CodePilot: Real Time Collaborative Programming with Asynchronous Version Control Support

by

Jeremy Benjamin Warner

Submitted in Partial Fulfillment of the

Requirements for the Degree

Master of Science

Supervised by Professor Philip J. Guo

Computer Science Department

Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester

Rochester, New York

2016

Dedicated to all who have suffered through merge conflicts.

Biographical Sketch

The author was born and raised in eastern Long Island, New York, eventually graduating from Riverhead High School. He attended the University of Rochester, and graduated with a Bachelor of Science degree in Eletrical and Computer Engineering. He pursued his research in Computer Science under the direction of Professor Philip Guo.

Acknowledgments

First and foremost, I am grateful to my father and mother, Hollis and Paula Warner, whose unwavering support enabled and encouraged me to pursue my dreams wholeheartedly. Your love and guidance showed me the importance of caring for others and working to benefit your greater community.

I thank my research adviser Philip Guo for providing consistent support and invaluable advice. Working with you has been an absolute pleasure, and helped me grow both as a researcher and as a human being.

I am also thankful to Dean Wendi Heinzelman for guiding my work in your research lab prior to my working with Professor Guo, as well as supervising my senior design project. Your concise judgment and willingness to help were critical in many of my successes.

Thank you to Professor Chen Ding, for reminding me that there is beauty to be found in the winter, where stars sharpen and glitter.

Finally, thank you to the countless others that I have met and worked with at the University of Rochester. It brings me great joy to be a part of such a vibrant and positive environment (weather notwithstanding).

Abstract

This thesis analyzes perceptions towards collaborating with version control and introduces a new system, CodePilot, for project-wide synchronous collaborative programming backed by version control. We review our study of programmers using the system to collaborate on web development tasks, and suggest that combining synchronous collaborative programming with version control (asynchronous collaboration) is a novel, powerful paradigm in programmer collaboration. It not only makes version control more accessible for novice programmers, but prompts experienced programmers to coordinate and parallelize their software development efforts.

Contributors and Funding Sources

This work was supported by a thesis committee consisting of Professor Philip Guo (advisor) and Professor Chen Ding of the Computer Science Department and Professor Marvin Doyley of the Electrical and Computer Engineering Department. All work conducted for the thesis was completed by the student independently, and funded by the student's adviser's startup grant.

Contents

List of Figures

Chapter 1

Introduction

Software development is becoming more collaborative, popularized by GitHub and the open source movement. CodePilot is a web integrated development environment (IDE) which combines synchronous and asynchronous collaboration. It does this by allowing users to import any public repository of code from GitHub into CodePilot, where it syncs the state of the project across multiple users in real-time. It does this in a fashion similar to that of Google Docs or other real-time online editors. However, CodePilot is not only limited to a single buffer of text and supports syncing changes across many files simultaneously. Collaborators can see each other's edits as they type, and when they are satisfied with their changes, they can review, commit, and push their code back to GitHub, allowing that work to be shared with users outside of CodePilot.

1.1 Software Collaboration

There are two main modes of synchronization in collaboration: real-time and non-real-time. Real-time synchronization usually applies operational transform syncing to maintain a persistent single state of a document as changes come in from multiple sources [12]. This is the technology that supports Google Docs and Spreadsheets - all edits are seen in real time. There are also non-real-time collaboration systems, such as distributed version control. In this model, multiple copies of the same codebase are allowed to exist in different states, and periodically update themselves from each other. Commonly there is still a centralized repository host (such as GitHub), and teams will use branches to split changes, or break the codebase into different relevant repositories if the complexity deems it necessary.

Coding with others cohesively is hard. This difficulty generally grows when distributed version control is used, since more people can program together using version control when compared to traditional pair programming. While version control is widely hailed as a useful and perhaps even indispensable tool, dealing with merge conflicts can impede a novice programmer's collaboration efforts from being effective. This often leads to frustration and an initial distaste for version control. The budding programmer now must not only must learn the core computer science fundamentals, but become proficient with this new and unfamiliar system.

Compare this to collaboration with Google Docs: powerful real-time syncing mechanisms allow users to work on the same document and see each others changes as they happen. This feature is heralded as revolutionizing document collaboration and often credited for allowing Google to weaken the stranglehold that Microsoft Office previously had on the document editing market. This isn't to say that real-time editors will solve all our problems, but it does suggest that other effective modes of collaboration lie in the range between the two extremes we have found ourselves with.

Asynchronous version control (such as git or mercurial) provides a concise, discrete history of the changes that a codebase has endured. Each code save-point in version control (known as a commit) is labeled with a message which is generally used to describe what semantic meaning the change has, who wrote it, when it was wrote, and what exactly was changed. It may not be the most intuitive tool to learn how to use, but it has been embraced by the professional and open source software development communities, with GitHub hosting over 35 million git and subversion repositories from 14 million users [13].

Both of these collaboration methods provide unique benefits to programmers, but there has not been major implementations of collaborative programming systems which combine these two methods. CodePilot supports both real-time and non-real-time collaborative editing, finally bringing together real-time programming and non-real-time collaboration. It allows its users to import a repository of code from GitHub and start collaborating synchronously on a project, and still periodically commit updated versions.

1.2 Scaling Human Effort

The wisdom of the Fred Brooks' classic 'The Mythical Man Month' is relevant here. The larger a software team grows, the more overhead in communication and synchronization there is. People are less able to collaborate effectively on a project due to this overhead [5]. The relationship between workers and productivity in software is complex: perhaps 2 workers could complete a project faster than a single worker, but what about 200 workers? Designing and writing software is an incredibly complex process, and combining this with the coordination required for software teams to work efficiently may actually make smaller teams perform better than larger teams. According to Brooks, there are four essential difficulties that the essence of software contains:

Invisibility software's function is hard to visualize, thus hard to cognize
Changeability even after release, software is under pressure to be modified
Conformity requirements and interfaces man-made and not standardized
Complexity abstractions eliminate redundant parts of code, each part unique

CodePilot aims to tackle some of the challenges presented from these difficulties. CodePilot increases visibility through its web renderer, allowing programmers to see the representation of their code. It handles changeability with its operational transform syncing mechanism and the GitHub-backed repository branching and version control. It encourages conformity with its HTML, CSS, and Javascript debugger and its GitHub issue integration, by allowing revisions of the software to be examined and marked with any errors that are found. Complexity is not directly handled by CodePilot, but other tools such as Python Tutor offer in-browser tools that help with both invisibility and better understanding complexity even if not reducing it [17].

1.3 Online Compilation

Web browsers are an essential part of modern computer use, and developers frequently use them to access the internet, looking up documentation or asking questions on StackOverflow [4]. Increasingly, sites have come out such as HackerRank, Interview Cake or Python Tutor which allow users to evaluate dynamic code execution in the browser. One limitation of these sites is that they usually limit evaluation targets to code snippets that cannot be very complex. There is very limited support for executing larger projects, although continuous integration testing sites such as Travis-CI allow this.

Dynamic user interfaces are not easily tested, but our feedback generation tool captures a significant amount of relevant information about the page, creates a GitHub link with the entire state of the page and provides a link for the issue to be re-rendered and examined by others in the future. It does not entirely address software complexity, since evaluating the correctness of code automatically in the browser is a problem that is not fully solved. There exist tools such as Travis CI for running tests or other linters which exist to check for code syntax correctness but CodePilot doesn't address these.

CodePilot allows novice programmers to use version control together, in a manner that promotes self-efficacy and reduces frustration. CodePilot's integrated development tools, namely the testing and issues integration, fosters task parallelization between team members.

1.4 Thesis Overview

The key findings are that a mix of real-time and non-real-time version control systems can improve novice programmers sense of self-efficacy when using version control systems, and also that they can used to help parallelize software development and reduce the amount of unnecessary merge conflicts. We will discuss relevant collaborative editors, pair programming and education research, the sync-async collaboration paradigm, the CodePilot system implementation, evaluation of the CodePilot system, results of the evaluation study, discussion of study results, and finally conclusions and future work.

Chapter 2

Related Work

A good amount of relevant work has been done on other real-time collaborative editors, as well as editors that integrate more blended forms of asynchronous and synchronous collaboration features. There are also relevant studies of how people work best together on software projects, along with studies of how effective different types of pairs of students are together. This body of work was influential on the design of CodePilot. Collaborative learning and education is an active area of research which explores how students best work together. This work effects teaching practices, and therefore students who later become developers who collectively shape industry practice.

2.1 Collaborative Editors

Increasingly, code is stored and distributed on GitHub, the popular git repository hosting website. It can act as a library of guides and example code for beginners to work through. GitHub also offers a dedicated educational tool, GitHub Classroom, allowing programmers to easily learn both version control and coding fundamentals [13]. Even so, this often focuses on non-real time collaboration or single developer workflows. Still, there is exciting work being done on how collaborative editors can allow programmers to work together more effectively both synchronously and asynchronously.

There have been numerous editor plugins that aimed to provide a syncing mechanism for programmers, such as *CollabVS* [19] and *ATCoPE* [10], for Visual Studio and Eclipse specifically. These are less tied to version control than CodePilot, and mainly provide a real time syncing mechanism. More recently, there is MadEye and Cloud9 - web editors that allow you to share your session with someone else [26, 6]. MadEye is backed by your computer's file system while Cloud9 allows you to import one of your repositories from GitHub, but not to commit back. Both of these make great improvements over traditional pair programming schemes, but are designed for simply dropping in and solving an issue with the code. This focus on larger software teams collaborating opportunistically differs from the 'co-piloting' pair programming scheme.

One notable system that provides a similar intermediate mix of real-time and non-real-time collaboration is Collabode [14], which allowed users to have concurrent editors, but filtered out the error messages a single user would see to those that they themselves had generated. Merging actual copies of divergent copies of code could be delayed until the errors generated from the changes had been resolved, efficiently combining real-time changes with a software verification gating system. Other tools such as Syde seek to better synchronize developer awareness by providing contextual awareness beyond version control changes, by notifying developers of changes in source code in real time [18].

Developers use the internet as a resource to learn and remind themselves of coding concepts or syntax. Some developers even reported visiting the same site many times, acting as a sort of external memory bank. As the code stored on the internet grows, example-centric programming will become more popular, as developers fork relevant code and modify it to their ends [4, 3]. A web IDE such as CodePilot allows and encourages this information to be cached and accessed with less of a major context switch, as the user remains in the browser.

2.2 Pair Coding Dynamics

In a large quantitative study, code produced from pair programming contained fewer bugs, leading to overall higher quality code [33]. With this in mind, CodePilot sought to retain these benefits while allowing work to speed up by enabling two drivers (rather than having one person only monitor the driver's actions).

There is also something to be said about the social dynamics of working together, specifically between pairs in pair programming [32]. If there is a mutual perception of technical capabilities, then pairs worked better together. This is opposed to pairs where skilled programmers work with novice programmers. Furthermore, students with a good amount of confidence enjoy the pair programming experience most and students did their best work when working with similarly skilled programmers [24]. The skill metric was self reported, and five self-described warriors still got less under a 50% in the class. Overall, students liked pair programming and believe that it helped them achieve effective solutions.

