CodePilot: Scaffolding End-to-End Collaborative Software Development for Novice Programmers

Jeremy Warner
UC Berkeley
Berkeley, CA, USA
jeremy.warner@berkeley.edu

Philip J. Guo
UC San Diego
La Jolla, CA, USA
pg@ucsd.edu

ABSTRACT

Novice programmers often have trouble installing, configuring, and managing disparate tools (e.g., version control systems, testing infrastructure, bug trackers) that are required to become productive in a modern collaborative software development environment. To lower the barriers to entry into software development, we created a prototype IDE for novices called CodePilot, which is, to our knowledge, the first attempt to integrate coding, testing, bug reporting, and version control management into a real-time collaborative system. CodePilot enables multiple users to connect to a web-based programming session and work together on several major phases of software development. An eight-subject exploratory user study found that first-time users of CodePilot spontaneously used it to assume roles such as developer/tester and developer/assistant when creating a web application together in pairs. Users felt that CodePilot could aid in scaffolding for novices, situational awareness, and lowering barriers to impromptu collaboration.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

Author Keywords

novice programmers; pair programming; collaborative IDE

INTRODUCTION

Learning to code is now a popular activity around the world due in part to the abundance of software engineering job opportunities [12]. Enrollments within university computer science departments, coding bootcamps, and programming-based Massive Open Online Courses have been skyrocketing. The majority of programming courses (both in-person and online) focus on the actual mechanics of writing code. However, to become a productive professional software engineer, novices need to learn much more than how to write code properly. They need to also get accustomed to working within a modern collaborative software development environment – coordinating with their teammates using tools such as version control systems, testing infrastructure, and bug/issue trackers [1, 3, 10]. Although in theory anyone can install, configure, and manage a plethora of free open-source tools for all of these purposes, in practice the logistical overhead of learning to manage these tools can be extremely daunting to novices who are already expending large amounts of effort learning the mechanics of programming itself [2]. To lower the barriers to entry for novices to get acquainted with collaborative software development, we created a prototype IDE called *CodePilot*.

CodePilot enables multiple programmers to coordinate in real time throughout several major phases of modern software development – writing, running, and testing code, generating bug reports, managing the bug/issue tracker, and managing the version control system – with no required external software setup or configuration. It serves as convenient "training wheels" for novices to quickly get practice with techniques such as pair programming [13, 14] and test-driven development [7], which have been shown to improve productivity, software quality, and knowledge sharing. For instance, when a novice is paired up with an expert (e.g., a tutor or teaching assistant in a course), they can learn by serving as an apprentice and watching the expert work alongside them within CodePilot.

An eight-subject exploratory study found that novice students were able to use CodePilot to spontaneously assume roles such as developer/tester and developer/assistant when creating a web application together in pairs. Users felt that CodePilot could aid in scaffolding for novices, situational awareness, and lowering barriers to impromptu collaboration.

This paper contributes a novice-oriented IDE that scaffolds collaborative development activities such as pair programming, testing, bug reporting, and version control management.

RELATED WORK

To our knowledge, CodePilot is the first attempt to integrate real-time collaborative coding, testing, bug reporting, and version control management into a unified system. Prior real-time collaboration tools for software development are focused only on writing code itself rather than on the end-to-end development process. These are available as plug-ins for existing IDEs (e.g., CollabVS for Visual Studio [9] and ATCoPE for Eclipse [5]) and in web-based Google Docs-like collaborative editors such as Cloud9 (https://c9.io/), MadEye (https://madeye.io/), and Collabode [6]. Another class of real-time software collaboration tools (e.g,. FASTDash [1], Saros [11], Syde [8]) provides situational awareness by informing developers of what development tasks, files, and code

CHI 2017, May 06–11, 2017, Denver, CO, USA

ACM 978-1-4503-4655-9/17/05...$15.00
DOI: http://dx.doi.org/10.1145/3025453.3025876

