Understanding Students' Needs for Better Collaborative Coding Tools

Kimberly Michelle Ying
University of Florida
Gainesville, FL 32611, USA
kimying@ufl.edu

Kristy Elizabeth Boyer
University of Florida
Gainesville, FL 32611, USA
keboyer@ufl.edu

ACM ISBN 978-1-4503-6819-3/20/04.
http://dx.doi.org/10.1145/3334480.3383068

Abstract

Collaborative coding offers many benefits to students, but there has been little research on evaluating the applications that students use to collaborate on code. In this preliminary work, we ask "are students' needs being met by existing applications for collaborative coding"? A survey was distributed to students and faculty of computer science to determine if students had experience collaborating on programming projects and identify what applications, if any, they used to facilitate their collaborations. Survey respondents were also asked about the strengths and weaknesses of the applications they used. From the 126 student responses and 23 faculty responses representing 31 unique institutions, over 50 applications were mentioned. We manually clustered the applications based on their affordances and used participant responses to identify opportunities for improvement. We found that many students are retrofitting non-coding applications for their programming projects as a workaround to facing the large learning curves that many collaborative coding tools require. Our findings suggest a need for more novice-friendly collaborative tools.

Author Keywords

Collaborative tools; computer science education; collaborative coding

Research Questions

1) What applications are students using to facilitate their collaborative coding endeavors?

2) What are the strengths and weaknesses of existing collaborative coding applications?

CCS Concepts

•Human-centered computing → Collaborative and social computing systems and tools;

Introduction

Collaborative coding has proven beneficial in both industry and education resulting in higher quality code in less time [1, 22], more confident programmers [22], and higher retention rates in computer science courses [15]. We define collaborative coding as two or more people working on a program. This involves all aspects of the collaboration including planning, sharing and organizing files, as well as editing code. The collaboration could be *synchronous*, with programmers working in real-time on the same file or even the same piece of code simultaneously, or *asynchronous*. Asynchronous collaboration is generally supported by the use of source or version control such as Git [8]. Online platforms such as GitHub host remote repositories of source code [9], allowing development teams to access and edit the source code, while tracking changes with Git.

Synchronous collaboration can happen in person or remotely. In-person collaborations often follow the *pair-programming* paradigm, where one person (the *driver*) has control of the keyboard and mouse and writes the code while the other (the *navigator*) assists in the overall process, providing suggestions and catching syntax and logic errors [22]. Co-located pair programming does not require any additional software beyond what a single programmer uses, but remote pair programming is commonly facilitated by screen-sharing applications and/or real-time editing software through which two or more programmers can work on the same source code and see the edits updated live.

Many applications have been created to facilitate collaborative coding (i.e., [13, 17]), but little research has been conducted to evaluate these applications [6], especially in the context of education and novice users. We speculate that many of these applications are designed with industry needs in mind and are tailored toward power users. This work, therefore, is focused on understanding students' collaborative coding methods and user needs. The overarching goal is to provide designers of collaborative coding tools, and researchers on collaborative coding, a better understanding of student needs. To begin this investigation, we surveyed students and faculty in academia to determine the current use of—and perspectives on—existing applications for collaborative coding. In this preliminary work, we identify common themes from the survey responses, organize the mentioned applications by type, and highlight the needs and opportunities revealed by students and faculty.

Related Work

Few studies have investigated collaborative coding applications as a whole to determine whether user needs are being met. Individual applications are commonly evaluated after development (i.e., [19, 20]), but research on the collaborative process and the multitude of tools that can be used is much less common. Fiala, Yee-King, and Grierson [7] compared features of existing online collaborative coding interfaces with a platform they created called CodeCircle, but they did not involve users in the evaluations. Bani-Salameh and Jeffrey conducted a literature review on groupware and collaborative tools, specifically discussing collaborative development environments (CDEs) and social development environments (SDEs) [3]. While they included descriptions and feature-sets of many tools, user feedback is not mentioned and it is unclear how the researchers chose the applications to review. Additionally, only two of the reviewed applications were targeted towards novice programmers.