Side by side programming is another paradigm where people are at adjacent workstations, and both work on coding, differs from co-piloting code. (which would be pair programming but the other workstation focuses on testing, version control, and reference hunting). They cite a study comparing distributed and co-located pair programming: distance does not matter. Also relevant is concurrent programming and browsing - one person codes, the other person reviews the code or requirements documents [7]. Another programming paradigm is a rapid development cycle in which two developers work together in test-driven development, where one of them will work primarily on the implementation and one of them will work primarily on the tests. This idea was tested with the Collabode system, and the developers reported that this split was very natural [15]. CodePilot's pilot/copilot methodology was originally inspired by this work.

Systems such as FASTDash [1] pioneered a dashboard for keeping developers up to date on what part of a project they are working on as early as 2007, before Git or other distributed version control systems had started to balloon in popularity. In a similar vien, PatchWorks provides an interesting methodology on methods for allowing easier navigation of code [20]. While CodePilot simply uses a standard single pane text editor, working with more advanced views of code could be integrated into CodePilot. This could be a shared view that shows which portion of code is currently being viewed by your collaborator.

There is also the option of having many people contribute to the codebase, rather than a few trusted collaborators. Systems such as CrowdCode [25] exhibit the feasibility of this approach. Fleshing out psuedocode, creating unit tests, and writing additional documentation for a codebase can be effectively distributed many workers, given the appropriate back-end coordination system.

2.3 Collaboration in Education

Computing education is growing in popularity, as a new CS for all initiative was launched recently. However, coding is not done in pure isolation, and nearly all production software involves a great deal of collaboration between developers. While there has been much effort in analyzing how content or presentation may be augmented, it is important to also focus on how the methods of collaboration can be improved. When used in educational setting, pair programming serves as effective collaborative learning instance [29].

Work has been done that suggests that activity streams with individual programming assignments can positively influence learning outcomes. People are more likely to use a 'stream' (forum) if integrated into their programming experience, which they implemented with Visual Studio [22]. Social programming environments are have been introduced, focusing on the programmer's sense of community and ability to communicate with others, which are very important in educational environments [21]. This served as a guide when implementing the activity feed.

Pair programming can also be a effective to spread expertise among programmers, which is useful information for both industrial and computing education professionals. This adds to the value that programming together has, adding more weight to the argument that collaborating with others not only produces more ideas and potential solutions but also can be a form of training and educating programmers [23]. When pair programming, students tend to get stuck less and explore ideas more. Contrastingly, students reported that when programming alone they were more confident and understood their programs better after they were done [31]. CodePilot seeks to strike a middle ground between these two workflow patterns.

Implementing a curriculum where students can collaborate on projects early on can lead to much better retention rates, both in the class and the major [34]. Additionally, web based IDE's such as JavaGrinder strip away the tedium of setting up development environments and allow programmers to worry more about the execution of their code [11]. Using the JavaGrinder was as web IDE while pair programming has been found to positively effect student retention in introductory computer science classes. CodePilot can help support this by fostering more collaboration.

Chapter 3

CodePilot: Git Realtime Collaboration

3.1 Motivation

We hypothesize that combining both real-time and non-real-time collaborative tools will lead to a richer and more cohesive collaborative programming experience, as opposed to either of those methods alone. Synchronous editing, such as that supported by Google Docs, is excellent at keeping collaborators in tune with what each other is working and focusing on. Seeing edits performed in real time gives a sense of dynamism to the work being done, and allows potential misunderstandings to be cleared easily by presenting the same shared context. The model of collaboration for CodePilot is shown in Figure 3.1, where the shared editor's contents are being synced in real-time across multiple users.

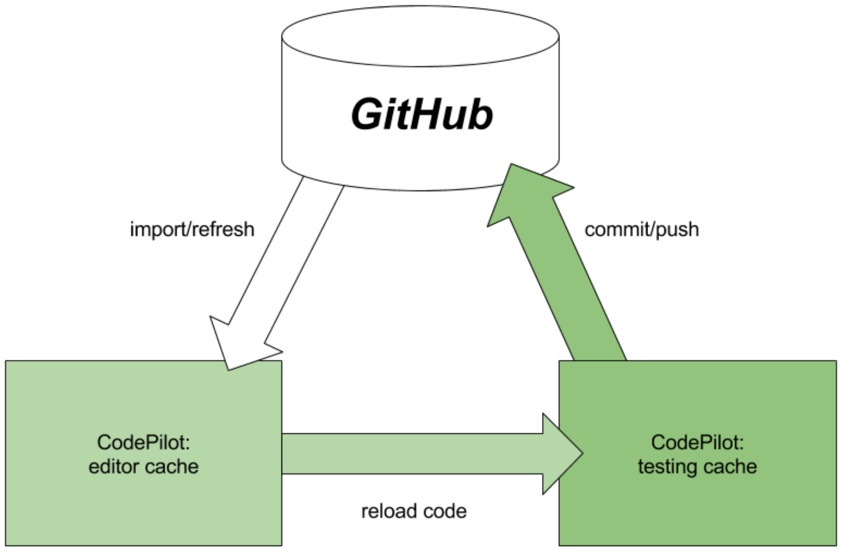


Figure 3.1: CodePilot File Flow Pattern

Issues can arise if communications between programmers editing the same codebase suffer significant delay. A common trope of multi-developer version control scenario is the commonality of the merge conflict. This often happens when a certain part of a file has been changed by two different developers, inadvertently stepping on each other's toes. Before the second developer pulls from the main repository host (GitHub, Bitbucket, etc.), another developer pushes a commit. When the second developer makes another commit locally (based on the last seen commit from GitHub) and tries to push it to GitHub, they might see an error like:

```
To https://github.com/USERNAME/REPOSITORY.git
 ! [rejected]        master -> master (non-fast-forward)
 error: failed to push some refs to 'https://github.com/USERNAME/REPOSITORY.git'
 To prevent you from losing history, non-fast-forward updates were rejected
 Merge the remote changes (e.g. 'git pull') before pushing again.  See the
 'Note about fast-forwards' section of 'git push --help' for details.
```

Then as suggested, they may try to pull and merge only to see a variant of this error message:

```
 CONFLICT (content): Merge conflict in <fileName>
 Automatic merge failed; fix conflicts and then commit the result.
```

This frustrates programmers since they simply want to share their work with their teammates, but now they must coordinate with the first committer and resolve the merge conflict by manually by creating a new version of the code from the two branches which incorporates both changes, or rebasing the head of their local branch off of the newly updated remote and then merging in the changes again. The benefit of synchronous collaboration systems is that there are *no merge conflicts*. The syncing mechanism (such as Operational Transform) automatically merges the changes coming in from each programmer and keeps a single persistent document for all collaborators. A diagram of the differences in merging strategies is shown in Figure 3.2.

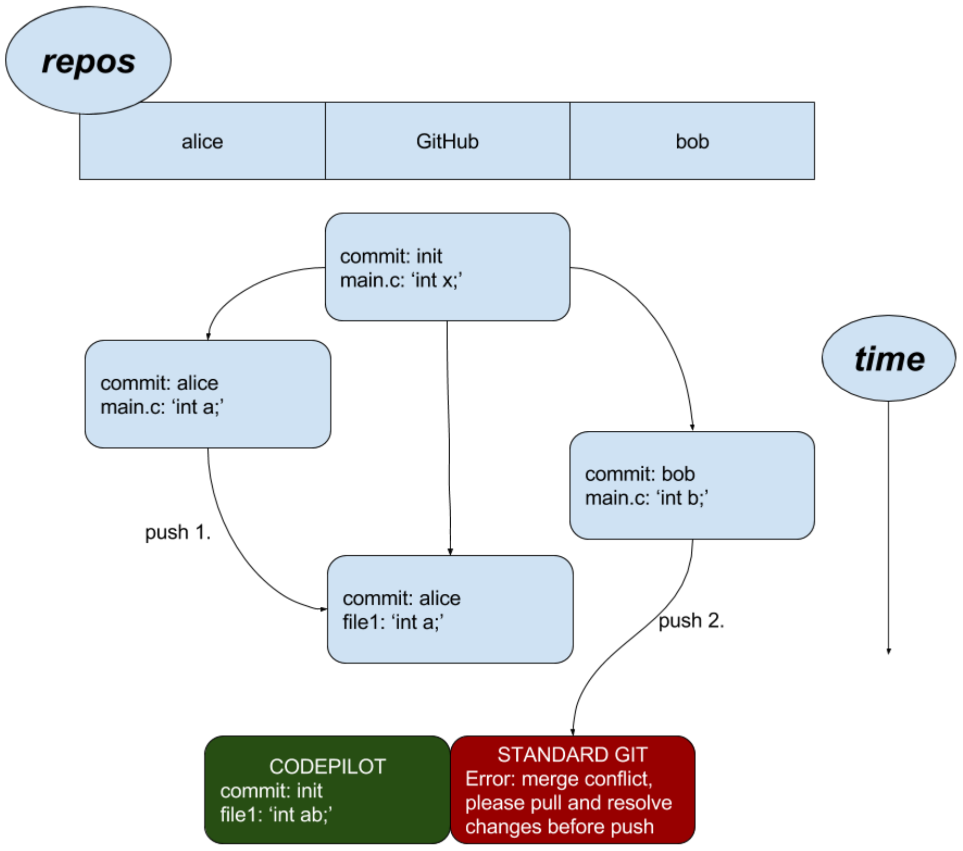


Figure 3.2: Difference Between CodePilot and Standard DVCS Merge

This is not to say that the synchronous collaborative editor is the perfect programming tool. Conversely, asynchronous collaboration through version control provides its users with the ability to update their versions of the codebase as they please, rather than keeping a single persistent shared state. The interactions between GitHub and CodePilot's servers is shown in Figure 3.3. Collaboration with version control through platforms such as GitHub and Bitbucket has been steadily growing more popular, and will likely continue to be a way that future developers collaborate together on software projects. There is often a significant overhead when learning to use git outside of a single developer context for budding coders, though. They are already building their fundamental computer science skills, on top of learning how to use git or mercurial. Merging divergent repository copies distracts them from the 'meat' of the programming, although learning how to use version control systems effectively is an increasingly important skill, corresponding with their usage's rising popularity.

