The impact of dynamic locking on collaborative programming

Adrian Shatte, Jason Holdsworth, Ickjai Lee
College of Business, Law and Governance (Information Technology)
Division of Tropical Environments & Societies
James Cook University, Cairns, Australia
Email: Adrian.Shatte@my.jcu.edu.au,
Jason.Holdsworth@jcu.edu.au, Ickjai.Lee@jcu.edu.au

Abstract

The parallel strategy of collaborative writing is commonly used by academia and industry. However, the nature of this approach may not scale to other genres of collaboration including collaborative programming. In this paper, we present a web-based prototype that supports parallel programming with the use of dynamic locking, and evaluate its scalability and system usability. Our results suggest there are benefits to our approach, and we discuss the implications and future directions for research.

Keywords
Collaboration, Parallel Writing, Pair Programming, Dynamic Locking, Web Technologies.

1. INTRODUCTION

The importance of collaboration has become increasingly prevalent in many fields and industries, including education (Beldarrain 2006), software development (Cockburn and Williams 2000), academia (Northway et al. 2001), and music production (Martinez-Borda et al. 2013). While there are many aspects to collaboration, Collaborative Writing (CW) is a frequently used strategy (especially in academia and industry) within the scope of the respective genre-specific documentation (Nelson 2000; Wheeler and McDonald 2000) to obtain a broader scope of writing problems and adjust writing accordingly (Yeh 2014).

According to Lowry, there are five strategies used in CW: *single-author*, *sequential single*, *reactive*, *mixed mode*, and *parallel* writing (Lowry et al. 2004). *Single-author writing* is a strategy in which a single author writes the document on behalf of the group. *Sequential single writing* occurs when the writing task is divided between the members of a group, and each group member writes their assigned portion before passing the document on to the next group member. *Reactive writing* is a different strategy in which all group members work on the writing task synchronously, reacting to and adjusting contributions as they are made. The *mixed mode* strategy denotes any CW task that combines at least two of the aforementioned strategies. *Parallel writing* is a strategy where the writing task is divided between the group members such that each member works on their assigned portion of the task at the same time. There are two subcategories of parallel writing strategy: (1) *horizontal division*, whereby each user writes a single section of the document, and (2) *stratified division*, where each member of the group is assigned a role in contributing to the final document, such as author, editor, facilitator, and group leader.

Recent research has determined that students predominantly adhere to the parallel writing strategy (Gimenez and Thondhlana 2012). However, there are limitations to the parallel writing strategy, including an increased possibility of poor group communication, information overload, and stylistic differences that may arise when students are working on separate sections of a document in isolation (Gimenez and Thondhlana 2012). Moreover, CW is not widely explored in the context of collaborative programming such as pair programming (Cockburn and Williams 2000).

Programming is a field in which collaborative technologies unlock significant benefits for users. Traditionally, programmers collaborated through the use of a shared computer and Integrated Development Environment (IDE), or through a shared repository of source code (Goldman et al. 2011). Recently, tools were developed that allow programmers to collaborate in real-time through an IDE (Salinger et al. 2010) or a web browser (Goldman et al. 2011; Hwang et al. 2012; Langton et al. 2004). We consider collaborative code editing as a specialised form of *parallel writing*. In addition to *horizontal* and *stratified* division, we propose a *dynamic* divisioning where each participant has an equal ability to perform various levels of changes to code as different participants move between intricate and deeply related tasks.

Locking is a simple way of maintaining data consistency in CW by permitting only a single user to edit the shared document or a subsection of the document at any one time making it read-only for other collaborative users until editing is completed (Menascé and Nakanishi 1982). In this locking scheme, a lock must be requested

before updating a critical region and locking is a must (Sun and Sosič 1999). This form of locking is extreme and can inhibit collaboration, prompting the investigation of dynamic locking schemes to enable flexibility in collaborative editing systems (Lautamäki et al. 2012). Dynamic locking allows for fine-grained control of text-editing in context-specific scenarios and ensures that data integrity is maintained (Sun and Sosič 1999). Dynamic locking is a key feature supported by collaborative applications that benefits *parallel* collaborative writing for programming.