There have also been applications specifically created for

Student Survey Questions

1) Have you ever collaborated (worked with other people) on a programming project?
yes, no (skip to last question)

2a) Have you used applications to collaborate in real-time, working on the same file or piece of code simultaneously?
yes, no
2b) Have you collaborated on code in person or remotely? (Select all that apply.)
in person, remotely

3) What applications have you used to collaborate?
free response

4) What have been the strengths and weaknesses of those applications?
free response

5) What do you feel are the most important features for collaborative coding applications?
free response

an educational context, but many times they do not include any user testing (i.e., [14, 21]). On the other hand some collaborative coding tools designed for students have resulted in positive user evaluations (i.e., [4, 5, 10, 11]), but these tools are not widely used. This may point to a larger problem of educational tools being created and researched, but not continuously supported once the research project ends. It is therefore necessary to understand how students are using commercial applications and whether these applications are meeting their needs.

This preliminary study differs from those mentioned because it crowdsources the applications to be evaluated instead of selecting applications based on the authors' own experience or web search. Students and faculty completed an online survey about collaborative coding and were asked (among other things) a free-response question on what applications they or their students use to collaborate on programming projects. This approach is beneficial in two ways: (1) it provides a more realistic view of what applications are being used by students and (2) it allows for a more holistic understanding of the collaboration process since respondents included applications they used for organization and communication, in addition to source-code editing. This approach is similar to previous work evaluating user needs and tools for practitioners collaborating on data [12]. In that work, Koesten, Kacprzak, Tennison, and Simperl crowdsource the applications to evaluate based on responses to a Twitter survey asking what tools and platforms practitioners use to collaborate on data.

Methods

Survey Recruitment and Distribution
Participants were recruited in two ways. First, contacts were compiled for 100 universities with computer science programs, based on a ranking of the best schools in the US [18]. To reduce confusion and limit unsolicited emails, only one contact from each university was collected from the institution's computer science department. Depending on the available email addresses listed for each program, we chose one that we felt was most appropriate for reading our recruitment email and (potentially) forwarding to students and faculty. These contacts were generally advising, admissions, and general info email addresses for the computer science (or related) department. Email contacts specifically for undergraduate programs were chosen over those for graduate programs, if the institution listed them separately. Second, the recruitment email was sent to the SIGCSE-members listserv [16], an opt-in mailing list managed by the Association for Computing Machinery (ACM) Special Interest Group for Computer Science Education (SIGCSE) for the discussion of computer science education as well as professional announcements. The SIGCSE community includes members from both US-based institutions and international institutions.

Survey Content and Flow
A survey was created to determine (1) whether computer science students were collaborating on programming projects, (2) what applications they were using to collaborate, and (3) perceptions of the aforementioned applications. The survey was administered online through Qualtrics. After consenting, participants were asked what university/institution they were affiliated with so that we could determine the distribution and reach of our recruitment emails. The student and faculty survey questions varied slightly in wording, but were identical in the content solicited (see left margin).

Data and Analysis

Of the 100 institutions contacted for survey distribution, 16 had at least one student or faculty survey response. The remaining 15 participating institutions we assume are from

Faculty Survey Questions

1) Have any of your courses required students to collaborate on code?
yes, no (skip to last question)

2) Have your students used applications to collaborate in real-time, working on the same file or piece of code simultaneously? This type of collaboration could be in-person or remote.
yes, no, I don't know

3) What applications have your students used to collaborate?
free response

4) What have been the strengths and weaknesses of those applications?
free response

5) What do you feel are the most important features for collaborative coding applications?
free response

the email recruitment sent to the SIGCSE-members mailing list. This analysis includes only survey responses of those that were 100% complete, requiring the respondent to click the submit button. In total, there were 23 faculty responses and 126 student responses, resulting in over 50 unique applications mentioned.

Most student respondents (119/126) reported having experience collaborating on a programming project, and 57 of these reported having used real-time applications to collaborate. Similarly, most faculty respondents (18/23) reported requiring students to collaborate on code in the past, of which six reported that their students have used applications to collaborate in real-time. Of those with collaborative coding experience, 26 students had experience collaborating in person, eight had experience collaborating remotely, and 85 had experience with both in-person and remote collaborations.