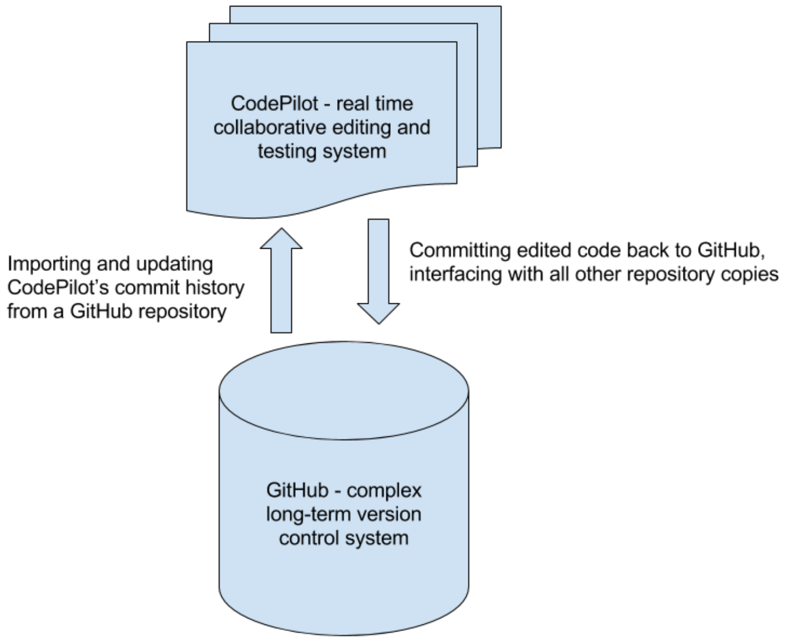

CodePilot - real time collaborative editing and testing system
Importing and updating CodePilot's commit history from a GitHub repository
Committing edited code back to GitHub, interfacing with all other repository copies
GitHub - complex long-term version control system

Figure 3.3: CodePilot and GitHub Server Interactions

CodePilot aims to take the best of real-time and non-real-time systems and combine them in a system that is more intuitive for novice programmers. CodePilot also strives to facilitate collaborators' awareness of each other's activity without hindering or distracting any collaborators that use it. This mix between two prevalent paradigms is demonstrated in Figure 3.4.

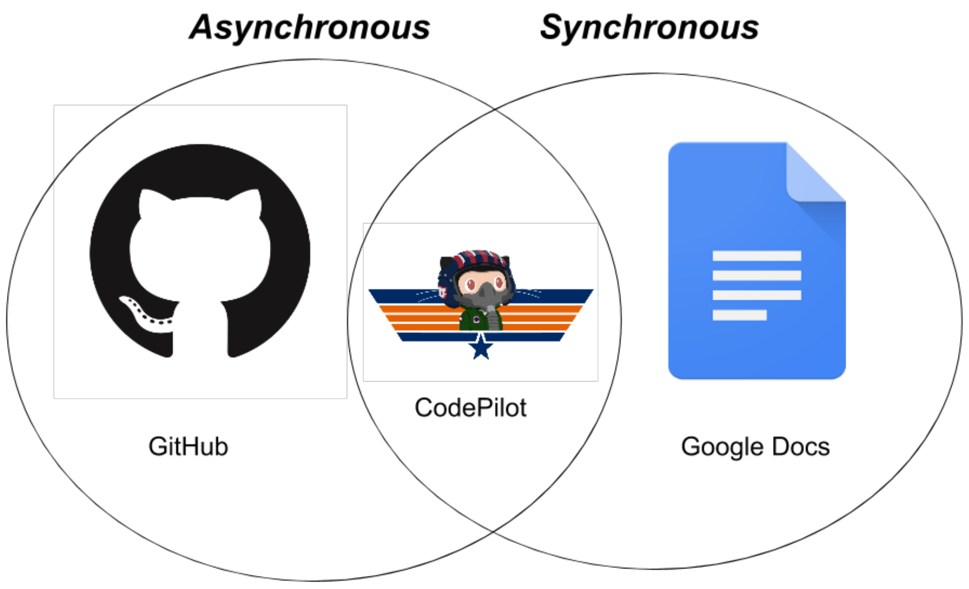


Figure 3.4: CodePilot, GitHub, and Google Docs Venn Diagram

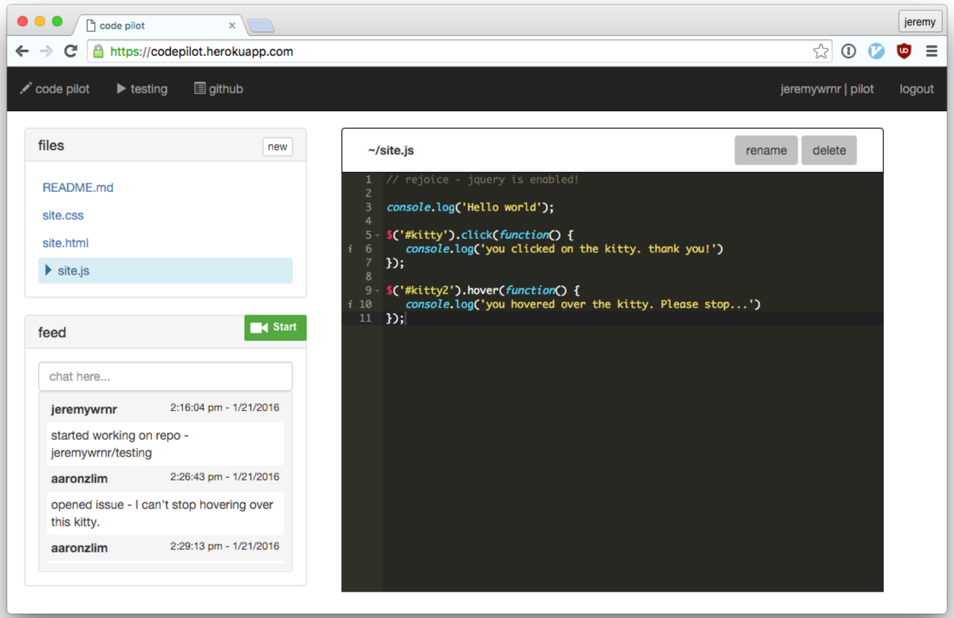
The ability to verify that code is working as expected is a fundamental part of the software development process, and any capable web IDE (integrated development environment) will provide some means of doing so. While there have been efforts to compile and evaluate code in the browser, the latency between running your code and seeing the results is still quite high. However, the technology in this domain continues to improve, and may reach the point where the simplicity and latency no longer dissuades developers from using online IDEs.

3.2 System Overview

CodePilot (currently found at codepilot.xyz) is a Meteor Javascript application that provides project wide real time collaborative editing, using shareJS as a back-end syncing library [27, 30]. It allows users to import a repository of code from GitHub. Once this is complete, they can edit their project and make a new commit based on their edited version of the document and push the new version back to GitHub. Once a repository has been updated on GitHub, changes can be pulled in from the head of a branch, along with whatever commits that have occurred since the last time CodePilot updated its local copy of the repository.

- supports simultaneous file editing
- import any GitHub repo into codepilot
- edit, test, and commit back to GitHub
- supports live branching, testing
- import, create, and close issues
- includes screenshot, live code

CodePilot is a tool meant to help people collaborate on code more seamlessly. The collaboration model was based on that of a pilot and a co-pilot, where the pilot does most of the actual programming, and the co-pilot performs background coding tasks. In general, this is loosely enforced, instead



jeremywrnr 2:16:04 pm - 1/21/2016

started working on repo - jeremywrnr/testing

aaronzlim 2:26:43 pm - 1/21/2016

opened issue - I can't stop hovering over this kitty.

aaronzlim 2:29:13 pm - 1/21/2016

```javascript
// rejoice - jquery is enabled!

console.log('Hello world');

$('#kitty').click(function() {
    console.log('you clicked on the kitty. thank you!')
});

$('#kitty2').hover(function() {
    console.log('you hovered over the kitty. Please stop...')
});
```

Figure 3.5: CodePilot Basic Programming Interface

opting for interface differences to suggest activities more or less prominently based on the user's role. This can include version control management, system testing, online reference lookup, and commenting or documenting source code. A screenshot of the main programming interface is shown in Figure 3.5.

3.3 Programming

When users first login, they see a link to the configuration panel where they can change which repository and branch they are working on. A list of the files in the repository is presented, with a prompt to begin editing them.

This interface is shown in Figure 3.6. Coming back to the main coding page in the future will simply show the file that was opened last.

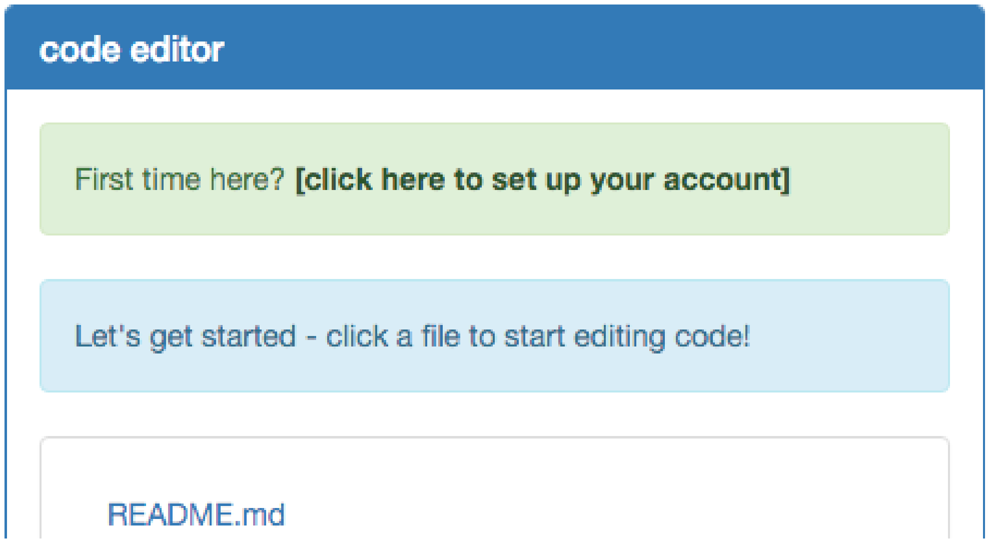
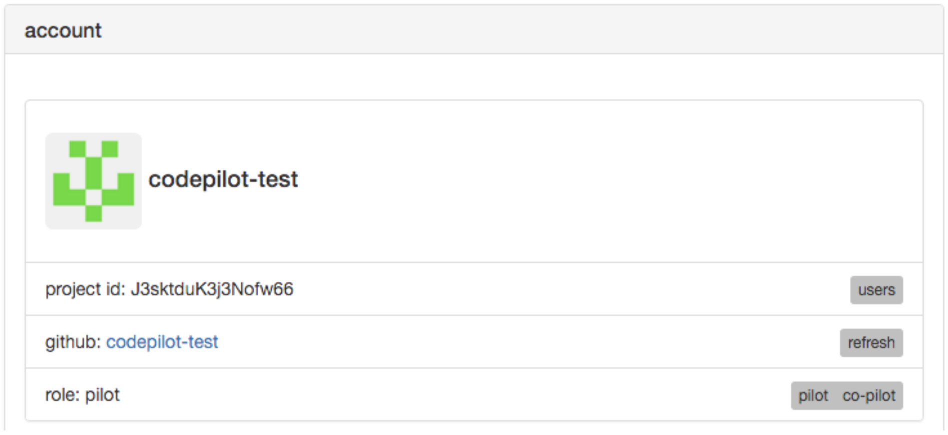

code editor
First time here? [click here to set up your account]
Let's get started - click a file to start editing code!
README.md

Figure 3.6: CodePilot New User Welcome Panel

Clicking on the setup link above will bring them configuration page, where users are able to select which repository and branch they would like to work on. This setup panel is shown in Figure 3.7. They are also able to view additional information about their account, such as who the other CodePilot collaborators are for their current repository, a link to the GitHub account that created their CodePilot profile, and the option to select which role they would have in the development process. These roles were designed to guide parallel development, encouraging the pilot to take charge of the software development process and the copilot to help with testing, documentation, and source control. The UI for the copilot does not contain the commit panel, and the order of the testing pane was rearranged to feature the issues and tasks more prominently. The motivation for this was that to make collaboration even easier, a link could be generated that could be sent to anyone, who can drop in on your session without having to login in or be a collaborator of the GitHub repository. This functionality was not built out however, so the interface differences previously mentioned are the only differences between pilots and copilots.



