit hashes, messages, author names, buttons and filters had no functionality.

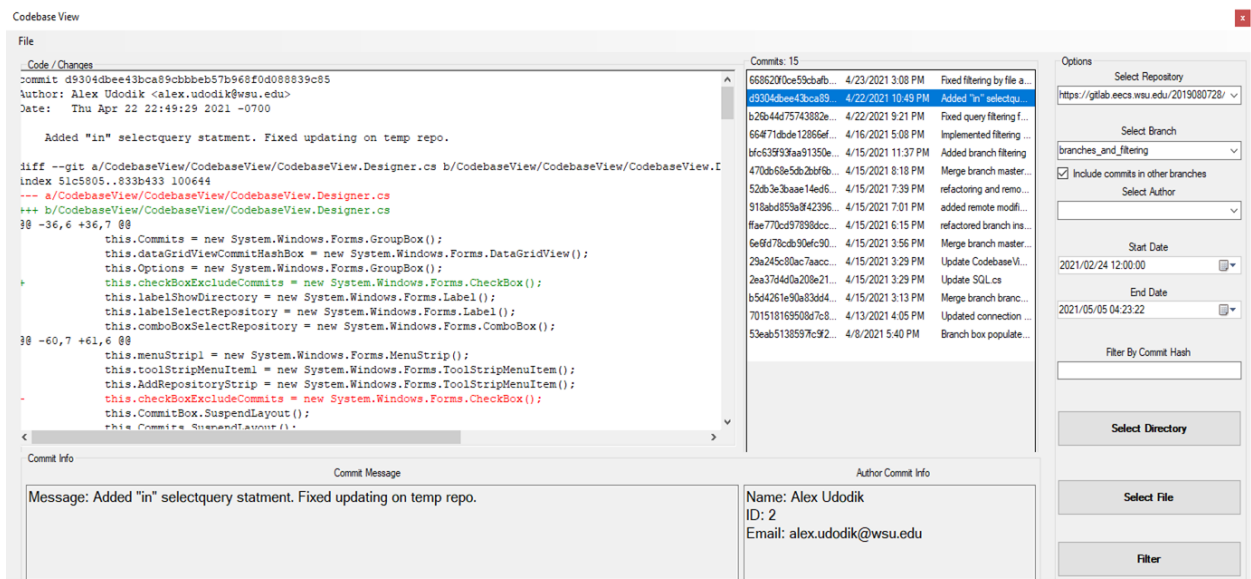
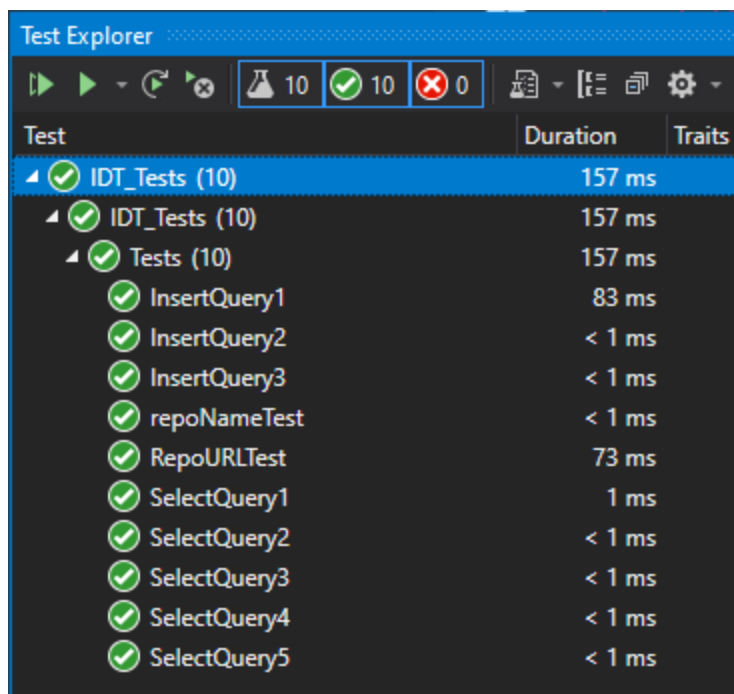


Figure 3.0: The final design of the codebase view. Has all features that were not completed in the alpha prototype, all filtering options work, new features were added such as: selecting a repository, branch, filtering by commit hash, adding repository, switching databases, and viewing developer grades.