Overview of Applications
Table 1 shows the range of application types mentioned with specific examples from the survey responses. Application types included version control, repository hosting, integrated development environment, source code editor, real-time collaboration, video conferencing, task management, chat messaging, desktop sharing, information sharing, file transfer, and DevOps. We see a lot of overlap in the types of collaborative tools used by data practitioners as reported by Koesten et al, namely *online tools*, *wiki-based platforms*, and *versioning tools* [12]. Due to space constraints, some applications are not included in the table. Additionally, many of the applications have multiple features and could be classified under more than one type but are listed here under the type that is most relevant (according to respondent elaborations as well as application descriptions from their respective homepages).

	Applications
Type	*Mentioned Examples*
Version Control	Git, Sourcetree, Subversion
Repository Hosting / File Sharing	GitHub, Google Drive, Bitbucket, Unity Teams
Integrated Development Environment (IDE)	Eclipse, IntelliJ, CLion
Source Code Editor	Atom, Vim, Sublime
Real-Time Collaboration	Visual Studio Live Share, repl.it, Codeshare, Google Docs
Video Conferencing / Screen Sharing	WebEx, Teams, Google Hangouts
Organization / Task Management	Trello, Jira, Excel
Chat Messaging	Slack, GroupMe, Discord
Remote Access / Desktop Sharing	Team Viewer, tmux
Information Sharing	Wiki, Piazza
File Transfer	email, PuTTY
DevOps	GitLab, CircleCI

Table 1: Application types derived from the over 50 collaboration applications mentioned by students and faculty.

Participating Institutions
Amherst College
Berea College
Boston University
Case Western Reserve Univ.
Christopher Newport Univ.
Clovis Community College
Dalhousie University
Duke University
Franklin University
Ithaca College
Johns Hopkins University
Metro. State Univ. of Denver
Michigan State University
Northern Kentucky University
Rochester Inst. Of Tech.
Roger Williams University
Saint Bonaventure University
Stanford University
State University of Piaui
Texas A&M University
Univ. of Calif., Santa Barbara
University of Central Florida
University of Florida
University of Freiburg
University of Michigan
University of Oregon
University of Tennessee
University of Texas at Austin
University of Washington
Virginia Tech
Whitman College

Application	Strengths	Weaknesses
Git, GitHub	supports revision control; facilitates issue tracking and working independently; integrates well with other software; easy to use; very robust for project management: organization, overhead, accountability, peer review; built-in backups; multiple features can be worked on at the same time	steep learning curve for beginners; lacks ability to discuss code that is not in a pull request; relies on members constantly pushing changes; not real-time; hard to set up; merging issues; easy to ruin the whole project; difficult to measure individual contribution
Google Drive, Google Docs	easy to learn; okay for sharing files/code; better for code snippets and explaining concepts; great for quickly setting up real-time collaborative documents for planning projects; immediate changes; works everywhere; easy to use	less usable for coding; editing certain file types not supported; hard to manage versions; not fully-featured; no syntax support; difficult to revert other people's changes; not integrated with GitHub
Visual Studio, VS Code, VS Live Share (LS)	great for large-scale projects; automatic syncing and merging of files on commit; overwrite and merge customization; easy to use; intuitive; good syntax/language support; LS provides shared editor and terminal	incorrect merges are difficult to repair; need more visualized application; hard to set up initially; laggy and almost unusable when the network connection is not good; no native remote screen sharing; LS guests have trouble saving changes to host's files

Table 2: Application strengths and weaknesses from survey responses. Some clauses were edited for succinctness and clarity.

We grouped strengths and weaknesses mentioned for each application so that all strengths for a particular application were listed together and all weaknesses for a particular application were listed together. Table 2 reports strengths and weaknesses expressed about the applications by students and faculty members for the most frequently mentioned applications.