In this paper, we describe the development of a browser-based collaborative HTML editor with dynamic locking built on web technologies and supporting libraries. The major aim of our prototype is to explore and demonstrate the ability of such an application to improve parallel writing and define a programming-centric class of CW. Existing research identifies dynamic locking as an important feature that assists in maintaining data integrity in collaborative editing scenarios (Sun and Sosič 1999). However, the majority of applications with support for locking are implemented using edit-based algorithms (such as Operational Transformation (Sun and Sosič 1999)) that are notoriously difficult to implement. As such, a secondary goal of our research is to achieve reliable and effective locking using simpler technologies to ensure ease of deployment alongside existing single-user applications.

We describe the results of our evaluation including a case study designed to determine the usability and acceptance of our application. Scalability is another important variable in CW. We conduct benchmarking studies to determine whether the scalability of the application is affected by the number of collaborators and the nature of collaboration.

The remainder of the paper is structured as follows: Section 2 provides a brief overview of related work in the field of collaborative software development; Section 3 describes the prototyping of a web-based collaborative programming environment to support *parallel* collaborative writing; Section 4 reports on the methods used to evaluate the prototype and results obtained from the studies; and Section 5 discusses the conclusions and speculates on future work.

2. RELATED WORK

Rudimentary approaches to collaboration that use technology include tools such as Concurrent Versions Systems (CVS), screen sharing, email, and instant messaging programs (Cheng et al. 2003). We believe that these approaches are inherently distracting because they require the user to minimise the development environment while focusing on other applications (Cheng et al. 2003). Additionally, CVS tools are limited in that they do not normally provide real-time collaboration.

As the number of collaborators on a project increases, so too does the importance of clear communication between those collaborators (Galegher et al. 2014). Non-technical solutions to establish a steady flow of communication include team-building exercises, regular meetings, agile software development principles, and support from management (Cheng et al. 2003). However, meetings and exercises can also hinder the progress of development, due to its separation from the development environment. For this reason, approaches that integrate and promote collaboration within the development environment used by collaborative programmers were investigated (Hupfer et al. 2004).

Other approaches integrate some of these tools directly into an IDE. Research has shown that implementing collaborative tools directly into IDEs has several benefits, including reduced friction in the development process, improved context, and immediate traceability between collaborative artefacts and code artefacts (Cheng et al. 2003). A noteworthy example of a real-time collaborative editing plugin for the Eclipse IDE is Saros (Salinger et al. 2010).

Recently, web-based IDEs or programming environments were developed that allow collaborators to work on software development within a web browser. There are many benefits to using browser-based programming environments to software-based IDEs, including automatic distribution, installation, and updating of applications (O'Reilly 2007), as well as independence from the development environment such that users working on different platforms can access the same content through a common interface (Goldman et al. 2011).

Table 1: Groupware systems for collaborative programming.

Name	Features	Reference
Collabode	Synchronous collaboration, Java pair programming and execution (minus GUI development).	(Goldman et al. 2011)

Group Homework Tool (GHT)	Synchronous code editor, assignment definitions and resources, chat, shared whiteboard.	(Langton et al. 2004)
CoRed	Browser-based collaborative code editor for Java applications.	(Lautamäki et al. 2012)
WPASC	Supports writing source code, browsing source code, compiling, executing and debugging, and peer feedback.	(Hwang et al. 2012)

Table 1 above summarises notable web-based collaborative programming applications from the literature, citing the major features of each system. These systems provide important features that support programming, such as syntax highlighting, automatic indentation, debugging, and execution of code. Additionally, the systems allow the shared code to be synchronised to all other collaborators. It should be noted that CoRed is the only one of these applications that provides support for locking, however the authors did not report on the benefits or usability of this feature. As such, our research aims to bridge this gap by exploring the impact of dynamic locking on collaborative programming.

