

# Jimbo: A Collaborative IDE with Live Preview

Soroush Ghorashi  
School of EECS  
Oregon State University  
Corvallis, OR, USA 97331  
ghorashs@oregonstate.edu

Carlos Jensen  
School of EECS  
Oregon State University  
Corvallis, OR, USA 97331  
cjensen@eecs.orst.edu

## ABSTRACT

Team collaboration plays a key role in the success of any multi-user activity. Software engineering is a highly collaborative activity, where multiple developers and designers work together to solve a common problem. Meaningful and effective designer-developer collaboration improves the user experience, which can improve the chances of success for the project. Learning to program is another activity that can be implemented in a more collaborative way, students can learn in an active style by working with others. The growth of online classes, from small structured seminars to massive open online courses (MOOCs), and the isolation and impoverished learning experience some students report in these, points to an urgent need for tools that support remote pair programming in a distributed educational setting.

In this paper, we describe Jimbo, a collaborative integrated development environment (IDE) that we believe is beneficial and effective in both aforementioned activities. Jimbo integrates many features that support better collaboration and communication between designers and developers, to bridge communication gaps and develop mutual understanding. These novel features can improve today's CS education by bringing students closer to each other and their instructors as well as training them to collaborate which is consistent with current practices in software engineering.

## ACM Classification Keywords

D.2.6 [Programming Environments]: Integrated Environments;  
D.2.2 [Software Engineering]: Design Tools and Techniques—  
*Distributed/Internet based software engineering tools and techniques*;

## Keywords

Jimbo; IDE; collaboration; communication; user awareness; live preview; designer-developer collaboration; collaborative learning; distance learning; pair programming; web development

## 1. INTRODUCTION

The development of software systems is a collaborative process, where team members work together to solve a problem by producing quality code. The designer-developer relationship at the heart of many of these collaborations is the force that moves a software project toward success. Unfortunately, in current software development practices, designers have no direct

engagement with developers in the development process, although the products performance depends on both. If we want to improve this relationship, and encourage better software products, we need to build development tools that improve the collaboration and work-flow for designers and developers.

One of the most popular and effective collaboration methods used in CS education is pair programming, which has been shown to be a very beneficial technique for teaching and engaging students with programming and new computing topics. The need for tools that support remote pair programming is becoming pressing with the growing popularity of massive open online courses (MOOC). While employing pair programming in a colocated classroom setting is relatively straightforward, there is a dearth of good options for distributed classroom settings. As students struggle to master concepts and build confidence in their skills, a tight code-artifact feedback loop/mechanism that allows students to verify that a change had the intended result is important.

We have built an IDE, called *Jimbo*, to facilitate the involvement of designers in the development process as first-class citizens. Our focus is on the development of web applications, which often require extensive interactions between designers and developers. In this paper, we briefly describe our tool, its main design goals, and then describe two major collaboration scenarios in which Jimbo can be effective and useful.

## 2. JIMBO

Jimbo is an IDE that enables users to more easily collaborate around a common project. We have tried to make the user interface easy to learn and memorable, but have also considered external consistency with other popular IDEs.

Jimbo is a web-based tool, which means that users only need a standard web browser and the setup time is zero. Sarma provides a comprehensive classification of collaborative tools for software development [1]; Jimbo could be considered a seamless tool at the top level as it provides many novel features to automate the development workflow and minimize user effort. In the following sections we describe these features briefly.

### 2.1 Synchronous and Asynchronous Collaboration.

Currently, distributed collaborative software development largely revolves around working in parallel on separate copies of the code, and integrating the resulting efforts using a source code version control system. Though this is an effective strategy, there is a lack of real collaboration, as developers are largely working independently, only coordinating their efforts when synching their code. The number of defects in code tends to rise as the amount of parallel work increases [2] and developers sometimes avoid this kind of development to avoid having to resolve conflicts [3].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CHASE'16, May 16, 2016, Austin, Texas, USA.

Copyright 2010 ACM 1-58113-000-0/00/0010 ...\$15.00.