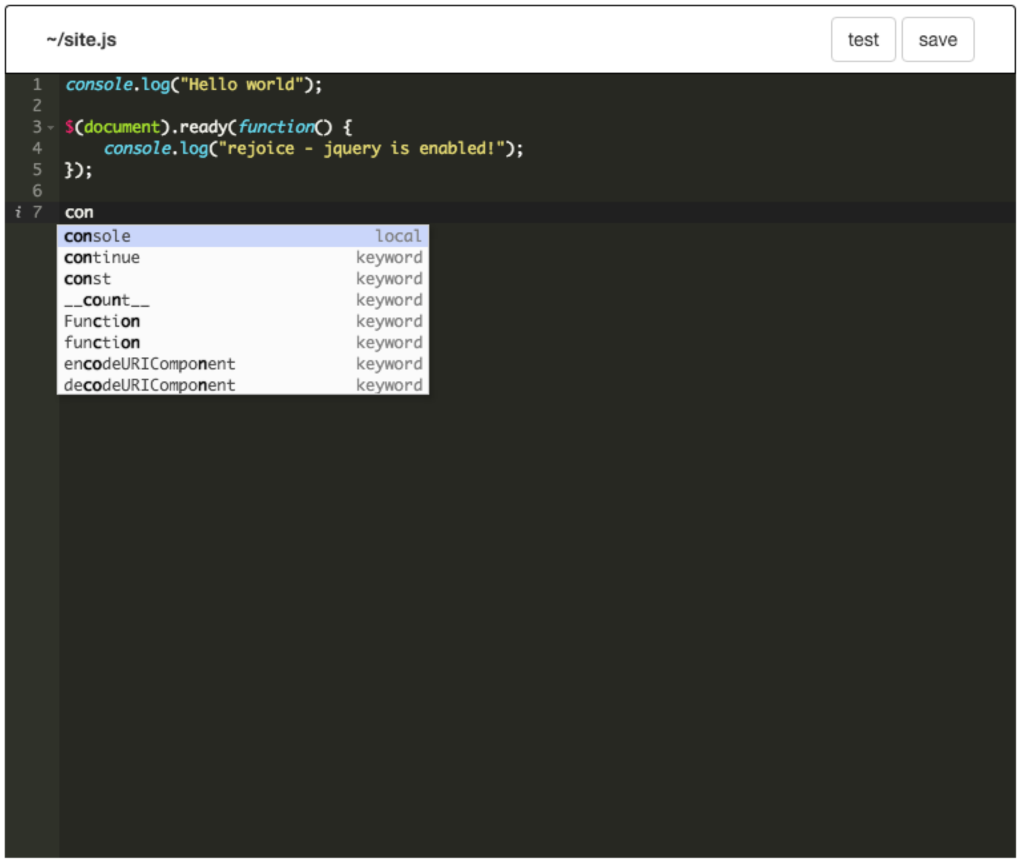
project id: J3sktduK3j3Nofw66

github: codepilot-test

role: pilot

Figure 3.7: CodePilot Configuration Panel

The actual editing pane is based on the popular Ace web editor, which features language recognition, syntax highlighting, and code auto-completion (shown in Figure 3.8). A syncing library (ShareJS [30]) is used to relay edits back to the server and then to all other connected users working on this same project. The file name and path are listed in the upper left corner of the pane, with testing and saving buttons in the upper right. This is to provide easy access to the other cycles of the software development process.



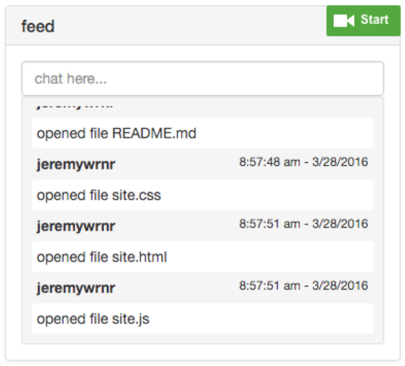
```
console.log("Hello world");

$(document).ready(function() {
    console.log("rejoice - jquery is enabled!");
});

con
```

```
console                 local
continue                keyword
const                   keyword
__count__               keyword
Function                keyword
function                keyword
encodeURIComponent      keyword
decodeURIComponent      keyword
```

Figure 3.8: CodePilot Autocompletion Example

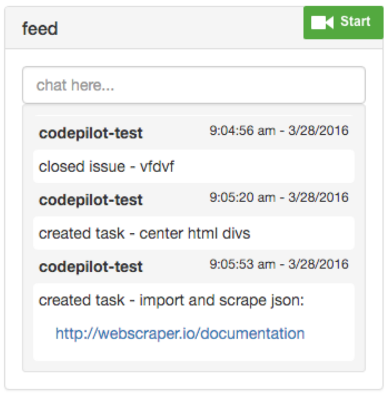
3.4 Activity Feed

CodePilot has an integrated activity and chat feed which updates in real time with the actions of each collaborator. Notifications appear when anyone creates a file, renames a file, opens a file, creates an issue, closes an issue, switches branches, makes a commit, and switches in or out of a repo. It also features a link to opening up a Google Hangouts session where you can video or audio chat with others collaborating on this repository in CodePilot. This feature of CodePilot is displayed in Figure 3.9.



opened file README.md
jeremywrnr 8:57:48 am - 3/28/2016
opened file site.css
jeremywrnr 8:57:51 am - 3/28/2016
opened file site.html
jeremywrnr 8:57:51 am - 3/28/2016
opened file site.js

(a) File notifications

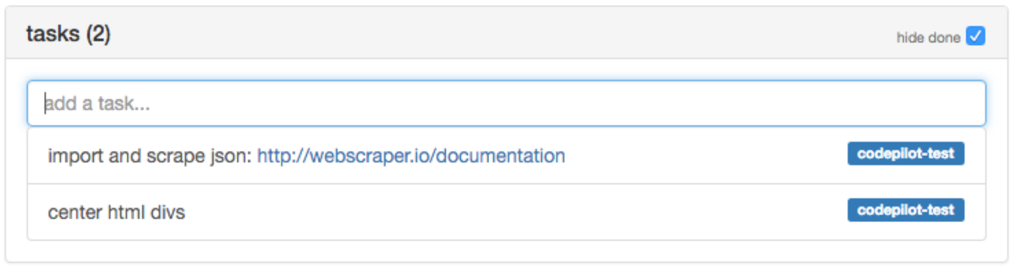


codepilot-test 9:04:56 am - 3/28/2016
closed issue - vfdvf
codepilot-test 9:05:20 am - 3/28/2016
created task - center html divs
codepilot-test 9:05:53 am - 3/28/2016
created task - import and scrape json: http://webscraper.io/documentation

(b) Task notifications

Figure 3.9: CodePilot Event Notification Feed

Tasks are able to created, completed and optionally deleted by collaborators. This system is designed to help outline the work that remains to be done. In the left side of each task there is a small label which displays the username of the task's creator, visible in Figure 3.10. Working with tasks also triggers the activity feed to update, so the main programmer can see the tasks and links coming in.



import and scrape json: http://webscraper.io/documentation codepilot-test
center html divs codepilot-test

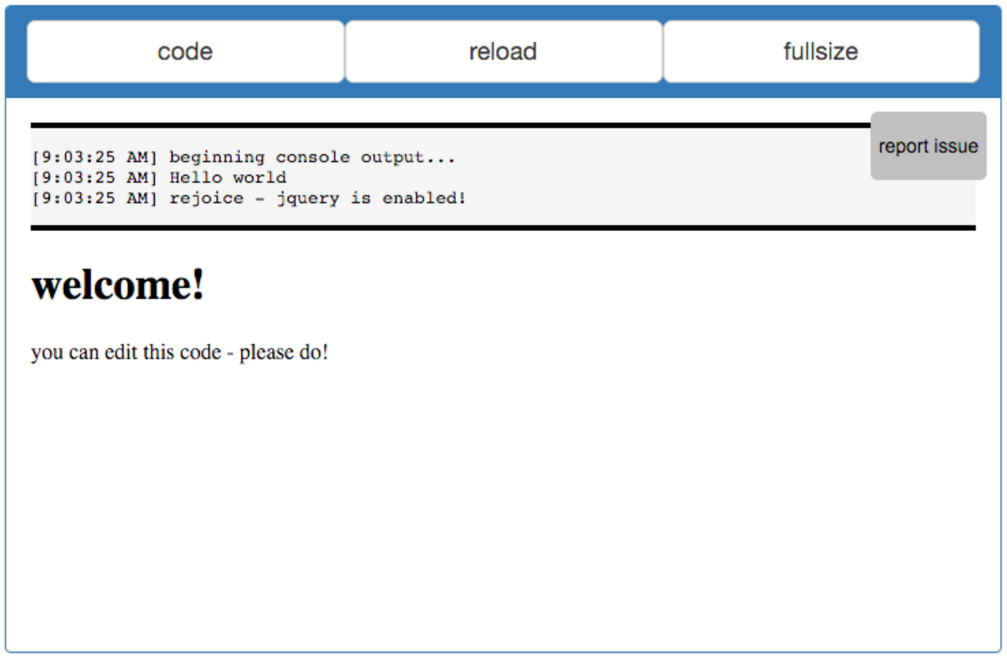
Figure 3.10: CodePilot Task List

3.5 Code Testing

The testing framework inside CodePilot is geared towards debugging basic websites that include HTML, CSS, and Javascript. This decision was made primarily due to the ease of having the browser render these. Since the browser is already made to render websites from these materials, we hypothesized that it would be a most natural to render basic website code since the synchronous nature of application best fit inside a web browser. Shown in Figure 3.11 is a screenshot of CodePilot's testing pane. It features links back to the coding pane, an option to reload code from the editor and re-render it, and a button to move to the full screen debugger. A choice was made to separate the code that was being edited and that which was being inspected in the renderer so it would not reset on a tester who was inspecting the code every time that their partner typed in a new character.

When you enter the testing pane and press the reload button, the code that is in the editor is ported into a difference cache, and injected into CodePilot's renderer. This allows the pilot to keep making changes while the tester debugs the code that was loaded into the browser, making it easier for the work to be done in parallel. Any change that was made by the pilot will not be loaded into the renderer until the reload button is pressed.