Next, we discuss our prototype web-based collaborative programming with specific focus on its major features and how these features intend to support and improve parallel programming for users.

3. PROTOTYPING OF A COLLABORATIVE PROGRAMMING ENVIRONMENT

First we describe the research methods that guide both the implementation and evaluation stages of our project. Next we describe the web technologies that were used in the development of our collaborative programming prototype. Finally, we provide an overview of the prototype with specific focus on its major features and how these features intend to support and improve parallel programming for users.

3.1. Research methods

Two main research methods are utilised in our ongoing research into dynamic locking for collaborative programming. First, our implementation is guided by the *Design Science* approach (Von et al. 2004). As our research is ongoing, the implementation and results reported in this paper reflect only the first iteration of prototyping, and the results are intended to guide further developments. The second research method guiding our study is *Mashup Development* (Yu et al. 2008), which describes web applications generated by combining content or application functionality from disparate Web sources. All of the tools used in our application development were selected for their interoperability and ease of deployment to the web.

3.2. Tools for development of prototype

The three primary tools utilised when developing the prototype for this study include the Node.js framework (Tilkov and Vinoski 2010), the DS algorithm (Fraser 2009) for consistency and convergence of shared text, and Ace editor to provide HTML syntax support[1]. These tools were utilised for their ease of development and compatibility. Each tool is briefly described in the following subsections.

3.2.1. Node.js

Node.js[2] is a server-side JavaScript environment based on Google's V8 runtime engine (Tilkov and Vinoski 2010). Node.js is commonly used to develop real-time applications due to its asynchronous nature. There are also a large number of modules that are written primarily in JavaScript that can be used by Node.js applications to extend the functionality. In addition, users are encouraged to upload their own modules to a central repository for public use, which means that commonly used libraries are readily available and well tested.

3.2.2. Differential Synchronisation

For collaborative editing features, we decided to use Neil Fraser's Guaranteed Delivery implementation of Differential Synchronisation (DS) (Fraser 2009). DS is a robust algorithm for document synchronisation with open source implementations written for common programming languages, including JavaScript (Lautamäki et al. 2012), which means that it is compatible with the other tools used in our prototype. We selected DS for our prototype due to its ease of implementation, high responsiveness, and high concurrency rate (Fraser 2009).

[1] http://ace.c9.io/
[2] http://nodejs.org

Finally, DS is designed so that it can be appended to existing applications, thus introducing real-time collaboration to previously static systems (Fraser 2009). That is, existing single-user applications can be provided with powerful collaborative features and adapted to suit a desired type of collaboration in an efficient manner.

3.2.3. Ace editor

Ace editor is an embeddable code editor that is written in JavaScript. Thus, it can be easily embedded into existing applications that run on web technologies, and is compatible with Node.js. Ace editor supports many features synonymous with programming IDEs, including syntax highlighting, automatic indentation, line wrapping, custom highlighting, and syntax checking. Such features are not typically required in regular collaborative text-editing applications and relate only to the semantics of programming languages. We selected Ace editor due to its compatibility with the other technologies used in our prototype. The client-side of the application is written in JavaScript, so it is simple to embed an Ace editor into the page. Further, the DS algorithm supports synchronisation of plaintext, which means that Ace editor can be synchronised without any modifications.

3.3. Collaborative HTML Editing Prototype

Utilising the technologies described in Section 3.2., we developed a collaborative HTML editing application aimed at novice programmers. The application runs on a client-server architecture, with each client able to connect through a web browser by navigating to the URL of the server. Figure 1 displays the client-side user interface of the application.