Final Prototype Test Results

The overall testing regimen for this project was completed by creating various unit tests that test internal functionality of the Codebase View's logic. After all of the unit tests are completed and have passed, the more in-depth acceptance testing through user scenarios can be completed. The various tests help to give a clear picture of which components are functioning properly.



Test	Duration	Traits
✓ IDT_Tests (10)	157 ms	
✓ IDT_Tests (10)	157 ms	
✓ Tests (10)	157 ms	
✓ InsertQuery1	83 ms	
✓ InsertQuery2	< 1 ms	
✓ InsertQuery3	< 1 ms	
✓ repoNameTest	< 1 ms	
✓ RepoURLTest	73 ms	
✓ SelectQuery1	1 ms	
✓ SelectQuery2	< 1 ms	
✓ SelectQuery3	< 1 ms	
✓ SelectQuery4	< 1 ms	
✓ SelectQuery5	< 1 ms	

Figure 4.0: Test explorer output after running unit tests.

- **Codebase View GUI Components:** Each of the GUI components function properly (code/changes window, commit message box, author commit info, selectable commit window, each of the filter options, select directory button, select file button, and the filter button.)
- **SQL Filtering:** All queries are executing the correct filters completely. The codebase view displays the results from the queries. The user can now filter and view commits from different authors, date ranges, commit hashes, branches, and repositories.
- **Repository & Branch Switching:** The codebase view can now switch between different repositories and branches. A user can now also add their own repository if they want to. Users can view commits that are only tied to the selected branch, not the whole repositories' commits.
- **VS Code Extension:** The visual studio code extension shows all changes that were made to files, when they are edited, created, and deleted. Also changed the executable into a button that launches the codebase view with the extension as one application.
- **Postgres Database & AWS Database:** The codebase view can now query data from both the postgres database and the AWS database. The user can switch between which database they want to use on the codebase view.

Final Prototype Validation Results

To complete testing validation, various user scenarios/ acceptance tests were run. The acceptance tests we ran were designed to replicate what a potential customer using our tools might do. Each of these user scenarios can be completed successfully.

- **User Scenario 1: Adding new repository:**
 - Users can add a new repository through the integrated add repository button by selecting File > Add repository > then pasting in the URL of the repository they would like to add.
- **User Scenario 2: Viewing code changes:**
 - Users can select a commit from the commits window and view the git diff (code changes) of that specific commit. Starting by selecting a repository from the dropdown in the 'Select Repository' list, then selecting the branch the commit is on. After selecting the branch, the user can now click the filter button which filters the list of commits based on the current repository and branch. The commits list now only shows related commits and the user can select any of the commits to view specific code changes completed from that commit.
- **User Scenario 3: Filtering commits:**
 - Users can select which filtering options they want to do. They filter by an author by selecting the combo box labeled "select author" then the user can select which author's commits they want to view, then clicking the filter button will return the commits from the selected author. They can filter by dates by selecting a start date and an end date, then selecting the filter button to see commits in that date range.
- **User Scenario 4: Switching repositories and branches:**
 - Users can select a repository through the "select repository" combo box, then select a branch in that repository through the "select branch" combo box. After selecting both a branch and repository, and clicking the "filter" button, it will show commits within the selected repositories branch. After viewing commits from the selected branch and repository, the user can go back to the combo boxes for both repository and branch if they want to see commits from a different repository. After selecting a new repository and branch, the user will only see commits from the selected repositories branch that they switched to.

Broader Impacts

The utility of this application extends beyond the scope of this project by prioritizing the statistical insights of contributions to a repository, over the ability to make changes. Gitlab does not offer an easy way to view the number of commits a specific user has made to the repository.

The extension of this application implies the ability to grade authors based on the number, frequency, and size of their commits. Further development could allow an AI to evaluate all of these factors, and even look at commits that were added and then reverted to track mistakes that were made.

Limitations and Recommendations for Future Work

CodeBase View Application Freezing: The current limitation with the CodeBase's View code changes box is that if a commit is selected from the box that displays the list of filtered commits, and that commit contains a large git diff, the code/changes box will not update and the application will freeze and in most cases will not allow the application to run further. The cause of this issue is because the code/changes text box is of type RichTextBox and has a specific character limit which is the max size of a 32-bit signed integer.

