

A Development Environment for Distributed Synchronous Collaborative Programming

Kristy Elizabeth
Boyer*

August A.
Dwight

R. Taylor
Fondren

Mladen A.
Vouk

James C.
Lester

Department of Computer Science
North Carolina State University
*keboyer@ncsu.edu

ABSTRACT

While collaborative approaches in the classroom have been shown to be highly beneficial for students of computer science, obstacles inherent in today's academic environment often prevent collocated collaborative approaches from being implemented. One solution to the collocation problem may lie with tools that facilitate distributed collaboration. This paper presents RIPPLE (Remote Interactive Pair Programming and Learning Environment), a development environment for distributed synchronous collaborative programming. RIPPLE is an open source software tool. Initial user tests demonstrate positive responses from students, and the potential for long term learning, motivation, and retention benefits is significant. In addition to its benefits for students, RIPPLE is a tool for computing education researchers who wish to collect data on collaborative programming.

Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education]:
Computer Science Education

General Terms: Human Factors, Languages

Keywords

Distributed collaboration, distributed pair programming, distance learning, distributed tutoring, programming environments, laboratory/active learning

1. INTRODUCTION

Computer science educators and researchers constantly strive to increase the quality of teaching and learning in computing courses. The motivations behind such efforts stem from a variety of goals including addressing pipeline issues, creating a more capable workforce, increasing diversity in computing, and advancing the state of knowledge in our field. Collaboration in various forms has shown promise in helping to progress toward these goals. However,

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 to publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ITiCSE '08, June 30–July 2, 2008, Madrid, Spain.

Copyright 2008 ACM 978-1-60558-115-6/08/06...\$5.00.

collaboration is often hindered in practice by a variety of issues such as facilities, scheduling, and geographic separation. One solution to the collocation constraint may be found in tools that facilitate distributed collaboration. Because programming is ubiquitous in computing education at most institutions starting with CS1 and continuing through a senior project course [1], distributed programming collaboration offers fertile ground for deployment of tools that support remote teamwork.

In this paper we present RIPPLE (Remote Interactive Pair Programming and Learning Environment), a freely available, open source software tool that enables two programmers to work remotely on the same Java program at the same time. RIPPLE was designed for use in educational settings to facilitate various forms of collaborative programming problem solving including distributed pair programming and distributed one-on-one tutoring. Since it was developed with educational research in mind, data collection through RIPPLE is straightforward. This paper describes and motivates three supported usage scenarios: distributed pair programming, distributed tutoring, and data collection for research on collaborative programming.

2. RELATED WORK

While other environments to support teamwork have been used successfully for distributed collaborative programming [2, 4, 13], RIPPLE differentiates itself by being freely available and open source with integrated support for data collection. In addition, RIPPLE's purpose-driven design has resulted in lower bandwidth requirements than general purpose shared desktop or remote meeting applications.

2.1 Distributed Pair Programming

Pair programming is a software engineering approach in which two programmers work side by side at the same computer [24]. One partner acts as the *driver*, who actively writes code and has control of the keyboard and mouse. The other partner acts as the *navigator*, who helps plan as well as identify and prevent any syntactic or strategic deficiencies. The collaborators may exchange roles at any time during the pair programming session, or not at all.

Pair programming has been shown to be an effective pedagogical approach in teaching courses such as introductory computer science [18, 19], undergraduate software engineering [25], and graduate object-oriented software development [3]. There are indications that the approach may also improve retention of women in computer science [23], and it even shows promise with middle school students

[22]. In concert with results that demonstrate the benefits of collaborative learning in CS1 [5] and theoretical classes such as algorithms [15], there is substantial evidence supporting the benefits of collaborative programming learning. However, obstacles such as limited facilities, geographic separation, and scheduling often present challenges to collocated pair programming.