Inside the debugging pane, an iframe with injected code renders the users code, parsed from the HTML, CSS, and Javascript files found in their project. The division with text 'beginning console output' is actually the where con-



```
[9:03:25 AM] beginning console output...
[9:03:25 AM] Hello world
[9:03:25 AM] rejoice - jquery is enabled!
```

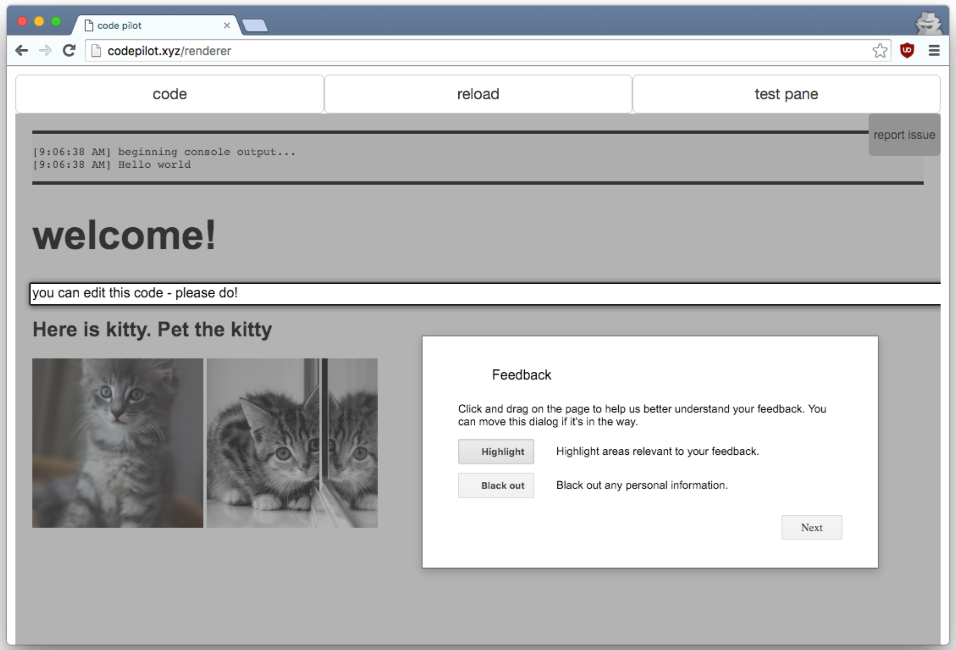
welcome!

you can edit this code - please do!

Figure 3.11: CodePilot Web Interface Debugger

sole.log and console.error are directed to based on the iframes template, which overrides these functions. This way, the developers can easily view the output of their code while debugging. It also features support for jQuery, the popular Javascript library that makes accessing the DOM tree more direct. Below that is the actual rendering of the user's code. Finally, in the top-right corner there is a report issue button, which is hooked into GitHub issues and allows users to easily track problems with their code, or make clear certain parts of the site that they want to change. The issue tracker records all of the state of the rendering, and when an issue is generated (such as in Figure 3.12), it includes a link back to the live, rendered, version of the website.

This tool could be used by clients who want to highlight certain aspects of their site that need to be changed, or by students seeking feedback on their website's overall design.



```
[9:06:38 AM] beginning console output...
[9:06:38 AM] Hello world
```

welcome!

you can edit this code - please do!

Here is kitty. Pet the kitty

Feedback

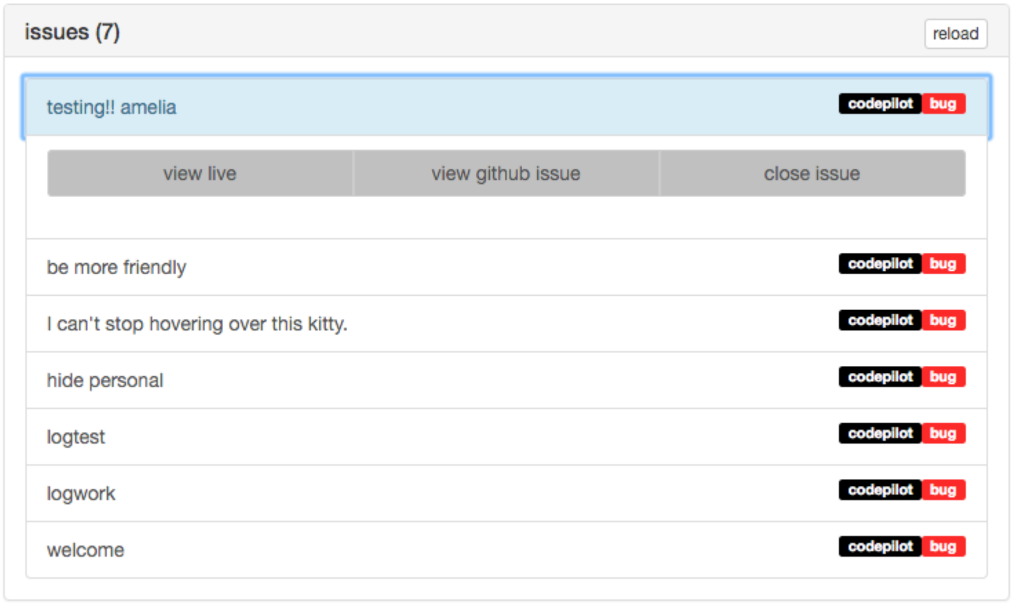
Click and drag on the page to help us better understand your feedback. You can move this dialog if it's in the way.

Highlight areas relevant to your feedback.

Black out any personal information.

Figure 3.12: CodePilot Web Interface Issue Recorder

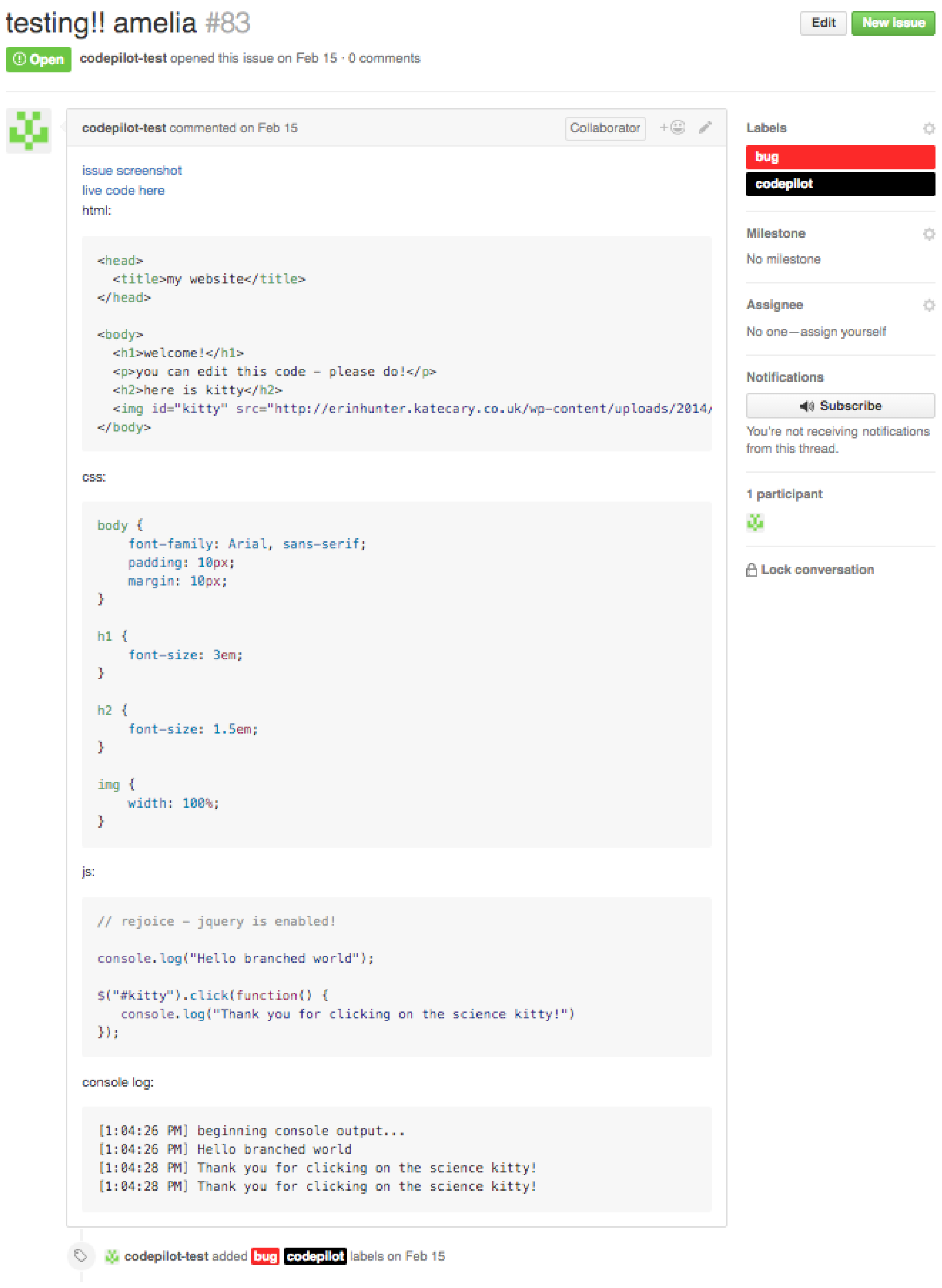
These issues (along with any issues imported from GitHub) are displayed in the CodePilot testing pane. Once you click an issue, a dropdown button list is shown, allowing users to view the issue on GitHub, close the issue (both in CodePilot and in GitHub), and if the issue was generated by CodePilot view the live demonstration of the rendered web code. This interface is shown in Figure 3.13.



testing!! amelia
be more friendly
I can't stop hovering over this kitty.
hide personal
logtest
logwork
welcome

Figure 3.13: CodePilot Issue Management Interface

This is because when an issue is created, the content of each aspect of the renderer (HTML, CSS, Javascript, and console output) is saved, and can be reloaded into the renderer template. This issue will be visible to everyone working on the repository (or if it is a public repository, then the entire public). An example issue that was generated by CodePilot is shown in Figure 3.14 as it appears on GitHub. Additionally, the link to the live version where the issue was found does not require users to be logged in with CodePilot, so if there are other collaborators using the standard git platform, they are still able to view and understand the issue.



testing!! amelia #83

codepilot-test opened this issue on Feb 15 · 0 comments

codepilot-test commented on Feb 15

issue screenshot
live code here
html:

```
<head>
  <title>my website</title>
</head>

<body>
  <h1>welcome!</h1>
  <p>you can edit this code - please do!</p>
  <h2>here is kitty</h2>
  <img id="kitty" src="http://erinhunter.katecary.co.uk/wp-content/uploads/2014/
</body>
```

css:

```
body {
    font-family: Arial, sans-serif;
    padding: 10px;
    margin: 10px;
}

h1 {
    font-size: 3em;
}

h2 {
    font-size: 1.5em;
}

img {
    width: 100%;
}
```

js:

```
// rejoice - jquery is enabled!

console.log("Hello branched world");

$("#kitty").click(function() {
   console.log("Thank you for clicking on the science kitty!")
});
```

console log:

```
[1:04:26 PM] beginning console output...
[1:04:26 PM] Hello branched world
[1:04:28 PM] Thank you for clicking on the science kitty!
[1:04:28 PM] Thank you for clicking on the science kitty!
```

codepilot-test added bug codepilot labels on Feb 15

Labels
bug
codepilot

Milestone
No milestone

Assignee
No one—assign yourself

Notifications
You're not receiving notifications from this thread.

1 participant

Figure 3.14: CodePilot Issue's GitHub Appearance

It should be noted that this testing approach is much more general than web development. Nothing prevents this from being extended to any language or automated testing framework. We chose HTML/CSS/JS since web development is most natural in this setting, but it could have been Ruby, Python, C++, or any other language. The testing here could feature sending the code to a server to run test cases, or even compiling to Javascript in the browser in the future with Opal, Emscripten, or other Javascript source compilers [28, 9].