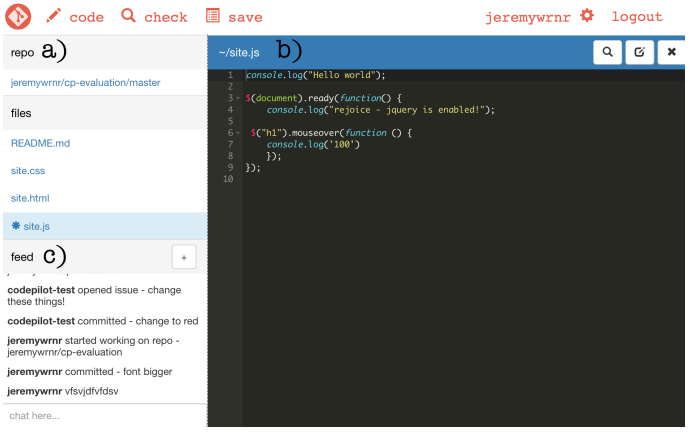

Figure 1. CodePilot is a web-based IDE designed for novices. Multiple users can connect to a shared session via a URL. a) The file selector enables each user to add, delete, and edit files. b) The code editor pane is a multi-user text editor similar to Google Docs. c) The activity feed shows all users' activities and enables inline text chat.

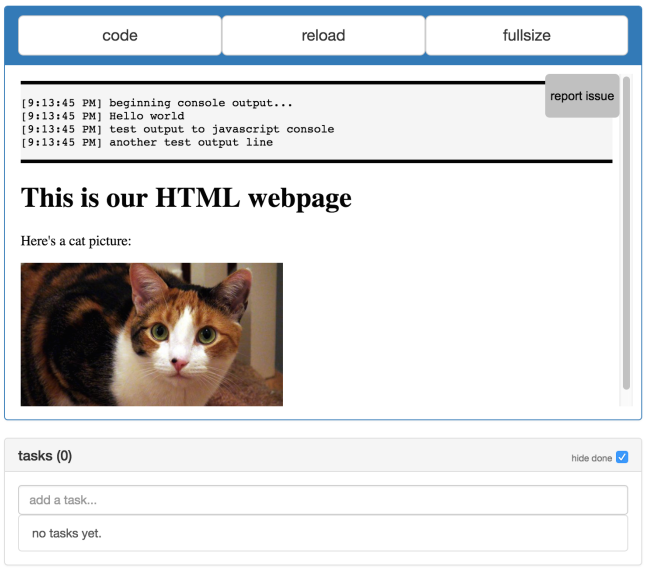

Figure 2. When the user clicks the "test" icon in Figure 1b, the code editor pane is replaced with a testing pane. This testing pane renders the user's current HTML/CSS/JavaScript code in the browser, along with a JavaScript console (top) and an editable task queue linked to GitHub's issue tracker for the project (bottom). Clicking the "report issue" button at the upper right pops up the bug reporting tool shown in Figure 3.

edits their partners are working on in real time. These tools directly inspired CodePilot's activity feed feature (Figure 1c).

FORMATIVE OBSERVATIONS AND DESIGN GOAL

We were motivated to create CodePilot after spending a semester observing students at our university's weekly student hacking (i.e., hobby coding) group meetings. One recurring barrier we noted from observations and interviews with novice attendees was the intimidating logistical overhead of setting up disparate development tools such as testing infrastructure, bug/issue trackers, and version control repositories, especially when doing so involved intricate knowledge of command-line and configuration file tweaking. As a result of this setup overhead, novices were often reluctant to "dive in" to work on more substantive projects that involved collaboration, testing, and version control; instead, they often lingered in the background and watched passively while the more experienced students worked on their projects during group meetings.

Based on these observations, we wanted to create an IDE to get these novices started with collaborative software development without the discouraging overhead of external tool setup. Production-grade web IDEs such as Cloud9 offer intricate configuration options and debugging mechanisms, but setting these up can also serve as a deterrent for novice programmers. Thus, our main design goal for CodePilot was to embed scaffolding for these tools into a simple unified interface, sacrificing advanced power-user features for a streamlined novice-friendly workflow. Even though we were directly motivated by in-person use cases we observed in the university computer lab, we designed CodePilot to work equally well in remote online settings such as Massive Open Online Courses.