- **Possible Solution 1:** If the amount of lines or characters from the git diff exceeds a certain threshold, display a warning message to the user and provide a link so the user can view the git diff on Github's website.
- **Possible Solution 2:** Offload the loading and parsing of the commit diff onto another thread and slowly feed in more lines into the textbox while removing the oldest viewed lines. This will allow the code/changes text box to be in its own safe character limits

Exhaustive Check and Inserting into a remote Database: The CodeBase View currently checks if a specific row exists in a database, and if it does not, the row will then be inserted into the database. This is the case for every insertion into the database which is sufficient for repositories that have a "normal" size and do not contain a large file/branch/commit count. The issue is when a database is hosted remotely and latency becomes a factor. Inserting a new repository into a local database on a fast computer takes a moment of around a few seconds, but inserting into a remote database such as one on AWS, takes a moment of at least 20 minutes. This is not ideal for a user that wants to view a repository of their choice.

- **Possible Solution 1:** Refactor the parsing and inserting of data to build single strings that contain multiple insert statements instead of individual strings with only one insert statement.
- **Possible Solution 2:** Utilize database caching. Using a local database can help improve database access times significantly because the remote database is not used until a database synchronization needs to be initiated which can be started from the selection of the user and/or when the user computer is idle which can be configured as a setting in a future window of the application.

Performing Updates on Cloned repositories in the TEMP directory: The current solution to perform database updates on cloned repositories is to check if each folder in the TEMP folder for the CodeBase View has a '.git' folder inside the repository. The issue with this is that the user can perform some actions within these folders such as deleting certain files and cloning another repository within another repository accidentally.

- **Possible Solution 1:** When performing an update, delete the repository completely and re-clone it back to the TEMP folder. A slight issue with this solution is that some users may be negatively affected by this if their internet connection has a bandwidth restriction on how much they can download during a certain period.
- **Possible Solution 2:** Check for git submodules and/or subtrees. There may be repositories that contain other repositories on purpose and it may be what the user wants.
- **Possible Solution 3:** Run a 'git rev-parse --is-inside-work-tree' git command within that repository which will return either true or false indicating if it is a git repository or not.

Reverting Commits: The state of the project is that the CodeBase view application only supports additions from repositories. There is no handling when a branch or commit is deleted.

- **Possible Solution:** Modify the schema to support "alive" and "deleted" branches. Only return branches that are not deleted for that current repository.

Git Log Parsing: If a user has a non-default git configuration file installed to modify their git shell window, the current git parsing code will break and will not be able to parse any git command outputs.

- **Possible Solution 1:** Notify the user to remove the config file before proceeding with git parsing.
- **Possible Solution 2:** Instead of using git log to parse commits, using Git's REST API will be a more cleaner parsing alternative and will allow the user to keep their git config file in place. The CodeBase view will be more of a wrapper to execute queries, retrieve data, and display the data to the user.

The Parsing of Repository Data: The parsing of all repository data is done by executing git commands through git bash in the background. This leads to exhaustive parsing and storing of repository data on the user's computer.

- **Possible Solution:** Switching to Git's REST API to retrieve all information about a repository including the git diff that displays the changes within a commit. Not only will this lead to a cleaner approach, but it will also make it easier to develop the application for other operating systems other than Windows.

Summary / Conclusions

The initial requirements were more broad and ill-defined, as we progressed we defined more specific requirements and focused the direction of the project. Instead of 4 separate tools it narrowed to two, and we focused on flushing out the functionality of the extension and the code search tool. This resulted in integrating the two tools together. The database used for the application was also migrated to AWS Aurora. This project has been a good proof of concept for retrieving git metadata and deriving useful views and insights with it.

Acknowledgements

Much appreciation to our mentor, Brandon Busby and instructor, Bolong Zeng for giving us the opportunity to develop and work on this project. Another Thank You to Brandon Busby for providing AWS accounts to manage the database hosted on AWS along with helping to pay for the costs of running the database.

References

1. <https://docs.microsoft.com/en-us/dotnet/api/>
2. <https://aws.amazon.com/rds/aurora/>
3. <https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History>
4. <https://code.visualstudio.com/api>

Appendices

Appendix A:

The iterative design activities were that, for the beginning of our activities, we met with our client, which is our mentor to gather data on what we want the project to be, as well introducing ourselves to our mentor. We discussed what the whole problem is and that we wanted to create a better set of tools for software developers.

We discussed what we wanted the project to have in terms of its requirements, such as what we wanted for the visual studio plugin, the code search tool, and the health & usage API. For our next iterative activity, we started brainstorming similar programs that perform the same functionality. And which packages and products that we planned to use for the project. The decided packages were visual studio code, git, postgres, and C# winforms.