DOI: <http://dx.doi.org/10.1145/12345.67890>

The most important feature of Jimbo is that it allows developers to collaborate on code in real-time without worrying about conflicts. Jimbo deploys an operational transformation (OT) algorithm to achieve this. OT is a technique that provides eventual consistency between multiple users working on the same code without retries, errors, or data being overwritten.

## 2.2 Communications

The integration of communication features into the IDE helps collaborators discuss and resolve issues without losing focus on the code. We recommend the following different but equally effective means of communication:

1. **Audio chat.** This provides virtual presence and makes it easier to coordinate and collaborate. This type of communication allows for quicker resolution.
2. **Text chat.** A text-based chat or instant messaging is a complimentary system for audio chat and is a must, as coders share code snippets and links to resources.
3. **Inline discussions.** Inline discussions are text-based semi-synchronous way of communication, in which users can associate a short discussion with an artifact such as a portion of code or a design document.

## 2.3 Live Preview

Live programming is a technique where programmers re-execute a program continuously while editing [4]. Live preview is a variation of this technique that refreshes the output immediately upon a change to the code, and it best fits UI-heavy application development such as websites. This technique provides an immediate connection between the code and the output for developers so they can see the effects of changes to their code. This feature leads to fewer iteration of the code, which means faster coding.

Currently web programming requires developers to write their code in an editor and then open it in a standard browser to see the results. Any changes they make to the code requires them to save the file, go to their browser and refresh the page to check the effect of changes made. When this is done collaboratively, the files edited have to be pushed to a server, and potential conflicts have to be managed, all of which is slow and effort intensive. Jimbo provides a real-time preview of the code being developed, which streamlines the development process by removing redundant refreshes on the browser.

## 2.4 User Awareness

Another fundamental requirement to supporting collaboration is awareness. Dourish and Bellotti define awareness as “an understanding of the activities of others that provides a context for your own activity” [5]. The main purpose of an awareness system is to provide information about development activities to help coordinate tasks. In Jimbo, we follow the “continuous coordination” model introduced by van der Hoek et al. [6]. The primary responsibility of such a system is to notify developers of events relevant to them, such as code changes, comments to discussion threads, user presence, etc.

Jimbo has a powerful channel based notification system [7] using push notifications. These notifications are persistent and stored on the server for future retrieval. To prevent cognitive overload, developers can request to receive notification about a specific portion of the code, a feature we call “code watch”. Once someone puts a watch on a portion of code, Jimbo pushes notifications regarding any changes to that section of code. This feature allows developers to keep track of sections of code their work depends on, or code that they have some ownership over.

## 3. COLLABORATION SCENARIOS

We have identified two main populations that can benefit from a tool like what we have described. We have therefore created two versions of Jimbo: One educational (Figure 1) and a professional (Figure 2). First, we discuss how Jimbo can help students and instructors, then we explain how designer-developer collaboration can be improved using Jimbo.

### 3.1 Educational Settings

One of the most popular models of collaboration is pair programming. The impact of pair programming in CS education has been confirmed by numerous studies in both lab and classroom settings [8, 9, 10, 11]. Pair programming requires two programmers to work together on the same computer, but the trend toward geographically distributed teams make long-distance adaptations necessary. A modified model in which collaborators are not collocated is called remote pair programming (RPP), and it has been shown that RPP provides the same benefits as collocated pair programming [12]. While it is important to facilitate both synchronous and asynchronous collaboration – providing the most flexibility for students, synchronous collaboration is perhaps the most important as it makes it possible for students to engage in RPP at any time and from any location without worrying about syntactic consistency.

The integration of communication features into the IDE could help students discuss without losing focus on the code. Most communication tools (Skype, IRC, email, etc.) are not directly integrated with the IDE, which can lead to a disconnect between code and discussion, or simply wasted effort by frequent context switching. A text-based chat system is a must, as students share code snippets and links to resources. An audio chat system allows students to talk freely similar to effective face-to-face meetings.