CODEPILOT SYSTEM DESIGN AND USAGE SCENARIO

CodePilot is a web-based IDE (integrated development environment) where multiple users can connect to a collaborative software development session via a single URL. Project repositories can be imported from `GitHub`, a popular Git repository hosting platform. Once imported, users collaborate on the project in real time, and can then commit and push any of their changes back to GitHub. CodePilot is implemented using MeteorJS, employs Firebase/Firepad for document synchronization, the Ace web editor, and GitHub's Developer API. It is open-sourced under the MIT License at `https://github.com/jeremywrnr/codepilot`. The authors also host a running instance of CodePilot at `http://codepilot.xyz`.

We illustrate the features and design rationale of CodePilot using an example usage scenario. Alice and Bob are two students working together to create a HTML/CSS/JavaScript web application. Alice first logs into CodePilot with her GitHub account and creates empty HTML, CSS, and JavaScript files for her new project. She then emails Bob the unique URL of her CodePilot session. When Bob visits that session URL, both see the IDE interface shown in Figure 1.

Writing Code

Alice and Bob can each write code in any project file just like in a regular IDE (Figure 1b). If they are editing the same file concurrently, their changes are synchronized in real-time like in Google Docs, using Firepad to synchronize edits. They gain situational awareness [1, 8, 11] and can coordinate via the *activity feed* (Figure 1c), which shows a real-time stream of events (e.g., "Alice created issue X", "Bob committed changes to GitHub") and allows inline text chat. Alice and Bob can now engage in pair programming [13, 14] where both work on the same file at once or in side-by-side programming [4] where each works on their own file while coordinating via inline chat. The Ace web-based text editor provides language-specific syntax highlighting and code auto-completion. CodePilot supports an arbitrary number of concurrent collaborating users subject to network and CPU limitations.

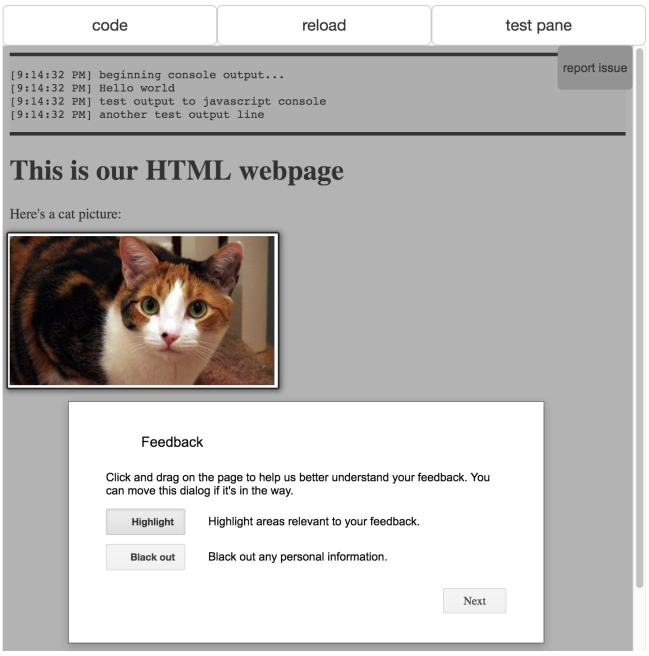

Figure 3. When the user clicks the "report issue" button in the testing pane, a pop-up appears that allows the user to highlight parts of the rendered webpage for emphasis and to black out private data prior to taking a screenshot. Then the user can write a textual bug report and CodePilot submits it to GitHub's issue tracker along with that screenshot and a snapshot of the current code, to facilitate reproducibility.

Running, Testing, and Debugging Code

Let's say Alice is the primary software developer on their project, and Bob is serving as her tester. As Alice writes new code, Bob can click the "test" button in Figure 1b at any time to save a snapshot of their code and enter a testing pane shown in Figure 2, which overlays over the editor pane in Figure 1b.

CodePilot's testing pane renders the snapshot of the HTML/CSS/JavaScript code that Bob just saved and includes an inline JavaScript console to support `console.log` print debugging. Using this pane, Bob can check that the HTML/CSS is rendering as expected and that dynamic JavaScript events are behaving appropriately. Note that Alice's new code edits will *not* be incorporated into the testing pane until Bob clicks the "reload" button to explicitly pull in new changes. This design enables Bob to test a known code snapshot without risking it invisibly changing without his knowledge.