3.6 Version Control

Users are able to select and import any repository that they are a collaborator on, or repositories that their organization owns and they have access to. This selection pane is shown in Figure 3.15. In addition to this, they are able to easily fork any existing repository (through the GitHub API), and begin working on that in Codepilot. Branches can also be browsed and selected, so that you are able to work on any branch from GitHub. Users are also able to create a new branch based off the your current CodePilot branch, and this change is immediately synced back to with GitHub.

Once a repository is imported, you can use the programming interface to edit the contents of any text files that are located. Images in project repositories may be opened, but no interface is provided to edit them. Once any change is loaded into the renderer or the refresh button is pressed, a

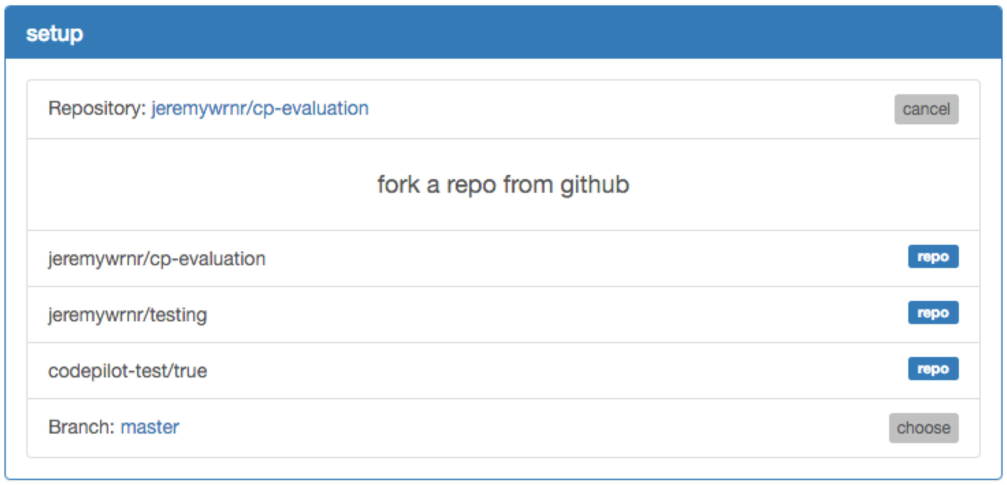
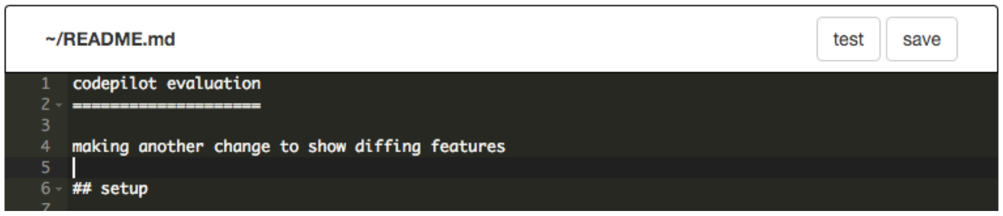


Figure 3.15: CodePilot Repository Selection Interface

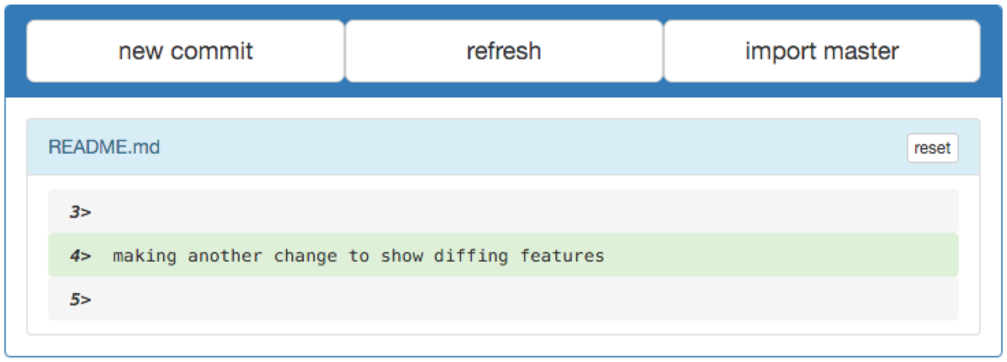
difference will be generated.



```
codepilot evaluation
====================

making another change to show diffing features

## setup
```

Figure 3.16: CodePilot File Management Interface

This is similar to the output that you would see if had entered 'git diff' on the command line, although it is integrated directly into CodePilot. There is a small reset button in the top right section of each diff, so that the user may reset changes that were made but does not want to commit. There is also an 'import master' button, which shows the name of whatever branch is currently being developed in CodePilot. This will pull down the latest commits from GitHub and load that version of the project into CodePilot.



```
3>
4>  making another change to show diffing features
5>
```

Figure 3.17: CodePilot Code Diffing Interface

The commit panel is only visible to pilots, excluding copilots. This goes with our theme of having the main pilot have the final say over what gets committed, but allowing others to help out on the code. Once a commit has been made, the differences will be reset, and the commit that was just created and pushed will be added to the list of commits in both CodePilot and GitHub. This is demonstrated in Figure 3.18.

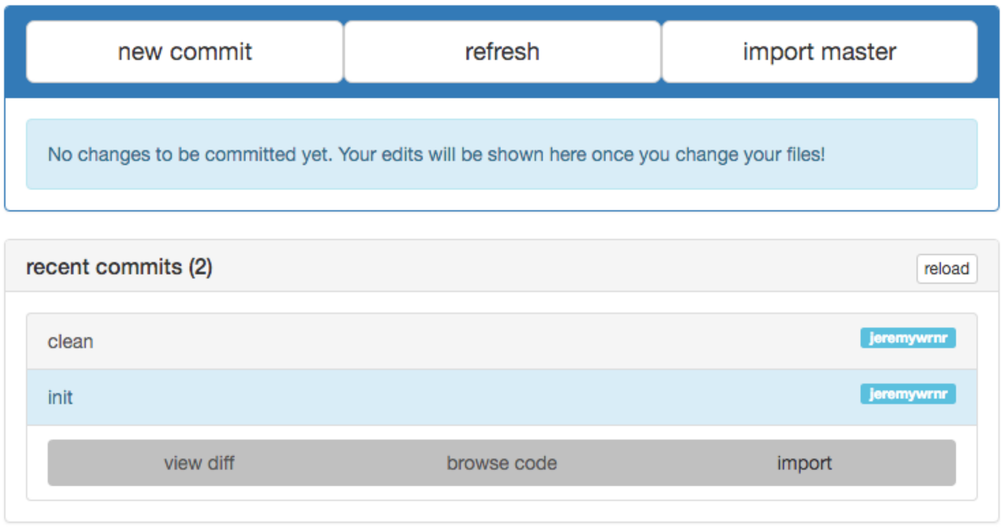


Figure 3.18: CodePilot Commit Interface

Again, it should be noted that this version control approach is more general than just pilot/copilot as we use them here. There is the possible use case of having a single main developer who frames the code and specifications, then guides other developers as they implement the basics. Additionally, this system could be used in a scenario where the pilot seeks help on a specific problem, and allows a copilot to enter the session and try to solve it. The pilot still retains the sole access to push new versions of code to GitHub, but can now seek help from a broader pool of developers.

Chapter 4

Evaluation

4.1 Method

We developed *CodePilot*, a real time collaborative text editor with GitHub integration. We wanted to test whether this tool allows novice programmers to effectively use the git version control system in a collaborative environment, and whether the development tools inside CodePilot encourage members to parallelize their development efforts.

4.1.1 Participants

We recruited six pairs of students that were familiar with both web development and version control to participate in an 1 hour study evaluating how CodePilot performed as a collaborative development tool. We also took screen recordings of 10 out of the 12 study participants. The reason for not recording screencasts of the last 2 participants was that they did not have any screen recording software installed, and we were not prepared with a screen recorder for their operating system. To avoid the study going over the allotted time, we continued without recording the screens of these two participants. All were fairly experienced with version control and web development, as shown in Figure 4.1 and Figure 4.2. The study had eight male and four female students. There were two graduate students, six seniors, three juniors, and one sophomore, represented in Figure 4.3.

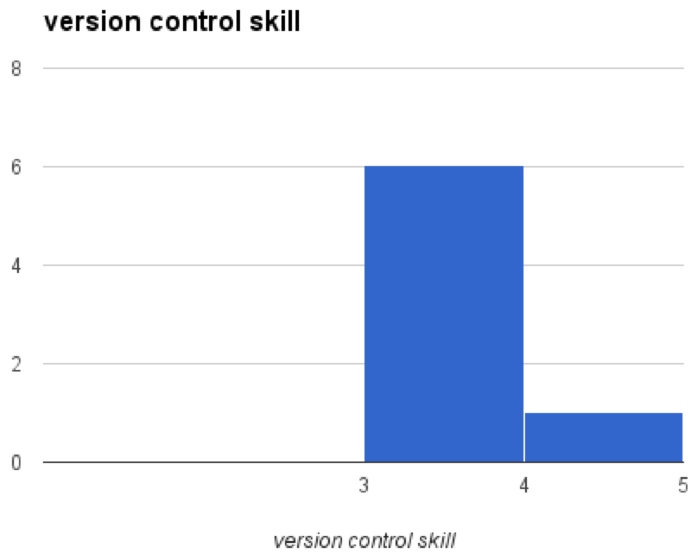


Figure 4.1: Participant Version Control Skill

4.1.2 Procedure

Each hour long pair programming session was structured as follows:

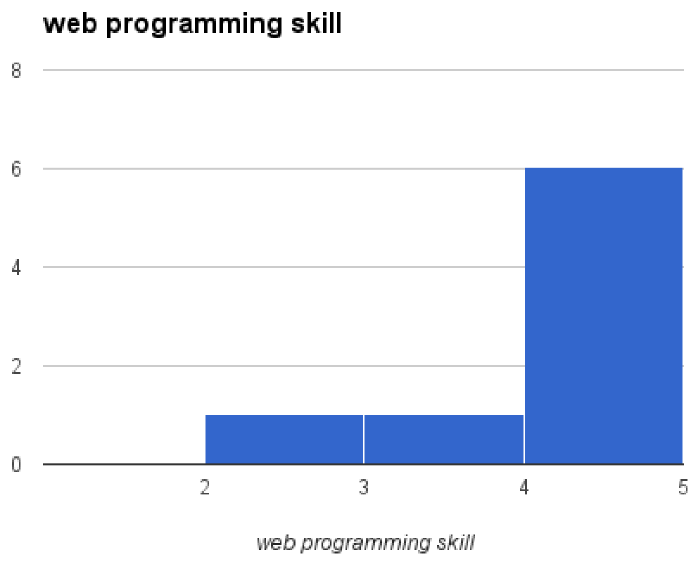


Figure 4.2: Participant Web Programming Skill

1. Explanation of the study and its purpose
2. Signing of study consent forms
3. Demonstrating CodePilot features
4. Starting screen recorder software
5. Signing in, linking with GitHub
6. Collaborating on web development (30 min.)
7. Interviewing about experience (15 min.)