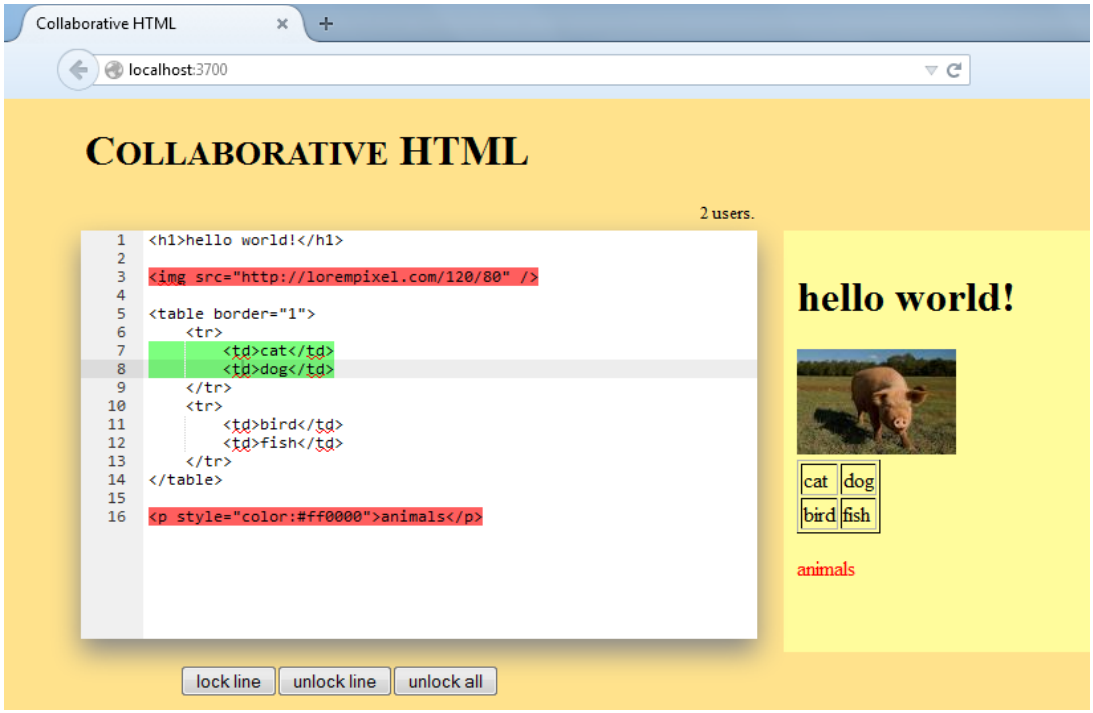

Figure 1: Screenshot of our collaborative HTML editing prototype.

As discussed in Section 1 of this paper, we define the nature of teaching HTML programming concepts to novice programmers as a "fine-grained" level of collaborative writing. As such, the traditional form of *parallel* writing whereby each user completes a separate section in isolation cannot be strictly adhered to. Instead, we propose the concept of *dynamic division*, where each participant has an equal ability to perform various levels of changes to code as different participants move between the deeply related tasks.

Our prototype includes a collaborative code editing window (see Figure 1), in which users are able to contribute HTML code to a shared document. This code editing window is synchronised on a regular basis (every 400ms) to ensure that the content remains consistent for all collaborators. The code editor provides many usability features that are intended to improve the nature of programming in the browser, including line numbers, syntax formatting, and automatic tab indentation.

To the right side of the code editor is a second window which automatically renders the HTML code as it is updated. This provides collaborators with instant feedback on how the current code will appear in a web

browser. For example, Figure 1 displays several HTML elements including a header, an image, a table, and a paragraph.

To support *parallel* collaborative writing concepts, we implemented a locking feature to allow users to claim exclusive ownership over any section of the shared document at any time. Similar to traditional parallel writing strategies, users can divide the task at hand into smaller tasks to be worked on individually. The benefit of a locking scheme in a shared document is that users are able to view the progress of collaborators in real-time which can increase awareness, facilitate communication, and reduce the effects of stylistic differences that may exist between users.