For our current prototype, we created and evaluated a testing pane for HTML/CSS/JS web applications, but one could imagine creating other testing panes for different kinds of apps or platforms. With additional engineering effort, it is also feasible to hook CodePilot into external testing services, such as a class grading platform like Gradescope (`https://gradescope.com/`) or a continuous integration testing service like TravisCI (`https://travis-ci.org/`).

Bob can chat with Alice in the activity feed to give feedback, provide documentation links, and add tasks such as feature requests to an asynchronous queue for Alice to address later. This queue is synced with GitHub's issue tracking system so

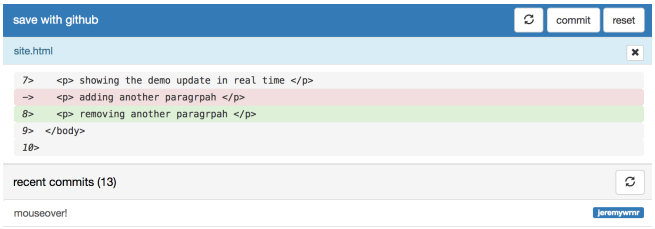

Figure 4. Before performing a commit, users are shown a visual diffing interface inside CodePilot, which provides affordances similar to `git status` and `git reset` by summarizing changes since the last commit and providing a mechanism to reset the file's content to its last committed version.

that teammates can see tasks on the GitHub website, even when not logged into CodePilot. Existing GitHub issues are also automatically imported into CodePilot, so all issues can be opened or closed from both platforms.

Generating Bug Reports

If Bob finds a bug when he is testing, he can click the "report issue" button in the testing pane, which pops up the bug reporter (Figure 3). Using this tool, Bob can highlight areas on the webpage for emphasis and black out private information that he does not wish to share as a screenshot in his bug report. In Figure 3, Bob highlights the cat image for emphasis and then writes a full textual report describing the bug he found.

When Bob submits his bug report, CodePilot creates a new GitHub issue (using its issue tracker) containing his report text, screenshot with highlights/blackouts, and a snapshot of the underlying code that Bob was testing. This way, anyone can reproduce this bug by checking out that code snapshot. Alice now gets notified of Bob's new bug report in her activity feed. At her convenience, she can switch over to her own testing pane to see the contents of Bob's report and check out his snapshot to reproduce and investigate the bug herself.

The main design consideration in the bug reporter was to collect as much relevant information inline as possible so that a tester such as Bob can quickly report bugs without context switching. Additionally, this graphical selection method of debugging was chosen because it is intuitive for web interface designs, which are inherently visual and therefore often will have bugs that can be recognized visually.

Managing the Bug/Issue Tracker

CodePilot integrates tightly with GitHub, leveraging it as both a bug/issue tracker and a Git-based version control system. Alice and Bob can use the testing pane (Figure 2) to directly add, edit, and resolve issues on GitHub, and they can use the version control pane (Figure 4) to check out, commit, visually diff, and revert code versions in Git.

GitHub integration makes it possible for Alice and Bob to write code, test code, report bugs, and commit changes to version control entirely within CodePilot while coordinating via the activity feed. Without this feature, they would need to use external apps such as the Git command-line client or the GitHub website to manage the bug/issue tracker and version control system. Having all of these workflow phases unified

into one IDE also makes it easy for other teammates to join this project by visiting a URL, as long as they have a GitHub account and the proper access permissions.

Managing the Version Control System

CodePilot's fully-synchronous nature prompted some design compromises and us choosing sensible defaults with regards to how to best interface with GitHub's API for asynchronous-style collaboration. Each commit serves as a complete snapshot, so if code is checked out at a specific commit with CodePilot, users can append to the existent history on GitHub, but cannot rewrite it. CodePilot loads all the code of the chosen commit, which by default is the HEAD of the active branch. Then, when committing back to GitHub, the new commit is simply stacked on top of the HEAD of the active branch.

One immediately apparent problem occurs when trying to collaborate with users outside of CodePilot, as the repository histories diverge. In an educational setting to train novices, we recommend for everyone to work within CodePilot. But to ameliorate this problem when it does come up, CodePilot allows for the current state and history to be manually pulled in from GitHub, but in the process overwrites any uncommitted changes with the newest commit. It also separates versions of code by branch, so users working on different branches in CodePilot do not interfere with each other's work. New branches can be created from those inside CodePilot, allowing for multiple tracks of development inside a single project.