Participants were assigned the following web programming tasks with a single teammate (emulating pair programming), and encouraged to complete as many of them as possible in a 30 minute time frame that was allocated in the middle of the experimental session. The original repo with instructions can be found here:

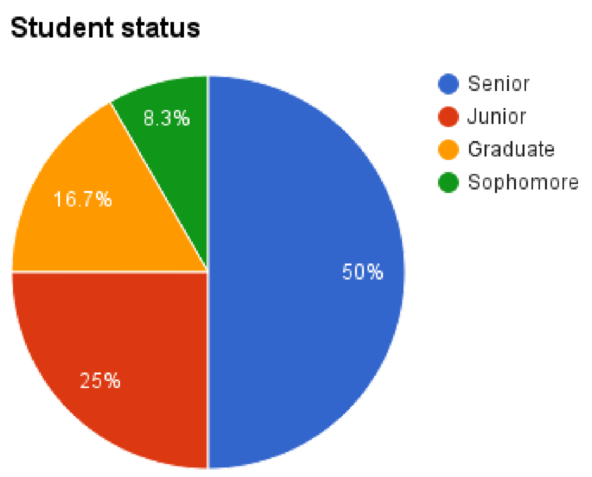


Figure 4.3: Participant Student Status

`https://github.com/jeremywrnr/cp-evaluation`.

Styling (CSS)

- horizontally center the h2 class
- set the background color to something dark
- set the text color to something light
- make the text change color when you hover over it

Structure (HTML)

- create an h2 heading with both your names
- make an unordered list with an id of weather
- make an h3 heading with an id of average
- change the title to be 'recent weather'

Data (Javascript)

- open the JQuery.getJSON documentation
- using this method, parse a hosted json weather data
- add each data point as items to our weather list
- compute the average temperature from this data set
- display this temperature in our temperature heading

Upon completion or the passing of 30 minutes from when the tasks were started, the first person who forked the repository would make a final commit of what they had to their version of the forked original repository. Each participant would then complete a brief interview together about their experiences developing with CodePilot was like. They also were asked to complete a demographic survey afterwards, which asked about their student status (year), gender, and web programming and version control experience. We also performed a short interview of both partners after programming together to learn more about their experience interacting with the system. The aim of this interview was to gain insight into our subject's ease of collaboration with CodePilot, their awareness of their collaborator's actions, their sense of self-efficacy while collaborating and other general CodePilot feedback.

4.1.3 Analysis

When each pair of students first arrived at the session, we discussed what the study would go over, had them sign consent forms for participating as part of our IRB review process, and then gave them an overview of CodePilot.

They were instructed that they could leave at anytime and there would be no negative repercussions for them. After the initial overview and our CodePilot explanation and demonstration, they were given a set of web development tasks, divided up into the structure of the page (HTML), the style of the page (CSS), and the data and interactivity of the page (Javascript). The focus is less on what the task is though, and more on how they worked together to more efficiently complete these tasks. One could note that the division of these tasks could have lent itself to easier pair programming, since the tasks were already categorized into the type of work they required and also the file that work would be done in.

In hindsight, it would have been ideal to perform this study on public Mac computers, since those would have screen recording software built in, and did not want to take time to install additional screen recording software. Thus our data is somewhat fragmented, as we cannot entirely quantify the actions of those last two participants. Ample qualitative notes were taken during the study however, and there was at least one screencast generated per session. Through the study, we measured the amounts of commits (total and per team member). For those that were able to provide a screencast, we measured their time on task, the amount of web searches, and the amount of context switches from their screen recordings of their study sessions.

4.2 Results

4.2.1 Ease of Collaboration

After the collaborative programming session, each pair was interviewed about what their experience using CodePilot to collaborate on web development was like. They were also asked about how it compared to more traditional, asynchronous collaborative workflows (namely collaboration using git or other distributed version control systems). Some participants appreciated the ability to see each other's changes in real-time:

> I think its like a different kind of workflow. We were only able to do that because we were right next to each other at the same time... like for that, I think it was really great. It was great for pair programming.

When asked how making commits was with CodePilot when working with other people, there was a resounding positive response: "Committing was really easy, it worked well". When compared to GitHub, one participant felt that GitHub would continue to be the premier method of long term hosting for large scale projects, but that he preferred CodePilot's synchronous features when collaborating with someone in real time:

> GitHub is always going to be the collaboration tool for large scale projects, but if you want to real time collaborate, this is

> perfect. GitHub is definitely not for real time, I am not expecting to collaborate on the same code at the same time, or there will be a ton of merge conflicts.

When asked if they had used version control with other's before and found themselves in situations where it held them back from actually working effectively on their code, one answered:

> Yea. Sometimes it can get in the way if you are trying to get something done and there is a merge conflict it just takes more time, its annoying. If the person is not familiar with git, that's just another thing on its own.

Another echoed that, while also noting some of the downsides that come with real time synchronization:

> It was nice to see changes being made in real time. There wasn't that much of an issue with the other person typed and what I see on the screen, so that was very helpful because it allows you to see what the other person is doing which then avoids other issues, where you might be doing the same work at the same time, or avoiding a merge conflict. Of course the downside of that is that you are both working on the same thing, and you can't see their cursor so it was hard to experiment often; kind of difficult to do work simultaneously, especially if someone is doing work in the same file above or below you.

When asked about if they had to work with another programmer who was not as experienced with version control if this would be a helpful tool, participants again agreed that it would be much easier to use than teaching that person all of how to use git, avoiding the headache of merge conflicts. Finally, another student told of how they had branched off on a project when working with others, and the large amount of merge conflicts when they tried to bring in their code:

> It's definitely been an issue for me... I branched pretty early, worked on my own thing for a week, then tried to back together and the merge conflicts were just kind of ridiculous... We ended up having to do some git acrobatics, we were cherry picking things and putting them back in the master branch. It would have been nice to a synced up version with other collaborators in the first place so we didn't have to deal with then spending a lot of time trying to figure out how to get someone else's changes back in.

Our participants maintained that version control is an essential skill which is worth learning, but can be challenging at first. One suggested that a system as CodePilot could be nice to include novice programmers, slowly introducing them to version control concepts such as commits and branches (which CodePilot supports):

> Git is super useful to fully know... but that being said, this is definitely a good starting place so that they can at least have

> the idea that a commit is a save point in the code that they can go back to, and then later on introduce branches.

Overall, using a synchronous editor was found to be better at keeping developers in the same context than a standard distributed workflow, where programmers edit different code bases and then later merge them together. On the other hand, there is the chance that something will be edited by your collaborator, which would then impact what you were working on. Collabode has done interesting work with supports optional merging only after a partner's code has passed a test compilation, which could lead to an optimal balance of knowing what your fellow developers are doing without it impeding what you are working on [14].

4.2.2 Awareness of Collaborators

One of the benefits of having a broad web editor is the diminished need to perform context switching, and the corresponding avoidance of mental flow disruptions. Your partner can easily send you relevant links or even create a branch from the current system without having to distract you from whatever coding task you are working on. When asked if they had a good sense of what their teammate was doing, students responded positively, noting that the integrated activity feed (Figure 3.9) helped them stay synced up:

> I feel like even without the cursor syncing, the fact that the log says '[partner] opened site.js' or '[partner] committed css' was

really helpful.

Participants tended to immediately divide up the work and then work on the separate parts right from the initial starting up. Several times they discussed which part they were currently working on, confirming that they would not be editing each other's work. This would not be a conflict in traditional development workflows, but you would still face the problem of later merging code back into the central hosted repository and resolving whatever conflicts could be generated from that.

One feature that multiple participants requested in these interviews were the ability to see where their collaborators cursors were in each file, similar to how Google Docs allows you to jump to where your collaborator's text selection or cursor is in the file.

Interestingly, the distribution of commits between team members was distributed in a manner almost as if we had enforced the pilot and copilot programming schemes. On almost every team, there was one person who took over the version control duties, making all the commits as seen in Figure 4.4. There was one team where both partners authored and pushed commits back to GitHub, but they verbally communicated this, so there was some additional communication. This suggests that people work well when they split up the work, and make clear roles for themselves even if they are not assigned.

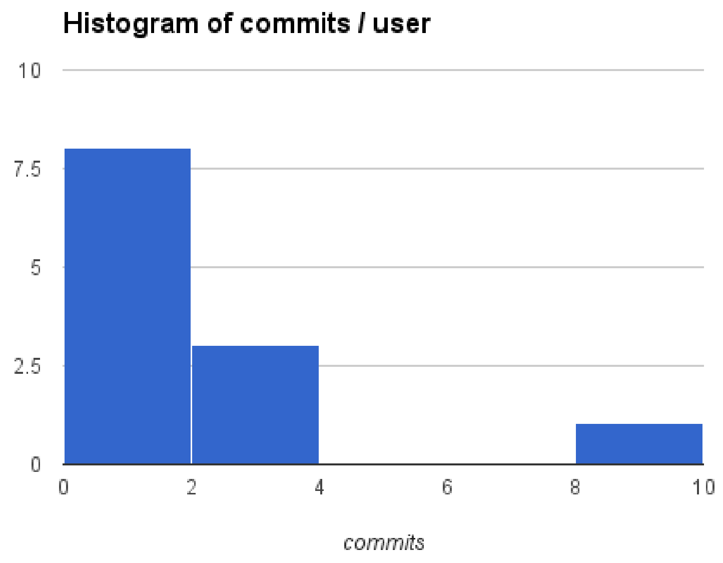


Figure 4.4: Participant Commits

4.2.3 Incorporating Feedback

When asked if using CodePilot was enjoyable to use, all participants generally responded positively. One noted the unique workflow that it encouraged:

> Overall, using it was a pretty pleasant experience, had a lot of fun. I would be a little bit worried using that as an alternative to version control, since I think it is important to learn how to use version control.

After conducting the surveys, we took the study participant's feedback into consideration and implemented several fundamental system improvements and some basic user interface updates. We added the ability to check out any single commit, or the head of the current branch that they were on. This provided users more flexibility in the state of the code that they were editing in the browser, if they wanted to check out what the state of code was at a given commit.