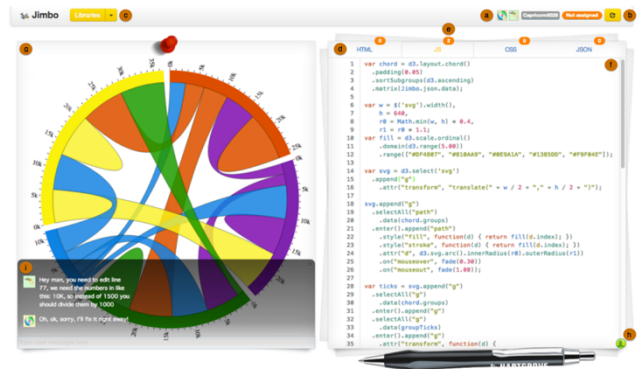


Figure 1. Jimbo educational edition.

Another popular practice is “live coding,” where an instructor writes code in front of students, exposing their thought process. While this is more effective than just showing the final solution to students, it is a passive learning technique. Gaspar and Langevin [13] successfully used a student-led version of this approach to engage students in active learning and expose students’ thought process to the instructor for more in-depth feedback. To better support student-led live coding and engage all students during a class session, having a tight code-artifact feedback loop that allows students to verify that a change had the intended result is important. One advantage of combining a live preview component with pair programming is that live preview supports a distributed and thus scalable way of engaging in live coding. This would allow all students to benefit from this practice during short class periods, whereas before only a handful of them would get the chance to practice this technique.

Figure 1 shows the main view of the Jimbo educational edition. To simplify the programming process for students who are new to the web development topics, there are exactly one editor for each of the HTML, CSS and JavaScript codes and they are located on the right side of the view. Live preview, if enabled, can be found on the left side of the view. Text chat panel is located on the bottom left and students can join an audio call by simply pressing a button which is positioned on the top right.

### 3.2 Professional Settings

Software teams usually have more than one developer and designer collaborating. Instead of isolating designers from the development, we suggest including them in the implementation process, working with the developers while these work with the code. Many designers have basic development skills, most notably HTML, CSS and JavaScript for prototyping. Having synchronous coding integrated into IDE tools enables designers to make modifications directly, or work with the developers and direct their development efforts without going through a long set of steps to deliver their ideas to the engineering team effectively. This can facilitate a more design oriented process in which designers create the skeleton of the product and then developers fill in the gaps to bring the design to life. This way, designers guard their designs themselves and developers are forced into the habit of getting more immediate and meaningful feedback.

The integration of communication features into the IDE could help developers and designers discuss and resolve the issues they may face without losing focus on the code. Audio chat allows for quicker resolution in case of minor design misunderstanding between developers and designers that if stay unresolved, it sometimes will lead to major defects in the product and delay in the process and in worst case failure of the project. Unlike transitory audio discussions, inline discussions are tied to people and artifacts and tend to be permanent, having their own context that will not be lost over time.

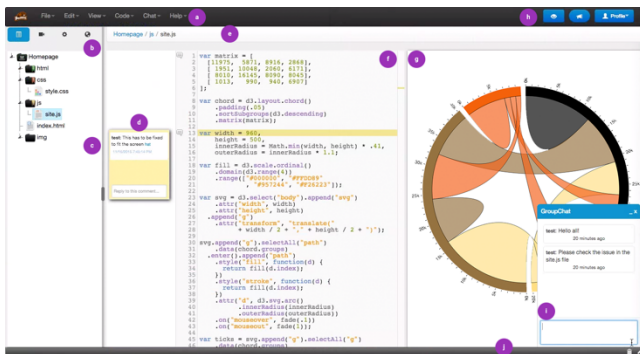


Figure 2. Jimbo professional edition.

In an environment where designers and developers are collaborating with each other, live preview is a powerful tool. Using live preview, designers can instantly see what changes developers are making and provide feedback and direction, warning the coding team if they are deviating from the design vision. It also streamlines the flow of communication within the team, and makes it easier for developers to ask designers for input or help when changes have to be made.

Awareness is required to coordinate teams, but can be distracting if it interrupts or requires too much attention from members. It is not an easy issue to address in a collaborative environment, as we juggle the need for asynchronous editing for some developers, and the need for real-time preview of the resulting code for others. A

good awareness system in such environment will provide useful answers to the questions that a user may have, for example: Who made what change in the code? Who should I contact if I have a question about this part of design/code? Who is currently available? What are they doing now?