Enabling students to collaborate from different locations may provide appreciable benefits compared with individual work. While remote work comes with its own challenges [20], recent results suggest these challenges may not be insurmountable when it comes to distributed pair programming [3]. The importance of allowing students to work on their programming projects remotely has been recognized by systems for distance learning (e.g., [9]). Software support of distributed pair programming is being explored in other systems (e.g., [13]); in fact, RIPPLE directly extends an existing system for remote pair programming [14].

Table 1 displays an excerpt of an actual textual dialogue accompanying a distributed pair programming session using RIPPLE. Each textual dialogue message is preceded by the user name and time of day associated with that message. In this excerpt, the collaborators are able to discuss changes made to the code in real time because actions taken by the driver are transmitted to the remote user as those actions occur.

Table 1

Sample Pair Programming Excerpt	
User 1 (13:13:11):	Wanna start with this one?
User2 (13:13:24):	ok
User1 (13:16:30):	What the...you see what's wrong here?
User2 (13:17:05):	Yeah, I was just about to say something. You kinda forgot to put that inside a method.
User1 (13:17:10):	Duh, thx
User2 (13:20:05):	Seems good. But did we decide we should initialize that in the constructor?

This excerpt illustrates a key benefit of pair programming: the ability of one programmer to quickly catch mistakes made by the other programmer. Such immediate feedback can improve efficiency and may reduce student frustration [25]. It is also plausible that during pair programming sessions, students may be prompted by their partners to explain the reasoning behind programming actions. Such a prompt to explain oneself may lead to higher learning gains, a widely studied phenomenon known as the *self-explanation effect* [8].

2.2 Distributed Tutoring

One-on-one tutoring has been shown to be highly effective for student learning [6]. RIPPLE allows a student and a tutor to work together remotely because the tutor can see student programming actions in real time as well as carry on tutorial dialogue through the textual dialogue interface. While tutoring through textual dialogue cannot rival the richness of face-to-face interaction, many studies

have shown students still benefit greatly from tutoring in typed dialogue (e.g., [10]).

Table 2 illustrates an excerpt of a distributed tutoring session using RIPPLE. The tutor was able to see all student programming actions, and the pair had a running dialogue about the student's actions, questions, and misconceptions.

Table 2

Sample Tutoring Excerpt	
Student (17:30:00):	Okay. Then will I need to redeclare digits = new int[5]?
Tutor (17:30:10):	No, we've already declared it. We can just use it.
Student (17:31:11):	I'm sorry. I'm still stuck on that last loop we wrote because I feel like that's where my digits should come from.
Tutor (17:31:16):	That's fine.
Tutor (17:31:22):	That is where the digits are coming from.
Tutor (17:31:29):	They're stored in the digits array.
Tutor (17:31:39):	Does that make sense?
Student (17:32:32):	Sort of, but then how do I add them together? Is it something like digits[1] + digits[2] + digits[3] + ... or is there a different way to approach this since we're using an array?
Tutor (17:32:50):	That's exactly what we're going to be doing, but we can do it with a loop instead of typing it out one at a time.

2.3 Research on Collaboration

Researchers can use RIPPLE to generate detailed logs of distributed collaborative programming sessions.¹ These logs can then be mined to investigate a variety of research questions. For example, the study of human tutoring can give researchers insight into the processes involved in teaching and learning [12, 17]. Researchers have used detailed logs of human tutoring to better understand how students learn subjects such as computer programming [16] and basic human anatomy [10]. Facilitating such research is an important function of RIPPLE.

3. RIPPLE

RIPPLE is a plug-in for the popular Eclipse integrated development environment [21]. When two users work together remotely through RIPPLE, each executes a separate local installation of Eclipse with the RIPPLE plug-in. From each programmer's perspective, there are two panes: the program development pane and the textual dialogue pane (Figure 1).

¹ Researchers are responsible for obtaining permission from their institutional review board prior to collecting any student data.

3.1 Synchronized Programming View

RIPPLE extends the architecture implemented in Sangam [14], an Eclipse plug-in for distributed pair programming. Compilation and execution of code, as well as generation of console messages, are performed directly by Eclipse.