When creating a new commit, there can be only a single author; to attribute work done by others, one workaround is for the commit author to explicitly include who they worked with in the commit message. Before committing, users are shown a visual summary of changes (Figure 4) and given the chance to undo any changes they do not want to save. There is no option to commit a subset of files though, as we opted to keep the simpler model of saving the entire project as a snapshot rather than confusing users with partially-committed directories.

CodePilot also features the ability to fork any GitHub repository accessible to the current user. For example, with a team assignment skeleton repository in a class, students can easily fork it and immediately begin working together, avoiding the initial hassle of environment setup while gaining the additional benefits of version control and real-time collaboration.

Finally, there is no notion of explicitly 'git-pushing' one's code back to GitHub when working within CodePilot. Instead, GitHub serves as a simple reference log of project snapshots, so whenever a commit is made, it automatically gets added to the HEAD of the current active branch. This abstracts away a fair deal of the networking complexity and distributed system management that are often error-prone for novice Git users.

EXPLORATORY STUDY OF NOVICE PAIR PROGRAMMING

To gauge novice perceptions of CodePilot, we ran an exploratory user study where four pairs of students (all first-time users) used it to collaboratively build a simple web application.

We recruited 8 computer science students from our university with low to moderate levels of web programming experience (6 male, 2 female). We paired them up randomly and gave each pair 30 minutes to build a simple weather display app with four components – `html`: create a template for the weather display, `styling`: style it according to CSS specifications, `data-fetch`: use JavaScript to fetch JSON data from an online weather data source, `data-display`: dynamically parse, format, and display the fetched data on the webpage. We set up CodePilot's code editor pane with three skeleton files (HTML, CSS, JS) that participants had to fill in, and we began each session with a 10-minute tutorial on CodePilot's features.

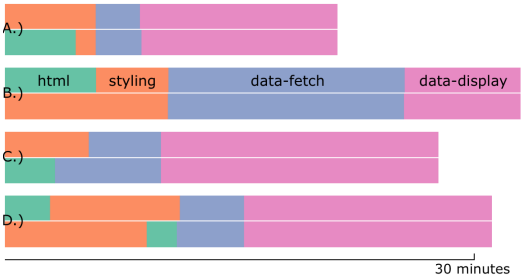

Figure 5. The amount of time that each pair of programmers spent on each component of the web programming task using CodePilot. Each bar is a pair, split into two halves, one for each programmer in the pair.

Using pilot tests (CodePilot pilots!) we refined this task to both resemble a realistic web app and one that can be implemented in 30 minutes. We recorded participants' monitors and conducted 15-minute semi-structured follow-up interviews to assess their perceptions of CodePilot and how it compared to software development tools they have used in the past.

We let each pair work however they wanted within CodePilot so that we could observe what kinds of interactions emerge naturally. As expected, pairs varied in how much they worked collaboratively and also how much time they took to complete the task. No pair quit the study before completion.

Figure 5 shows how each pair split their time. In general, participants worked separately on the simpler `html` and `styling` components and then joined together to complete the more challenging `data-fetch` and `data-display` parts. Since each pair was co-located to make it easier for the research team to observe them working together (and to replicate the in-person student hacking club experience that motivated us to develop CodePilot), they always coordinated verbally. However, in follow-up interviews, several mentioned that they could have easily used text chat if necessary.

Figure 6 shows a detailed timeline of activities that each pair engaged in, with five types of events – *view*: looking at a source code file (HTML, CSS, or JavaScript) in the code editor, *edit*: actively editing a file in the editor, *test*: testing the app in the testing pane, *docs*: reading documentation webpages in the browser, *git*: managing version control pane. We told each pair to make at least one Git commit when they finished, but some made additional commits to save intermediate progress.