Versioning Tools

The most popularly reported tools were those for version control, with 81 out of 119 students mentioning Git and/or GitHub specifically. Other version control platforms like Bitbucket were also mentioned by some students, and others mentioned their IDEs or code-editors like Visual Studio Code having built-in version control. One of the most common weaknesses mentioned for Git and GitHub was a large learning curve. While many students said Git or GitHub was easy to use, about 20% explicitly stated that it was difficult to learn and only one of the 81 students wrote that it was easy to learn. Students also expressed concern about inadvertently "ruin[ing] the whole project."

Another common weakness reported for versioning tools was merging issues. One student reported wanting "A way to see what other people are currently working on before they push their code to master." During our literature review, we found a group developing a tool called *Collaboration Over GitHub* that addresses this need by providing real-time information to collaborative developers about any arising conflicts [2].

Student anxieties about using versioning tools may have led some students to use more user-friendly or familiar applications that were not necessarily designed for code collaboration, such as Google Drive and Google Docs. One respondent mentioned they use email to transfer code. Over 10% of the students mentioned using Google Drive or Google Docs specifically. Two students did not mention currently using the Google suite for their collaborative programming projects, but used the products to describe their ideal development tools, writing "I wish there was a Google Drive but for development," and "Something like a Google Document would be much better, since both parties can edit code at the same time."

One student did not find value in collaborating in real-time, reporting that it "seems like more of a hassle over just working at the same time and using Git." This may indicate a personal preference and comfort using version control tools, but it could also mean that current real-time tools are too difficult to set up.

Desired Support for Pair Programming

Some students and faculty mentioned their (students') collaborations were mostly done through pair programming. Traditionally, this does not require the use of any additional tools beyond what a single programmer uses, but some students revealed a desire for additional support. One student wrote, "Collaborating in person is great, but it would be nice to both be able to use a keyboard." Two other students also liked pair programming saying it was "familiar and reliable" and that it was good because they were "both looking at the same thing," but that it was difficult for the non-driver to point things out. Our previous research confirmed similar sentiments, in which some students criticized pair programming for its reduction in hands-on learning [23]. Even in cases of in-person collaboration, there is potential for technologies to create a more efficient workflow.

Conclusion and Future Work

In this paper, we used student and faculty survey responses to identify applications students use to collaborate on programming projects as well as their corresponding strengths and weaknesses. From this preliminary analysis, we conclude that many students' needs are not being met by current tools and therefore more novice-friendly collaborative tools are needed. Although applications have been created for an educational collaborative context, these tools are not widely used and so we urge those creating commercial applications to be more deliberate in considering the novice user. To gain a deeper understanding of students' needs, future work will include conducting focus groups and user studies on common collaborative tools. Since steep learning curves were one of the most pervasive application weaknesses, we plan to observe students with no prior experience using the tools.

Limitations

While there were 31 unique institutions represented in the survey responses, it is important to note that some institutions have a much higher response count. The results from this analysis may be biased towards the more represented populations. The rankings from U.S. News & World Report (used to compile the list of 100 institutions) is based on master's programs. Institutions without graduate programs as well as those that are not among the highly ranked should also be recruited for participation.

Acknowledgements

We would like to thank the members of the LearnDialogue Group for their help and encouragement. This material is based upon work supported by the National Science Foundation under grant CNS-1622438. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s)and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, and S. Z. Sarwar. 2010. Agile Software Development: Impact on Productivity and Quality. In *2010 IEEE International Conference on Management of Innovation Technology*. 287–291. DOI: http://dx.doi.org/10.1109/ICMIT.2010.5492703

[2] Ritu Arora, Sanjay Goel, and Ravi Kant Mittal. 2017. Supporting Collaborative Software Development over GitHub. *Software: Practice and Experience* 47, 10 (2017), 1393–1416.

[3] Hani Bani-Salameh and Clinton Jeffery. 2014. Collaborative and Social Development Environments: A Literature Review. *International Journal of Computer Applications in Technology* 49, 2 (2014), 89–103.