The collaborators' views are synchronized over several dimensions. When one user undertakes a supported action, that action generates an event that is transmitted to the remote user. Synchronized actions include file system manipulation (e.g., creating and deleting files), editor operations (e.g., typing and highlighting) and program launch (execution).

To allow robust operation even with a less than perfect network connection, RIPPLE performs integrity checks regularly to assure the editor contents are identical for both (distributed) users. To preserve the low bandwidth requirement, RIPPLE does not transmit more complex features such as mouse pointer location. In addition, to allow maximum flexibility for the individual preferences of programmers, RIPPLE does not force identical configurations of the Eclipse development environment. Each user may tailor toolbars, views, compiler settings, and many other options with no negative impact on RIPPLE's functionality.

The event-driven behavior of RIPPLE requires that language-specific messages be transmitted between users. Because of this requirement, RIPPLE currently only supports Java programming. Java is very common in computing curricula in the United States [11], but extension to support other programming languages is a promising direction for future work.

3.2 Textual Dialogue

RIPPLE features textual dialogue support through an instant-message-style chat program. In standard instant messaging (such as the kind provided with many popular online chat applications), both users may construct messages simultaneously and send them when desired. RIPPLE supports this format of dialogue as its default setting. As users of online textual chat programs know, it is common for one user to send a message while the other user is actively constructing another message. Thus, messages sometimes appear out of sequence (with respect to conversational thread) in the dialogue history. Table 3 illustrates a conversation in which one user sent a message starting a different conversational thread while the other user was constructing a message pertaining to the previous thread. In this example, Rohit was constructing message 3 while Sarah was constructing message 2 – and Sarah sent her message slightly before Rohit sent his. Clearly Rohit's second message was meant to be a continuation of his first one and in no way a response to Sarah's message which was sent moments earlier.

For two human users in most natural collaborative settings, this message sequencing problem is easily overcome because the collaborators have watched the conversation unfold in real time. However, in certain research settings, the ambiguity can confound analyses of the dialogue. There are times when a structured form of dialogue is required, i.e., when strict turn-taking must be imposed on the collaborators. The textual dialogue component of RIPPLE supports enforced turn-taking in dialogue. In this structured mode, only one user may type a message at a time. This setting guarantees that messages appear in the dialogue history in the order in which they were constructed.