Figure 2 shows the main view of the Jimbo professional edition. The UI in this version is similar to popular IDEs such as Eclipse, with the project file tree on the left, and code editor expanding to the right. The text chat panel can be found on the bottom right of the screen, and inline discussions are located on the right edge of the code editor. If a user wishes to have their live preview enabled, they can find its panel on the right side of the view. Finally, notification center, audio conference call and user profile options are located on the top right of the screen.

### 4. RELATED WORK

Researches have studies collaborative coding and built many tools following the WYSIWIS (what you see is what I see) metaphor that supports collaboration, including: GROVE [14], ShrEdit [15], DistEdit [16] and Flesce [17]. One of the key challenges for these tools is making sure that the code is always error-free, as any collaborator may run and test the code at any point in time, regardless of what the others are doing. Collabode [18] has been developed to address this issue by only sharing modifications that result in an error-free code, acting as an automatic safety buffer.

A number of tools have been developed to provide real-time awareness of code changes to facilitate coordination and emerging conflicts in collaborative environments. FASTDash [19], ProjectWatcher [20], Palantir [21] and Syde [22] are all examples of this kind of tools. Crystal [23] proactively watches the code and precisely identifies and reports conflicts. There are also growing number of plug-in services being developed for existing IDE's, trying to add more awareness and collaboration features to familiar tools such as Eclipse JAZZ project [24].

Some recent live programming systems include Superglue, Flogo II, Lively Wiki and Brackets [25, 26, 27, 28]. In a real world example, Khan academy recently successfully deployed a basic version of a live programming environment for an online programming course for students with no experience [29]. Researchers are trying to enhance live programming environments by focusing on debugging [30].

### 5. CONCLUSION AND FUTURE WORK

In this paper, we introduced Jimbo, a web based collaborative IDE with live preview built mainly for web development. Jimbo integrates a number of techniques and features, including synchronous coding, user awareness, live preview and various communication channels that previously existed in separate tools. We briefly discussed two main user scenarios – the educational and professional – that we think Jimbo can be most beneficial in.

We have developed multiple editions of Jimbo to support collaboration in educational settings where students thrive on collaboration in problem solving, and also professional settings where people in different roles try to solve a common problem collaboratively. Currently we are conducting multiple user studies following these two settings to investigate whether our tool Jimbo; truly helps users to solve real-life problems effectively, and we look forward to reporting the results of these studies.