[4] Kristy Elizabeth Boyer, August A. Dwight, R. Taylor Fondren, Mladen A. Vouk, and James C. Lester. 2008. A Development Environment for Distributed Synchronous Collaborative Programming. *SIGCSE Bull.* 40, 3 (June 2008), 158–162. DOI: http://dx.doi.org/10.1145/1597849.1384315

[5] Davor Čubranić and Margaret Anne D. Storey. 2005. Collaboration Support for Novice Team Programming. In *Proceedings of the 2005 International ACM SIGGROUP Conference on Supporting Group Work (GROUP '05)*. Association for Computing Machinery, New York, NY, USA, 136–139. DOI: http://dx.doi.org/10.1145/1099203.1099229

[6] Bernardo José da Silva Estácio and Rafael Prikladnicki. 2015. Distributed Pair Programming: A Systematic Literature Review. *Information and Software Technology* 63 (2015), 1–10.

[7] Jakub Fiala, Matthew Yee-King, and Mick Grierson. 2016. Collaborative Coding Interfaces on the Web. In *Proceedings of the 2016 International Conference on Live Interfaces*. 49–58.

[8] Git. Retrieved December 25, 2019 from https://git-scm.com/.

[9] GitHub. Retrieved December 25, 2019 from https://github.com/.

[10] 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)*. Association for Computing Machinery, New York, NY, USA, 155–164. DOI: http://dx.doi.org/10.1145/2047196.2047215

[11] Timothy J. Hickey, John Langton, and Richard Alterman. 2005. Enhancing CS Programming Lab Courses using Collaborative Editors. *Journal of Computing Sciences in Colleges* 20, 3 (2005), 157–167.

[12] Laura Koesten, Emilia Kacprzak, Jeni Tennison, and Elena Simperl. 2019. Collaborative Practices with Structured Data: Do Tools Support What Users Need?. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article Paper 100, 14 pages. DOI: http://dx.doi.org/10.1145/3290605.3300330

[13] Aditya Kurniawan, Christine Soesanto, and Joe Erik Carla Wijaya. 2015. CodeR: Real-time Code Editor Application for Collaborative Programming. *Procedia Computer Science* 59 (2015), 510–519.

[14] Andrés Moreno, Niko Myller, and Erkki Sutinen. 2004. JeCo, a Collaborative Learning Tool for Programming. In *2004 IEEE Symposium on Visual Languages - Human Centric Computing*. 261–263. DOI: http://dx.doi.org/10.1109/VLHCC.2004.33

[15] Nachiappan Nagappan, Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. 2003. Improving the CS1 Experience with Pair Programming. *SIGCSE Bull.* 35, 1 (Jan. 2003), 359–362. DOI: http://dx.doi.org/10.1145/792548.612006

[16] Special Interest Group on Computer Science Education. 2019. Mailing Lists. Retrieved December 20, 2019 from https://sigcse.org/sigcse/membership/mailing-lists.html.

[17] Michael Reeves and Jihan Zhu. 2004. Moomba - A Collaborative Environment for Supporting Distributed Extreme Programming in Global Software Development. In *International Conference on Extreme Programming and Agile Processes in Software Engineering*. Springer, 38–50.

[18] U.S. News & World Report. Best Computer Science Schools. Retrieved November 30, 2019 from https://www.usnews.com/best-graduate-schools/top-science-schools/computer-science-rankings.

[19] 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)*. Association for Computing Machinery, New York, NY, USA, 48–55. DOI: http://dx.doi.org/10.1145/1833310.1833319

[20] Till Schümmer and Stephan Lukosch. 2009. Understanding Tools and Practices for Distributed Pair Programming. *Journal of Universal Computer Science* 15, 16 (2009).

[21] Jason Vandeventer and Benjamin Barbour. 2012. CodeWave: A Real-Time, Collaborative IDE for Enhanced Learning in Computer Science. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*. ACM, 75–80.

[22] Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. 2000. Strengthening the Case for Pair Programming. *IEEE Software* 17, 4 (2000), 19–25.

[23] Kimberly Michelle Ying, Lydia G. Pezzullo, Mohona Ahmed, Kassandra Crompton, Jeremiah Blanchard, and Kristy Elizabeth Boyer. 2019. In Their Own Words: Gender Differences in Student Perceptions of Pair Programming. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, 1053–1059.