What was also taken into account were the stakeholders for this project. We considered that the project mainly would be used by people who use Visual Studio Code and git. After brainstorming what the project needs, we started thinking about potential stakeholder & client needs. We gathered that we wanted the application to show useful information on commits, the number of commits made by a user, the database that will store the git metadata, as well as testing the reliability of the product, making sure the information is correct across multiple systems that use the software.

After having through research on potential stakeholders & client needs, we came up with a proper description of the project. It was decided that the project will consist of a Visual Studio Code extension that will be tracking all the changes made to files (edited, created, deleted), the codebase view that will show the commit information, as well as filtering through commits, and a database that will store all of the commit information.

Once a proper project description was created, we started looking for potential solutions that will complete each component. A solution that was found for retrieving git metadata, we decided to parse through the git long information to retrieve the necessary git information (commit id, author, date created). For the visual studio code extension the solution for tracking files, we decided to have it trigger git commands based on which file event happened. That way once the file has been changed, it puts the file in the git staging area to be committed. The solution for code base view design, we decided that the GUI will display the commit information from the database. As well as the commit itself, the changes to code, authors, messages, and filtering options for the commits.

After having the final technical specifications, we started building the alpha prototype for the application itself. The visual studio extension would track changes made to files by triggering file event handlers. When a file was created, deleted, or edited, it will display a message box saying what action was performed. As for the code base view application, it was only able to display the commit message, commit id, and the author name. After finishing the alpha prototype for both the extension and codebase view, we came to the conclusion that the extension and GUI have to function as one application, and the GUI filtering features have to be completed. After finishing the alpha prototype, we plan on applying those features onto the next semester.

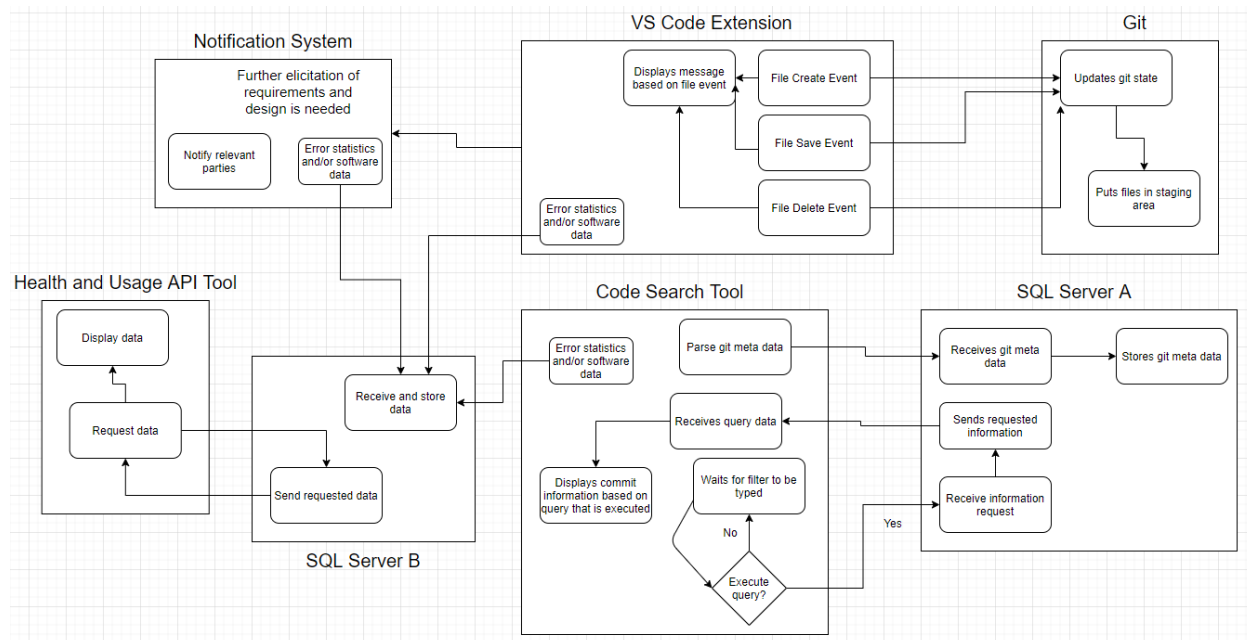


Figure 5.0: Shows a diagram of the original system. Shows how the extension communicates with the code search tool, notification system, git, the health and usage data API, and the database.

Team Photos:

Alex Udodik



Malachi Potts



Jacob Huber



Shawn Poole