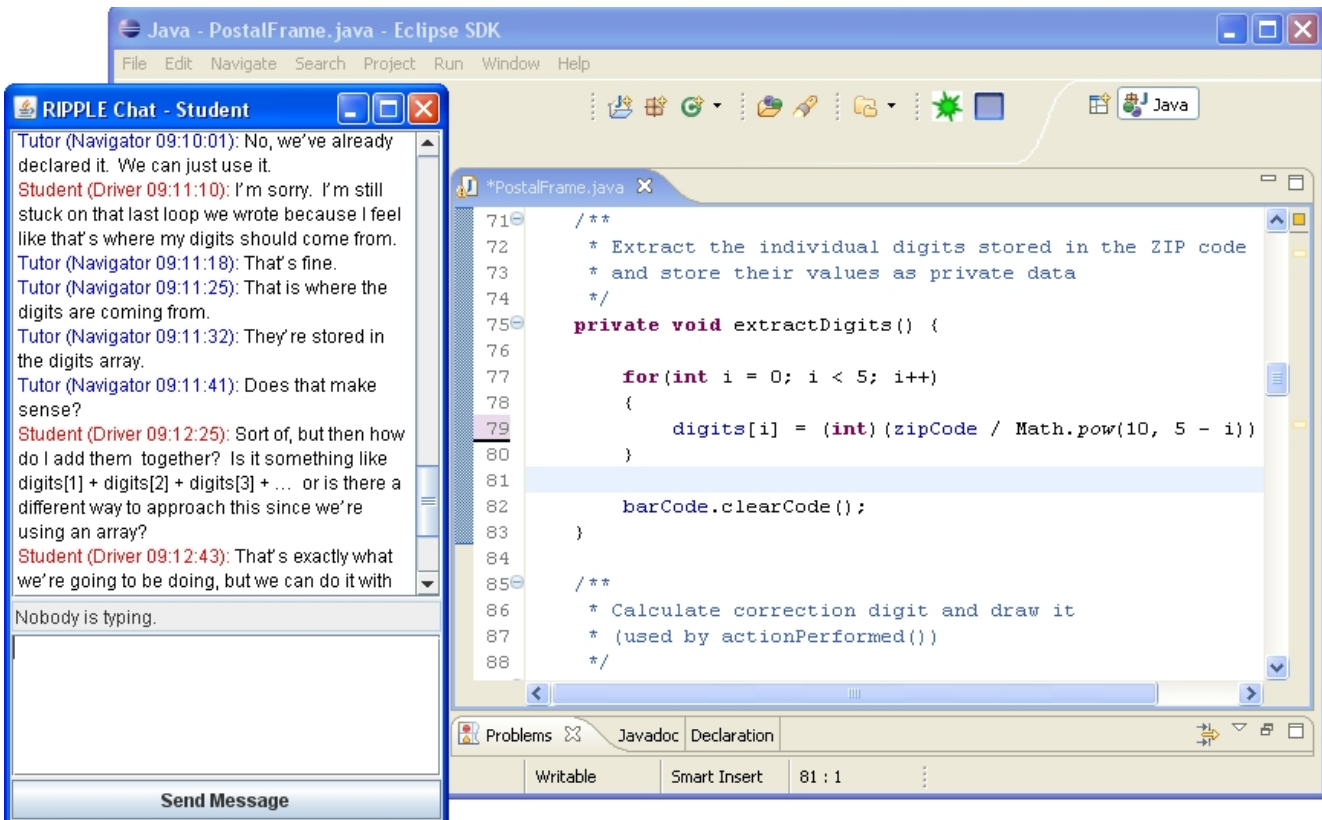


Figure 1: Screenshot of RIPPLE

Table 3

Sample Dialogue Exhibiting the Message Sequencing Problem	
Rohit 11:09:13:	We're just about ready to submit this project.
Sarah 11:09:58:	Sounds good. Hey, do you remember the name of that TA who helped us last week?
Rohit 11:10:00:	We just have to finish documenting it like the assignment describes.

3.3 Data Collection Feature

RIPPLE is a research tool as well as a software tool for programmers. When logging capabilities are enabled, all programming events and textual dialogue messages are transmitted to a database in real time. Data are logged sequentially by user sessions. This information can then be used to reconstruct user sessions for further study. A tool to create properly formed database tables is provided as part of the standard RIPPLE distribution.

4. RESULTS FROM USER TESTS

Prototypes of RIPPLE have been deployed to CS1 classes in controlled studies over two semesters. The pool of 83 subjects was comprised of students enrolled in a Java based CS1 course. Subjects were primarily non-computing majors in engineering disciplines. Rather than attending their eighth regularly scheduled 2-hour supervised laboratory session, these students agreed to complete the laboratory assignment using RIPPLE in a controlled environment, with a programming partner located in a separate room from the student.

4.1 First Classroom Test

The results of the first classroom test show that students enjoyed completing the laboratory assignment more when using RIPPLE compared to previous laboratory sessions in which they worked individually. The results also convincingly demonstrate that the subjects found the system easy to use and would use it again if given the opportunity.

In Fall 2006, 41 students used a RIPPLE prototype for 50 minutes each to work on their regular weekly laboratory exercise. In a pre-survey, only 46% of students agreed that they enjoyed their past CS1 lab assignments, of which there had been seven. After working with RIPPLE for approximately one hour, 85% of students reported that they enjoyed that day's lab assignment, a statistically significant difference ($p < 0.05$). This evidence suggests RIPPLE contributes to creating a more positive student experience during programming assignments. In addition, 95% of subjects stated they would use the system again given the opportunity. Finally, when asked to agree or disagree with the statement that RIPPLE was difficult to use, 100% of students disagreed.