In our prototype, users can request to lock the current line of code by clicking the *lock line* button below the code editor. This sends a request to the server and if no conflicting locks already exist, the user is granted a lock. Similarly, a user can undo a lock or all locks that they own by clicking the *unlock line* or *unlock all* buttons respectively. To improve the usability of the application in terms of locking, we decided to provide a visual representation of the locks to the user. Locks owned by the user are highlighted in green, whereas locks owned by other users are highlighted in red.

If a user attempts to edit a lock that is owned by another user, a warning message appears to indicate that the user does not have permission to edit that section of the document. In contrast, if a user attempts to edit a lock that they own, no warning will appear and the edits are synchronised between collaborators. Subsequently any user can edit a section of the document that is not locked (indicated by a lack of green or red highlighting). If one user was to lock all of the shared content, a new user could easily begin a new line and lock it, creating an equal playing field in which no single user can control all of the shared text.

4. EVALUATION

Next we discuss the experimental results we obtained through benchmarking and a small case study. These experiments aimed to determine both the scalability of our system and the usability and acceptance of the application.

4.1. Determining the Scalability of Our Implementation

As our application was developed to support collaborative editing among multiple users, it is important to determine its ability to provide consistency and efficiency for an increasing number of collaborators. While previous studies have determined that DS is scalable (Fraser 2009), our implementation has combined several additional technologies that may affect efficiency. Thus, we simulated an increasing number of users collaborating and conducted benchmarking on the application to verify that the application maintained scalability as the number of collaborators increases.

A custom script was developed in JavaScript which ran in the browser alongside our prototype. The script provided random input to the application (retrieved from a dictionary of common, basic HTML expressions), simulating the behaviour of typical users in a novice collaborative programming environment. There were three conditions: (1) equal edits, in which the number of additions and deletions made to the shared text were equal; (2) more additions than deletions, in which there was a greater chance for edits to add text to the document; and (3) more deletions than additions, in which there was a greater chance for edits to delete text from the shared document. We selected these three conditions to identify any differences that may exist between different types of editing tasks.

Each condition was simulated with an increasing number of simulated users. While smaller collaborative programming scenarios such as pair programming require support for a trivial number of users, there may exist other contexts where there is a greater requirement for large groups of users, e.g. an undergraduate programming course with interactive lectures, Massive Open Online Courses, and so on. For this reason, we simulated both small and large numbers of users. There were five conditions tested in terms of number of simulated users: one user, two users, ten users, fifty users, and one-hundred users.

Two timestamps were recorded within the algorithm, one at the beginning of a synchronisation loop, and one at the end of a synchronisation loop. These timestamps were recorded in a log file for further analysis. The difference between these times was calculated and recorded. The process was repeated one-hundred times for each experimental condition and an average was calculated.

The results of the benchmarking experiments are displayed in Figure 2 and Table 2. Note that the data was cleaned and averaged to obtain an average synchronisation score for each condition.

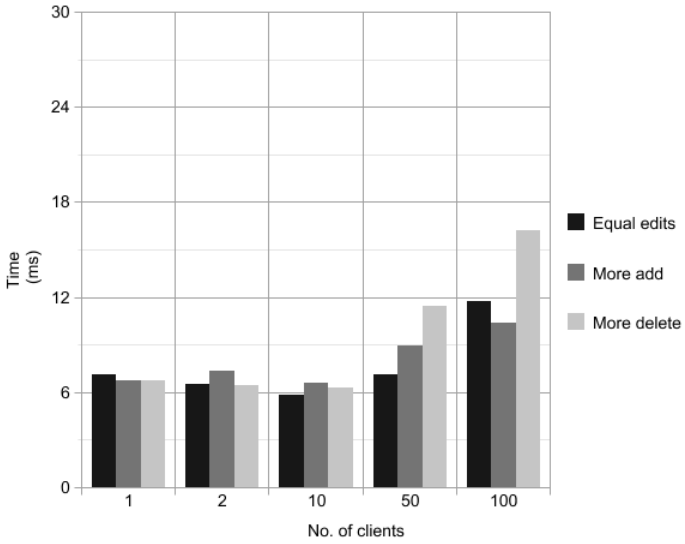

Figure 2: Mean synchronisation times captured through benchmarking.