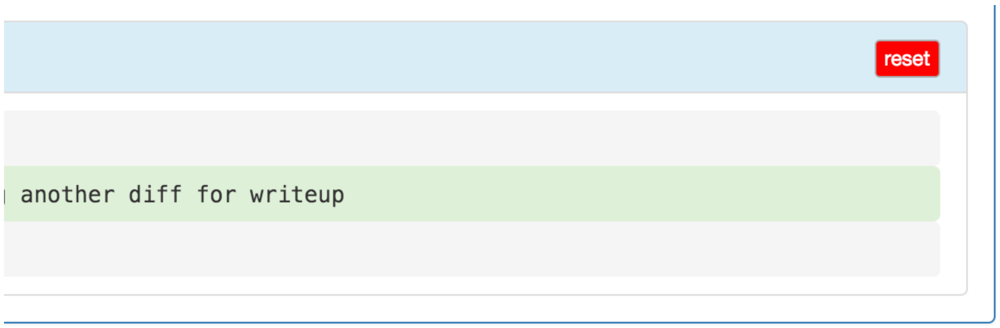
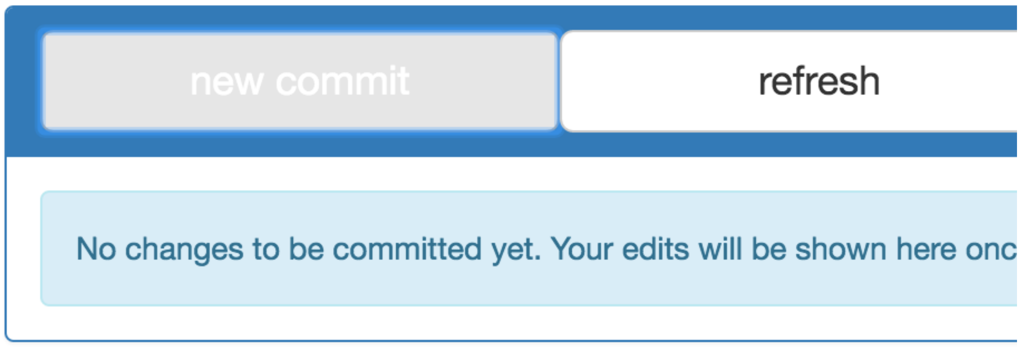


Figure 4.5: CodePilot File Resetting Addition

This included improving the file contents difference recognition mechanism, so users could preview what the differences were between the cached version of their project and the last committed version from GitHub. It also provides users with the options to reset a file back to how it was from the last commit if they accidentally change a file that they did not mean to change. This is shown in Figure 4.5.

We also improving the commit interface, notably disabling the ability to commit anything if there were no differences between the current state of the project in CodePilot and the latest version that was pulled in from GitHub, shown in Figure 4.6. Finally, we augmented navigation between code modes (editing, testing, and saving) so that the switching was less of a mental barrier, and allowed them to do so more easily. One of the most



No changes to be committed yet. Your edits will be shown here onc

Figure 4.6: CodePilot Commit Prevention

requested features in the feedback was to be able to see their collaborators cursors, but this has not been implemented at the time of writing.

Chapter 5

Conclusion

Coding is increasingly collaborative, with the ever popular GitHub's motto being 'social coding' [13]. No longer does the single genius hack away in isolation to bring about the latest innovation. However, parallelization of software development is an incredibly complex task, noted famously by Fred Brooks in the mythical man month [5]. As the future generations of software developers gets trained, it is important to reconsider how we work together on large pieces of complex software, and to ponder on whether there are better ways that collaboration could be done. Git's versioning model is not necessarily intuitive immediately for most new programmers, so an intermediate tool for collaborating on code together could be helpful for them getting used to working with a full fledged version control system such as git in a multi-developer context.

5.1 Future Work

Other modes of task collaboration could be explored, such as implementing a specification test from design documents, while others begin writing the implementation. Frequent checks and syncing mechanisms help avoid the large overhead of syncing massive code changes together. While making commits frequently and atomically is broadly considered best practice, more software can be developed that helps share and encourage these best practices. Developers could begin to embrace a broader set of collaborative models besides the traditional centralized repository, expanding the abilities teams to work dynamically, and for the relevant help to jump in more easily. CodePilot-style online programming sessions could become more popular, to the point where advanced developers could jump from session to session solving more novice programmer's issues. This could possibly be in return for a financial reward, creating bounty hunter style developers. In order for this to work there would be a need to objectively verify that the software bug as described in however it was advertised had since been removed after the novice was helped.

One very viable route that online code evaluations systems could proceed in is extending CodePilot or other collaborative systems to allow the execution of arbitrary code in a browser sandbox, or improve the interfaces to back-end server compilers. Languages such as Ruby already have an implementation in Javascript, the core language that modern web browsers execute. This implementation is called Opal.js [28], and it compiles Ruby code into Javascript which can then be evaluated directly in the browser without having to ping a server to check the results. This could improve the latency and lack of support for more complex projects in the software verification cycle that hinders current online IDE's most predominantly.

Currently, the committing pane of CodePilot where users make new commits is limited to committing entire files (or sets of files). One augmentation that could be done is to add functionality to commit just part of a file's changes, such as in 'git add –patch' or 'git add –interactive' [8]. Furthermore, this augmented committing process could allow committing all changes from a specific user (or a set of user). Each edit operation already has an author identification tag, which could then used to filter out which parts of the code to include in that commit.

There are also many improvements that could be made in terms of shortening the feedback loop between editing code and inspecting the differences in output. CodePilot's HTML/CSS/JS renderer offered some feedback loops, but there are already much more complex online code running services, such as pythontutor.com for visualizing code execution or Travis-CI for running an arbitrarily more complex suite of tests. Systems such as Mad Eye already support sharing access to your local machine's terminal, thus avoiding the need for the advanced helper to replicate the novice's development system [26]. There are potential security issues with allowing other people direct access to your local machine though, so perhaps with more limiting modifications this could be a solid means of collaboration. Alternatively, the development environment could be bundled up and then cloned in a web IDE, where it could run actively. This is closer to what Cloud9's web IDE seeks to provide, where you are even give a limited BASH shell to run commands on the remote server [6].

5.2 Contributions

The contributions of this work are building and analyzing how future modes of collaboration could be combined to work together to allow developers to maintain a closer shared context without impeding each other's progress. This work explores new ways for collaborative development to be less encumbering for budding programmers, and allow for more parallelized development for experienced programmings.

As software development becomes more popular and collaborative, it is important to examine the tools that people use to collaborate, along with how these tools are actually used in practice. CodePilot is just one example of a system that can extend how developers traditionally work together, and lower boundaries to novice programmers when getting involved with version control.

CodePilot combines synchronous and asynchronous collaboration, encouraging a greater sense of awareness for what your collaborators are working on, but this does not come without costs. Namely, the additional context gained from synchronously editing files is offset by the inability to edit code without effecting your collaborators. This can be mitigated by compartmentalizing which codebase each person works on, or splitting up collaborators so that one or two developers are mainly writing code while others write tests or documentation.

Version control has made the world of software development a better place, but is still often challenging to use when collaborating with a large group. GitHub has done a great job in making it more accessible, but there is still issues that people encounter when working together, such as merging divergent copies of the repository. Tools can be built (such as CodePilot) that bring together the asynchronous real-time editor with that of the complex back-end version control system, as well as some method of software verification, which was implemented using the web renderer. CodePilot helps explore this increasingly viable and important space of collaborative programming systems.

Bibliography

[1] Jacob T. Biehl, Mary Czerwinski, Greg Smith, and George G. Robertson. Fastdash: A visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1313–1322, New York, NY, USA, 2007. ACM.

[2] Bitbucket. `https://bitbucket.org/`, 2016.

[3] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. Example-centric programming: Integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 513–522, New York, NY, USA, 2010. ACM.

[4] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1589–1598, New York, NY, USA, 2009. ACM.

[5] Frederick P. Brooks, Jr. *The Mythical Man-month (Anniversary Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[6] Cloud9. `https://c9.io/`, 2016.

[7] P. Dewan, P. Agarwal, G. Shroff, and R. Hegde. Distributed side-by-side programming. In *Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on*, pages 48–55, May 2009.

[8] Git Documentation. `https://git-scm.com/docs/git-add`, 2016.

[9] Emscripten. `https://github.com/kripken/emscripten`, 2016.

[10] Hongfei Fan, Chengzheng Sun, and Haifeng Shen. Atcope: Any-time collaborative programming environment for seamless integration of real-time and non-real-time teamwork in software development. In *Proceedings of the 17th ACM International Conference on Supporting Group Work*, GROUP '12, pages 107–116, New York, NY, USA, 2012. ACM.

[11] Joseph Flieger and James Dean Palmer. Supporting pair programming with javagrinder. *J. Comput. Sci. Coll.*, 26(2):63–70, December 2010.

[12] Neil Fraser. Differential synchronization. In *Proceedings of the 9th ACM Symposium on Document Engineering*, DocEng '09, pages 13–20, New York, NY, USA, 2009. ACM.

[13] GitHub. `https://github.com/about`, 2016.

[14] Max Goldman, Greg Little, and Robert C. Miller. Real-time collaborative coding in a web ide. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 155–164, New York, NY, USA, 2011. ACM.

[15] Max Goldman and Robert C. Miller. Test-driven roles for pair programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '10, pages 13–20, New York, NY, USA, 2010. ACM.

[16] Philip Guo. Pythontutor.com. `http://pythontutor.com/`, 2016.

[17] Philip J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[18] Lile Hattori and Michele Lanza. Syde: A tool for collaborative software development. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, pages 235–238, New York, NY, USA, 2010. ACM.

[19] R. Hegde and P. Dewan. Connecting programming environments to support ad-hoc collaboration. In *Proceedings of the 2008 23rd IEEE/ACM*

International Conference on Automated Software Engineering, ASE '08, pages 178–187, Washington, DC, USA, 2008. IEEE Computer Society.

[20] A. Z. Henley, A. Singh, S. D. Fleming, and M. V. Luong. Helping programmers navigate code faster with patchworks: A simulation study. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, pages 77–80, July 2014.

[21] Christopher D. Hundhausen and Adam S. Carter. Supporting social interactions and awareness in educational programming environments. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU '14, pages 55–56, New York, NY, USA, 2014. ACM.

[22] Christopher D. Hundhausen, Adam S. Carter, and Olusola Adesope. Supporting programming assignments with activity streams: An empirical study. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 320–325, New York, NY, USA, 2015. ACM.

[23] D. L. Jones and S. D. Fleming. What use is a backseat driver? a qualitative investigation of pair programming. In *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*, pages 103–110, Sept 2013.

[24] Neha Katira, Laurie Williams, and Jason Osborne. Towards increasing the compatibility of student pair programmers. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 625–626, New York, NY, USA, 2005. ACM.

[25] Thomas D. LaToza, W. Ben Towne, Christian M. Adriano, and André van der Hoek. Microtask programming: Building software with a crowd. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 43–54, New York, NY, USA, 2014. ACM.

[26] MadEye. `https://madeye.io/`, 2016.

[27] Meteor. `https://www.meteor.com/`, 2016.

[28] Opal.rb. `http://opalrb.org/`, 2016.

[29] David Preston. Using collaborative learning research to enhance pair programming pedagogy. *SIGITE Newsl.*, 3(1):16–21, January 2006.

[30] ShareJS. `https://github.com/share/ShareJS`, 2016.

[31] Beth Simon and Brian Hanks. First-year students' impressions of pair programming in cs1. *J. Educ. Resour. Comput.*, 7(4):5:1–5:28, January 2008.

[32] Lynda Thomas, Mark Ratcliffe, and Ann Robertson. Code warriors and code-a-phobes: A study in attitude and pair programming. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 363–367, New York, NY, USA, 2003. ACM.

[33] Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. Strengthening the case for pair programming. *IEEE Softw.*, 17(4):19–25, July 2000.

[34] Laurie Williams, Charlie McDowell, Nachiappan Nagappan, Julian Fernald, and Linda Werner. Building pair programming knowledge through a family of experiments. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering*, ISESE '03, pages 143–, Washington, DC, USA, 2003. IEEE Computer Society.