4.2 Second Classroom Test

The second study indicates that student enjoyment of a laboratory assignment on which they used the RIPPLE system was higher than their enjoyment on previous laboratory assignments. These

subjects, like the first group, found the system easy to use and reported they would use it again given the opportunity.

In Spring 2007, 42 students used the RIPPLE prototype for 55 minutes each on the same programming problem that had been used the previous semester. These students had a different classroom instructor and different laboratory teaching assistants from the previous semester's students. On the pre-survey, students expressed slight disagreement to moderate agreement (on a five-point Likert scale) with the statement that they normally enjoyed CS1 lab assignments. After working with RIPPLE the average response was moderate to strong agreement that subjects enjoyed the lab assignment. This difference is statistically significant ($p < 0.05$). Students strongly disagreed with the statement that RIPPLE was difficult to use, and expressed agreement to strong agreement that they would use RIPPLE again given the opportunity.

4.3 Discussion

The results from both user tests indicate positive response to RIPPLE. Although it was well received, the results must be interpreted in the context in which the tests were conducted. First, the user tests were conducted in natural sequence with a set of subjects who were engaged in an authentic CS1 experience. Second, the students had not been exposed to any integrated development environments or graphical programming tools during their previous assignments. On one hand, this lack of exposure could have caused students to respond less favorably to RIPPLE because of its comparatively complex Eclipse-based interface. However, it is more likely that students would have appreciated the conveniences afforded them (e.g., continuous compilation and visual cues noting compiler errors) to such an extent that some portion of the subjects' positive response to RIPPLE must be attributed solely to Eclipse. Third, students had not regularly collaborated on past laboratory exercises, so a portion of the subjects' positive response may also be attributed to collaboration in general.

The two studies reported here were single-use tests, so no long-term effects on learning or other measures are reported. A promising direction for future work is to deploy RIPPLE across an entire semester. It is expected that more extensive evaluation results will be available at the completion of these larger deployments.

5. SUMMARY

RIPPLE is a development environment that supports distributed collaboration between two programmers. In addition to enabling programmers to work together from different locations, RIPPLE also allows researchers to collect data from collaborative interactions both in natural settings and through designed experiments. RIPPLE is an open source tool that is tailored to the needs of collaborative programming in educational settings.

Educational Use. In initial tests, student response to RIPPLE was positive. Students reported enjoying their laboratory programming assignment more when using RIPPLE as compared to situations where they worked solo. Students found the system easy to use, and said they would use it again given the opportunity. Work is ongoing to explore whether learning gains, improvements in student retention, effects on self-efficacy, and other benefits are accrued over time.

Research Utility. To date, RIPPLE has been used successfully in multiple experiments (e.g. [7]). Researchers can take advantage of

logging capabilities enabled through simple configuration options. For research groups with specialized needs, the fully modifiable and extensible Java source code is freely available.

Download. The full Java source code, along with installation, setup, and usage instructions for RIPPLE, are freely available from the project web page at <http://research.csc.ncsu.edu/ripple>.

6. ACKNOWLEDGMENTS

This work is supported in part by the NSF STARS Alliance (CNS-0540523), a NSF Graduate Research Fellowship, and the NC State University Department of Computer Science. The authors wish to thank the members of the Intellimedia Center for Intelligent Systems and the Realsearch Group at NC State University for their ongoing intellectual contributions and project development support.