Table 2: Average synchronisation time in all three conditions for increasing users.

No. of users	Group 1: Equal edits	Group 2: More add (50% increase)	Group 3: More delete (50% increase)
1	7.16	6.71	6.75
2	6.55	6.47	7.33
10	5.86	6.32	6.61
50	7.12	11.47	8.94
100	11.76	16.18	10.40

Based on these results, it is evident that the average synchronisation time only suffers slightly as the number of collaborators increases. Importantly, the mean difference between synchronisation times for 100 users compared with 1 user was less than 10 milliseconds, indicating minimal inconvenience to larger groups of users.

4.2. System Usability Scale

As the development of our prototype is still in progress, it is important that we gain information on the usability of our system at an early stage to guide further development. To determine the usability and acceptance of our application for supporting parallel programming in its current state, we conducted a case study with a convenience sample of four users.

Four participants were recruited for this study to test our collaborative HTML editing prototype with dynamic locking. Each participant was determined to have at least basic knowledge of HTML editing due to previous study in an Information Technology degree that included a web programming component. Participants were instructed to work together to complete a HTML programming task and create a simple web page containing information about animals. The participants were instructed to divide the task between all group members by using the lock feature to claim ownership of any section that they were editing, and to unlock those sections after editing was completed. No further instructions on how to use the locking features of the system were provided.

Upon completion of the task, each participant answered the System Usability Survey (SUS) with specific focus on the dynamic locking feature. The SUS is a flexible and technology-agnostic survey used to quickly assess the usability of a product or service, and has been determined to provide valuable and robust insights into a broad range of user interfaces (Bangor et al. 2008). It can also be used on small sample sizes with reliable results that can be used to differentiate between usable and unusable systems (U.S. Dept of Health & Human Services 2014). The SUS produces a single score which represents the overall quality of the system. The results of the SUS for the four participants in our study are displayed in Table 3.

Table 3: SUS scores for our study.

	Participant 1	Participant 2	Participant 3	Participant 4
SUS Score	88.5	92.5	72.5	77.5

According to the recommendations made for analysing SUS scores (Bangor et al. 2008), our application on

average scores from *Good* to *Excellent*. This means that users were overall satisfied with the interface of the system, but there is still a room for improvement in further iterations of development. The implications of these results are positive, such that future improvements to the application should follow similar design principles. However, it should be noted that these scores are not diagnostic - they do not indicate which aspects of the system perform best - thus the results should be used only as an indicator of the prototype's ease of use, and additional feedback should be considered alongside this data such as the qualitative comments presented in Section 4.3.

4.3. Qualitative Feedback

We requested participants of the study to provide informal feedback on the locking aspect of the application. There were three main themes prevalent in the comments provided by participants.

Firstly, two participants commented that it was not ideal to lock a single line of the shared document at a time. One participant mentioned that it is possible to unlock all lines owned by a user with a single button press, so there should be a method in place to lock multiple lines at the same time if desired.

Second, one participant commented on the colours used to identify which locks are owned by the current user (e.g. green) and which locks are owned by other users (e.g. red). This participant suggested that while it was useful to easily determine which locks you could edit, it would be better to have a different colour for all users in the collaboration scenario, as well as other identifying features such as the name of the user or a symbol to assist potential users with colour-blindness in identifying locks.

Finally, one participant commented on the potential usefulness of such an application in other aspects of their programming workflow. The participant mentioned working on documentation that required input from several experts in differing skill areas, e.g. management, quality assurance, and software engineering. Currently, these users are required to work on multiple copies of the same document in isolation, and collate all of the different sections at a later time. In contrast, the technologies behind our prototype, including DS (Fraser, 2009), can work with any type of plaintext, thus allowing all users to work on a single copy of a text document, and lock their own section to ensure integrity.