Analyzing temporally co-occurring events in Figure 6 reveals that participants were able to use CodePilot to spontaneously assume well-known collaborative software development roles without any prompting by the research team. The most common role was ***traditional pair programming*** where both par-

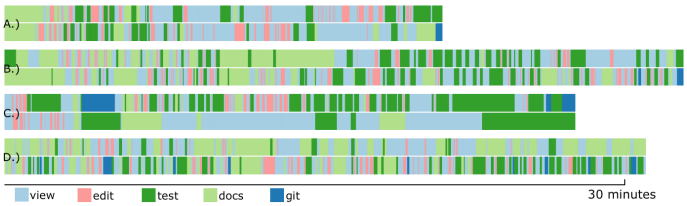

Figure 6. The amount of time that each pair spent on activities such as viewing code, editing code, testing, reading documentation, and managing the Git version control system. Each horizontal bar represents a pair and is split into two halves, one for each programmer in the pair.

ticipants were viewing and editing the same file at the same time in the code editor pane. This occurred during 23% of the total time, aggregating over all 4 sessions. The second most common role was ***developer/tester*** [7] where one person was in the code editor while their partner was in the testing pane, which occurred during 21% of the time. The third most common role was ***developer/assistant*** where one person would be coding while the other would be looking up documentation on the web to help out their partner, which occurred during 19% of the time. Other roles included both partners looking up documentation (9%) and both in the testing pane (7%).

Pair C was unique since the second partner (bottom half of Pair C in Figures 5 and 6) had significantly less programming experience. Thus, after completing the HTML, they served as a ***dedicated assistant*** the entire rest of the session and made no more code edits (no pink lines). They supported their partner's coding by testing, looking up documentation, and acting as a second pair of eyes on the code. Thus, CodePilot enabled collaboration even between people of widely varying skills.

DISCUSSION

Our exploratory study showed that novices can in fact use CodePilot to successfully work together in common collaborative roles on a simple yet realistic web programming project, unencumbered by many of the costs of starting up a collaborative software development session. We found three main themes from analyzing the qualitative data of user perceptions of CodePilot in post-study interviews:

Scaffolding for novices: Participants reported that as relative newcomers to programming, they found it hard to learn the nuances of Git at the same time as learning core programming concepts, which made them often neglect version control entirely in their own projects. They pointed out how CodePilot's integrated visual Git interface could be a useful instructional scaffold to gently introduce novices to version control concepts before seeing full-fledged command-line tools. They also said that experts would still prefer the flexibility of command-line tools, but that CodePilot can quickly get novices onboarded and working alongside their more experienced peers.

Situational awareness: Participants preferred using CodePilot over working side-by-side on separate computers using regular IDEs due to the extra situational awareness provided by the activity feed and concurrent editing in the CodePilot editor. Even without talking to one another, they could see what their partner was doing at each moment so that they did not work on something redundant or contradictory. In one instance, both partners of Pair B attempted to use JavaScript to parse the fetched data using different algorithms, but they saw the contradiction right away in the shared editor and talked to each other about how to proceed. Participants also appreciated not having to deal with unexpected Git merge conflicts that would have arisen if they had collaborated asynchronously.

Lower the barriers to impromptu collaboration: Even though some participants still preferred the power-user features of command-line tools, they appreciated how easy it was to jump into a new CodePilot session and to send a URL for their teammates to join as well. They mentioned how this tool could encourage more impromptu collaboration amongst developers, especially for prototyping, since it was so simple to get started using it. Being university students, our participants mainly expressed excitement about using CodePilot for class projects, hobby projects, and hackathons, but they also felt that it could be useful in their future jobs in the technology industry. Specifically, they mentioned that CodePilot could encourage coworkers to work together more due to lower barriers to collaboration, and it may also benefit system administrators and I.T. support personnel at software development organizations. Instead of wrangling employees to constantly install, upgrade, and manage multiple versions of disparate tools, I.T. staff can maintain a single up-to-date CodePilot web application that all developers use. Having a unified end-to-end environment also makes it easier for system administrators to debug configuration issues that stifle developers (e.g., "Why is my version control system acting so slow today?").

Although these initial results are encouraging for eliciting first impressions from novices and generating future design ideas, further studies are needed to more rigorously quantify CodePilot's effects on collaboration, productivity, and learning.

CONCLUSION AND FUTURE DIRECTIONS

As more people learn to code both in person and online, it is important to foster good collaborative software development habits. CodePilot takes one step toward this goal by unifying the collaborative development workflow into a single IDE and thus lowering the barrier to entry for novices. More broadly, as remote and distributed software development grow more prevalent in the coming years, it becomes ever more important to build in collaboration as a core design consideration in the next generation of IDEs. Our CodePilot prototype points toward a future where anyone can quickly jump into a software project and start making meaningful contributions.