7. REFERENCES

- [1] ACM/IEEE-CS Joint Task Force on Computing Curricula Final Report. <http://www.sigcse.org/cc2001>, (2001).
- [2] Apple Inc. SubEthaEdit Homepage. http://www.apple.com/downloads/macosx/productivity_tools/subethaedit.html (2007).
- [3] Baheti P., Gehringer E. and Stotts D. Exploring the Efficacy of Distributed Pair Programming. *XP Universe* (2002), 208–220.
- [4] Baheti P., Williams L., Gehringer E., Stotts D. and Smith J. M. Distributed Pair Programming: Empirical Studies and Supporting Environments. TR02-010. University of North Carolina at Chapel Hill Dept. of Computer Science, 2002.
- [5] Beck L. L., Chizhik A. W. and McElroy A. C. Cooperative Learning Techniques in CS1: Design and Experimental Evaluation. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education (SIGCSE '05)*, 2005, 470-474.
- [6] Bloom B. S. The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher*, 13, 6 (1984), 4-16.
- [7] Boyer K. E., Vouk M. A. and Lester J. C. The Influence of Learner Characteristics on Task-Oriented Tutorial Dialogue. In *Proceedings of the International Conference on Artificial Intelligence in Education*, Marina del Rey, California, 2007, 365-372.
- [8] Chi M.T.H., Leeuw N., Chiu M. H. and LaVancher C. Eliciting Self-Explanations Improves Understanding. *Cognitive Science*, 18, 3 (1994), 439-477.
- [9] Emory D. and Tamassia R. JERPA: A Distance-Learning Environment for Introductory Java Programming Courses. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education (SIGCSE '02)*, 2002, 307-311.
- [10] Evens M. and Michael J. *One-on-One Tutoring by Humans and Computers*. Lawrence Erlbaum Associates, Mahwah, New Jersey, 2006.
- [11] Forbes J. and Garcia D. D. ...But what do the top-rated schools do?: A Survey of Introductory Computer Science Curricula. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*, 2007, 245-246.
- [12] Fox B.A. *The Human Tutorial Dialogue Project*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1993.
- [13] Hanks B. F. Distributed Pair Programming: An Empirical Study. *XP/Agile Universe*. 2004, 81-91.
- [14] Ho C.W., Raha S., Gehringer E. and Williams L. Sangam: A Distributed Pair Programming Plug-in for Eclipse. *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, (2004), 73-77.
- [15] Hubscher-Younger T. and Narayanan N. H. Constructive and Collaborative Learning of Algorithms. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education (SIGCSE '03)*, 2003, 6-10.
- [16] Lane H. C. and VanLehn K. Coached Program Planning: Dialogue-Based Support for Novice Program Design. In *Proceedings of the SIGCSE Technical Symposium on Computer Science Education (SIGCSE '03)*, 2003, 148-152.
- [17] Lepper M. R., Woolverton M., Mumme D. L. and Gurtner J. L. Motivational Techniques of Expert Human Tutors: Lessons for the Design of Computer-Based Tutors. *Computers as Cognitive Tools*, (1993), 75-105.
- [18] McDowell C., Werner L., Bullock H. and Fernald J. The Effects of Pair Programming on Performance in an Introductory Programming Course. In *Proceedings of the 33rd technical symposium on Computer science education (SIGCSE '02)*, 2002, 38-42.
- [19] Nagappan N., Williams L., Ferzli M., Wiebe E., Yang K., Miller C. and Balik S. Improving the CS1 experience with pair programming. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education. (SIGCSE '03)*, 2003, 359-362.
- [20] Olson G. M. and Olson J. S. Distance Matters. *Human Computer Interaction*, 15, 2/3 (2000), 139-178.
- [21] The Eclipse Homepage. <http://www.eclipse.org>, 2007
- [22] Werner L., Denner J. and Bean S. Pair Programming Strategies for Middle School Girls. In *Proceedings of the Seventh IASTED International Conference on Computers and Advanced Technology in Education*, 2004, 16-18.
- [23] Werner L. L., Hanks B. and McDowell C. Pair-programming helps female computer science students. *ACM Journal of Educational Resources in Computing (JERIC)* 4, 1 (2004).
- [24] Williams L. and Kessler R. *Pair Programming Illuminated*. Addison-Wesley, Boston, 2003.
- [25] Williams L. and Upchurch R. L. In support of student pair-programming. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education (SIGCSE '01)*, 2001, 327-331.