5. CONCLUSION AND FUTURE WORK

CW has been an active research topic since its first groupware started in early 1970s. As the technology advanced, CW has been widely adopted in many applications. Scalability of CW has become one of the important features as the number of users involved in a CW grows. Dynamic locking is a flexible locking system that improves the rigidness of traditional locking system. In this paper, we studied the impact of dynamic locking on CW. First we presented the current state of our research on supporting and improving *parallel* collaborative programming with collaborative-editing software. We discussed the prototyping and evaluation of a web-based collaborative HTML editor aimed at novice programmers, with support for dynamic locking of shared content to claim ownership of subsections of a larger document.

The results of our preliminary evaluations indicate the need for deeper analysis in future studies. Firstly, our benchmarking experiments demonstrate that the system performs robustly up to one-hundred concurrent collaborators. However, there may exist scenarios in which a greater number of users are required to collaborate on a shared document. We have not accounted for such scenarios in this study, so future work may investigate the scalability of our system to deal with such scenarios. Fraser (Fraser 2009) proposes several methods which can assist when scaling the DS algorithm that may be useful in future research. It may also be possible that there is a human or social-interaction limit effecting the quality of collaboration when the number of users increases, which cannot be accounted for by a simulation. Therefore, a future research goal is to determine the scalability of our system with real users.

In addition, the results of our user case studies indicated that the participants were satisfied with the locking aspect of the system to support collaboration. While this is a positive sign, there are deficiencies in our system that are evident from the comments provided by users. For example, it was stated that the locking system is not flexible enough for all scenarios (e.g. locking only a single line at a time). Future development can improve on this by implementing a more flexible locking scheme similar to other systems (Sun and Sosič 1999). In addition, our qualitative case study investigated only the usability of the system, so future extensions of our research should investigate other areas including usefulness, ease of use, and impact on the quality of the output.

Further, we mentioned that one user complained that the colours used to indicate locks were not informative enough. One of the major features missing from our application is support for user attribution (Fraser 2009). User attribution refers to a system coupling every contribution with its author, and presenting this information in such a way that other collaborators can easily determine who contributed what content. This feature has been

shown to increase awareness and improve the overall collaboration experience for users (Kraut et al. 2003). Future development will include support for user attribution to improve collaboration.

To conclude, our research in this area so far has yielded promising results that indicate that collaborative software with dynamic locking functionality can improve CW, especially parallel programming. We intend to build upon this research in the future to investigate the benefits of dynamic locking in other areas of collaboration. We also plan to incorporate additional collaborative features such as user attribution and chat.

7. REFERENCES

Bangor, A., Kortum, P. T., and Miller, J. T. 2008. "An empirical evaluation of the system usability scale," *Intl. Journal of Human–Computer* Taylor & Francis.

Beldarrain, Y. 2006. "Distance education trends: Integrating new technologies to foster student interaction and collaboration," *The American journal of distance education* (27:2), Taylor & Francis, pp. 139–153.

Cheng, L.-T., de Souza, C. R., Hupfer, S., Patterson, J., and Ross, S. 2003. "Building collaboration into IDEs," *Queueing Systems. Theory and Applications* (1:9), ACM, p. 40.

Cockburn, A., and Williams, L. 2000. "The costs and benefits of pair programming," *Extreme programming examined*. Addison--Wesley Publishing Co., Reading, MA, USA, pp. 223–247.

Fraser, N. 2009. "Differential synchronization," in *Proceedings of the 9th ACM Symposium on Document Engineering*, pp. 13–20.

Galegher, J., Kraut, R., and Egido, C. 2014. *Intellectual Teamwork: Social and Technological Foundations of Cooperative Work*, Taylor & Francis.

Gimenez, J., and Thondhlana, J. 2012. "Collaborative writing in engineering: Perspectives from research and implications for undergraduate education," *European Journal of Engineering Education* (37:5), Taylor & Francis, pp. 471–487.

Goldman, M., Little, G., and Miller, R. C. 2011. "Collabode: collaborative coding in the browser," in *Proceedings of the 4th international workshop on Cooperative and human aspects of software engineering*, pp. 65–68.