## 6. REFERENCES

- [1] Sarma, A. A survey of collaborative tools in software development. Technical Report UCI-ISR-05-3, University of California, Irvine, Institute for Software Research, 2005.
- [2] Perry, D. Siy, H. and Votta, L. Parallel changes in large-scale software development: an observational case study. *ACM TOSEM*, 10:308–337, July 2001.
- [3] Grinter, R. Using a configuration management tool to coordinate software development. In *CoOCS*, pages 168–177, Milpitas, CA, USA, Aug. 1995.
- [4] Tanimoto, S. VIVA: A visual language for image processing. *J. Vis. Languages Computing*, 127–139, June 1990.
- [5] Dourish, P. and Bellotti, V. Awareness and coordination in shared workspaces. In *ACM Conference on Computer-Supported Cooperative Work*, pages 107–114, Monterey, California, USA, 1992.
- [6] Van der Hoek, A. and et al. Continuous coordination: A new paradigm for collaborative software engineering tools. In *Proceedings of Workshop on WoDiSEE*, Scotland, 2004.
- [7] OMG. CORBACos: Notification Service Specification v1.0.1. 2002.
- [8] Horn, E. M., Collier, W. G., Oxford, J. A., Bond Jr, C. F., & Dansereau, D. F. Individual differences in dyadic cooperative learning. In *Journal of Educational Psychology*, 1998.
- [9] Williams, L. and Kessler, R. R. “Experimenting with Industry’s ‘Pair-Programming’ Model in the Computer Science,” *Journal on SW Engineering Education*, 2000.
- [10] O'Donnell, A. M. and Dansereu, D. F. “Scripted Cooperation in Student Dyads: A Method for Analyzing and Enhancing Academic Learning and Performance,” in R. Hartz-Lazarowitz and N. Miller (Eds.) *Interactions in Cooperative Groups: The Theoretical Anatomy of Group Learning*, pages 120–141, London: Cambridge University Press, 1992.
- [11] McDowell, C., Werner, L., Bullock, H., & Fernald, J. (2002, February). The effects of pair programming on performance in an introductory programming course. In *ACM SIGCSE Bulletin* (Vol. 34, No. 1, pp. 38–42). ACM.
- [12] Berliner, B. CVS ii: Parallelizing software development. In *USENIX Winter 1990 Technical Conference*, pages 341–352, 1990.
- [13] Gaspar, A. Langevin, S. Active learning in introductory programming courses through student-led “live coding” and test-driven pair programming, *EISTA 2007, Education and Information Systems, Technologies and Applications*, July 12–15, Orlando, FL.
- [14] Ellis, C.A. Gibbs, S.J. and Rein, G.L. Design and use of a group editor. In *Engineering for Human Computer Interaction*, pages 13–25, Amsterdam, 1990.
- [15] McGuffin, L.J. and Olson, G. Shredit: A shared electronic workspace. Technical Report 45, Cognitive Science and Machine Intelligence Laboratory, Tech reports: 45, University of Michigan, Ann Arbor, 1992.
- [16] Knister, M.J. and Prakash, A. Distedit: a distributed toolkit for supporting multiple group editors. In *Proceedings of the 1990 ACM conference on Computer-supported cooperative work*, Los Angeles, CA, 1990.
- [17] Carzaniga, A. Rosenblum, D.S. and Wolf, A.L. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 2001. *ACM Trans. Comp. Sys.*
- [18] Goldman, M. 2011. Role-based interfaces for collaborative software development. In *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology (UIST '11 Adjunct)*. ACM, New York, NY, USA, 23–26.
- [19] Biehl, J.T. Czerwinski, M. Smith, G. and Robertson, G.G. FASTDash: A visual dashboard for fostering awareness in software teams. In *CHI*, pages 1313–1322, San Jose, CA, USA, Apr. 2007.
- [20] Schneider, K.A. Gutwin, C. Penner, R. and Paquette, D. Mining a Software Developer's Local Interaction History. *MSR 2004*, Edinburgh, 2004.
- [21] Al-Ani, B. Trainer, E. Ripley, R. Sarma, A. Hoek, A. and Redmiles, D. Continuous coordination within the context of cooperative and human aspects of software engineering. In *CHASE*, pages 1–4, Leipzig, Germany, May 2008.
- [22] Hattori, L. and Lanza, M. Syde: A tool for collaborative software development. In *ICSE Tool Demo*, pages 235–238, Cape Town, South Africa, May 2010.
- [23] Brun, Y. Holmes, R. Ernst, M. and Notkin, D. 2011. Proactive detection of collaboration conflicts. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE '11)*. ACM, New York, NY, USA, 168–178.
- [24] Cheng, L. Hupfer, S. Ross, S. and Patterson, J. Jazzing up eclipse with collaborative tools. In *18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications / Eclipse Technology Exchange Workshop*, pages 102–103, Anaheim, CA, 2003.
- [25] Hancock, C. M. Real-time programming and the big ideas of computational literacy. PhD thesis, Massachusetts Institute of Technology, 2003.
- [26] McDirmid, S. Living it up with a live programming language. In *Proc. of OOPSLA Onward!*, pages 623–638, October 2007.
- [27] Krahn, R. Ingalls, D. Hirschfeld, R. Lincke, J. and Palacz, K. 2009. Lively Wiki a development environment for creating and sharing active web content. In *Proceedings of the 5th International Symposium on Wiki and Open Collaboration (WikiSym'09)*. ACM, New York, NY, USA.
- [28] Adobe Brackets, <http://brackets.io>.
- [29] Resig, J. Redefining the introduction to computer science. <http://ejohn.org/blog/introducing-khan-cs/>, 2012.
- [30] McDirmid, S. (2013, October). Usable live programming. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software* (pp. 53–62). ACM.