In the future, we can extend CodePilot to cover additional phases of software development such as requirements gathering, project planning, design, and user testing. We also hope to support parallel prototyping and fork/branch-based models of development so that collaborators can easily explore alternatives without needing to work on only one version of the code. We can also investigate how to replay the recorded traces of collaborative development sessions to train novices offline. Even more broadly, CodePilot can serve as a research platform: If it were instrumented with fine-grained opt-in logging and deployed at scale, then software engineering researchers could mine that data and conduct controlled experiments to gain insights about how developers collaborate and learn.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grant NSF CRII IIS-1463864, by the Donald M. Barnard Scholarship, and by the University of Rochester GEAR Scholarship and University Research Award. Also, thanks to Dan Hassin for his valuable insights.

REFERENCES

1. Jacob T. Biehl, Mary Czerwinski, Greg Smith, and George G. Robertson. 2007. FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 1313–1322. DOI: http://dx.doi.org/10.1145/1240624.1240823

2. Yan Chen, Steve Oney, and Walter S. Lasecki. 2016. Towards Providing On-Demand Expert Support for Software Developers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3192–3203. DOI: http://dx.doi.org/10.1145/2858036.2858512

3. Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. 2012. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM, New York, NY, USA, 1277–1286. DOI: http://dx.doi.org/10.1145/2145204.2145396

4. Prasun Dewan, Puneet Agarwal, Gautam Shroff, and Rajesh Hegde. 2009. Distributed Side-by-side Programming. In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering (CHASE '09)*. IEEE Computer Society, Washington, DC, USA, 48–55. DOI: http://dx.doi.org/10.1109/CHASE.2009.5071410

5. Hongfei Fan, Chengzheng Sun, and Haifeng Shen. 2012. 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)*. ACM, New York, NY, USA, 107–116. DOI: http://dx.doi.org/10.1145/2389176.2389194

6. Max Goldman, Greg Little, and Robert C. Miller. 2011. Real-time Collaborative Coding in a Web IDE. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 155–164. DOI: http://dx.doi.org/10.1145/2047196.2047215

7. Max Goldman and Robert C. Miller. 2010. Test-driven Roles for Pair Programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '10)*. ACM, New York, NY, USA, 13–20. DOI: http://dx.doi.org/10.1145/1833310.1833313

8. Lile Hattori and Michele Lanza. 2010. Syde: A Tool for Collaborative Software Development. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE '10)*. ACM, New York, NY, USA, 235–238. DOI: http://dx.doi.org/10.1145/1810295.1810339

9. R. Hegde and P. Dewan. 2008. Connecting Programming Environments to Support Ad-Hoc Collaboration. In *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE '08)*. IEEE Computer Society, Washington, DC, USA, 178–187. DOI:http://dx.doi.org/10.1109/ASE.2008.28

10. James D. Herbsleb. 2007. Global Software Engineering: The Future of Socio-technical Coordination. In *2007 Future of Software Engineering (FOSE '07)*. IEEE Computer Society, Washington, DC, USA, 188–198. DOI:http://dx.doi.org/10.1109/FOSE.2007.11

11. Stephan Salinger, Christopher Oezbek, Karl Beecher, and Julia Schenk. 2010. Saros: An Eclipse Plug-in for Distributed Party Programming. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '10)*. ACM, New York, NY, USA, 48–55. DOI: http://dx.doi.org/10.1145/1833310.1833319

12. Patrick Thibodeau. 2013. India to overtake U.S. on number of developers by 2017. http://www.computerworld.com/article/2483690/it-careers/india-to-overtake-u-s--on-number-of-developers-by-2017.html. (2013). Accessed: 2016-09-13.

13. Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. 2000. Strengthening the Case for Pair Programming. *IEEE Software* 17, 4 (July 2000), 19–25. DOI:http://dx.doi.org/10.1109/52.854064

14. Laurie Williams, Charlie McDowell, Nachiappan Nagappan, Julian Fernald, and Linda Werner. 2003. Building Pair Programming Knowledge Through a Family of Experiments. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE '03)*. IEEE Computer Society, Washington, DC, USA, 143–. http://dl.acm.org/citation.cfm?id=942801.943642