Hupfer, S., Cheng, L.-T., Ross, S., and Patterson, J. 2004. "Introducing Collaboration into an Application Development Environment," in *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW '04, New York, NY, USA, pp. 21–24.

Hwang, W.-Y., Shadiev, R., Wang, C.-Y., and Huang, Z.-H. 2012. "A pilot study of cooperative programming learning behavior and its relationship with students' learning performance," *Computers & education* (58:4), Elsevier, pp. 1267–1281.

Kraut, R. E., Fussell, S. R., and Siegel, J. 2003. "Visual Information as a Conversational Resource in Collaborative Physical Tasks," *Human Computer Interactaction* (18:1), Hillsdale, NJ, USA: L. Erlbaum Associates Inc., pp. 13–49.

Langton, J. T., Hickey, T. J., and Alterman, R. 2004. "Integrating tools and resources: a case study in building educational groupware for collaborative programming," *Journal of Computing Sciences in Colleges* (19:5), Consortium for Computing Sciences in Colleges, pp. 140–153.

Lautamäki, J., Nieminen, A., Koskinen, J., Aho, T., Mikkonen, T., and Englund, M. 2012. "CoRED: browser-based Collaborative Real-time Editor for Java web applications," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pp. 1307–1316.

Lowry, P. B., Curtis, A., and Lowry, M. R. 2004. "Building a Taxonomy and Nomenclature of Collaborative Writing to Improve Interdisciplinary Research and Practice," *Journal of Business Communication* (41:1), pp. 66–99.

Martinez-Borda, R., Lacasa, P., and Ruth, M. 2013. "Classrooms or Rock Stages? Learning Music through Collaboration," *Breaking the Mold of Education: Innovative and Successful Practices for Student Engagement, Empowerment, and Motivation* (4), p. 125.

Menascé, D. A., and Nakanishi, T. 1982. "Optimistic versus pessimistic concurrency control mechanisms in database management systems," *Information systems* (7:1), Elsevier, pp. 13–27.

Nelson, S. 2000. “Teaching collaborative writing and peer review techniques to engineering and technology undergraduates,” in *Frontiers in Education Conference, 2000. FIE 2000. 30th Annual*, (Vol. 2), pp. S2B/1–S2B/5.

Northway, R., Parker, M., and Roberts, E. 2001. “Collaboration in research,” *Nurse Researcher* (9:2), pp. 75–83.

O’Reilly, T. 2007. “What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software,” *Communications and Strategies* papers.ssrn.com.

Salinger, S., Oezbek, C., Beecher, K., and Schenk, J. 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, New York, NY, USA, pp. 48–55.

Sun, C., and Sosič, R. 1999. “Optimal Locking Integrated with Operational Transformation in Distributed Real-time Group Editors,” in *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC ’99, New York, NY, USA, pp. 43–52.

Tilkov, S., and Vinoski, S. 2010. “Node. js: Using JavaScript to build high-performance network programs,” *IEEE Internet Computing* doi.ieeecomputersociety.org.

U.S. Dept of Health & Human Services. 2014. “System Usability Scale (SUS).” Retrieved 9 August, 2014, from http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Von, R. H., March, S. T., Park, J., and Ram, S. 2004. “Design science in information systems research,” *The Mississippi quarterly* Springer.

Wheeler, E., and McDonald, R. L. 2000. “Writing in Engineering Courses,” *Journal of Engineering Education* (89:4), Blackwell Publishing Ltd, pp. 481–486.

Yeh, H. C. 2014. “Exploring how collaborative dialogues facilitate synchronous collaborative writing,” llt.msu.edu.

Yu, J., Benatallah, B., Casati, F., and Daniel, F. 2008. “Understanding Mashup Development,” *IEEE Internet Computing* (12:5) ieeexplore.ieee.org, pp. 44–52.

INTELLECTUAL PROPERTY

COPYRIGHT