Enhancing CS Programming Lab Courses using Collaborative Editors

Timothy J. Hickey
J. T. Langton
Kenroy Granville
Richard Alterman
Brandeis University

March 30, 2004

Abstract

This paper describes the pedagogical implications of the GREWPtool (the Groupware Research in Education and the Workforce Project), a same time different place groupware tool built to support synchronous, collaborative coding among small to medium sized groups. GREWPtool exploits the educational benefits of paired programming while extending the model to allow students synchronous control of the same code. This effectively drops the constraints of turn taking, and allows for a richer interaction. The result was a powerful tool that can be used to provide interactive lectures, structure classroom activity, and facilitate paired programming during labs. In this paper we present the design and use of GREWPtool with an emphasis on how it enhances the classroom experience. In particular, we give a qualitative analysis of how this tool has been used in six lab courses over the past two years.

1 Introduction

There has been a great deal of research on collaborative technologies for the computer science classroom [27, 20, 7]. Curiously, most of these efforts have neglected to directly support the core element of any computer science curriculum: programming. Collaborative coding (or "pair programming") has been known and used in industry, and recently shown to be quite effective in academia [26][16][13] [14][1]. We wanted to extend the traditional model of this technique with software that would allow students simultaneous editing of the same code. In doing so we aimed to increase learning through a richer constructivist interaction.

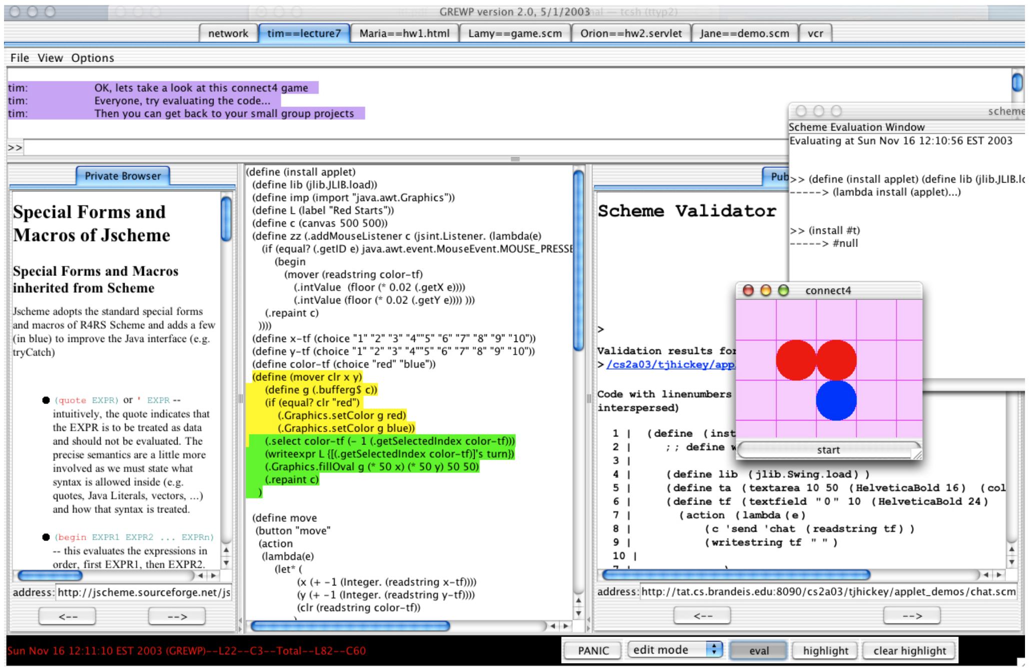
GREWPtool was designed as an Integrated Development Environment (IDE) to support this new model of cooperative coding [11]. Riordan [21] has undertaken a similar effort in building an ILLE, Integrated Learning Laboratory Environment, for novice programmers learning Java. Their software provides some learner scaffolding and allows instructors to interact with students through chat while they code, but only the student is doing any coding. This is analogous to a predecessor of GREWPtool called TATool [6] which, in addition to chat, enables instructors to reference and type next to specific lines of student code. Though useful, neither of these tools support the simultaneous editing of a single code document between pairs of students. Enabling this kind of student interaction is both a technical and a user interface design problem.

GREWPtool was an expansion on work addressing the implementation of shared editors [23], and their design in an educational context [17] [15]. It has not only supplemented the normal curriculum, but introduced benefitial dynamics and effective teaching techniques. In this paper we describe the design, implementation, and advantages of using GREWPtool in the following six computer science laboratory courses and independent study courses:

- TYPCosi/Aut02/Spr03 – 10, bright, inner-city, transitional students learning HTML/CSS, Scheme Servlets, Scheme Applets and TYPCosi/Spr04 – 20, bright, inner-city, transitional students learning HTML/CSS, Scheme Servlets, Scheme Applets
- CS210a/Spr03 – 4-5 local and 2-3 remote Masters/PhD CS students developing 3d graphics libraries in Scheme
- CS2a/Sum03 – 4 non-CS majors learning HTML/CSS, Servlets/Applets, Machine Language, Circuit Design, and Theory of Computation
- CS155a/Sum03 – 12 CS majors/Masters students learning standard 3d Graphics
- INET92a/Sum03 – 4 CS majors developing databased backed servlet widgets including bulletin boards, registration tools, etc.
- CS200/Aut03 – 4 CS graduate students writing a joint paper on Groupware in Education

The TYPCosi course is especially interesting as it is part of a program that brings bright students from under-resourced schools into an elite university for a year of immersion during which time they apply to elite colleges. The students generally have very little experience with computers, but are highly motivated. In the non-major Computer Literacy course (CS2a), the vast majority of the students had no previous experience in computer programming. The other courses were made up of advanced undergraduates or graduate students.

We will discuss how the GREWPtool technology was introduced in these courses. Using this kind of technology provides them both enriches their vocabulary of kinds of software applications while an affording them the opportunity to collaborate on what for them will be challenging material in a manner that helps them overcome their fears with technology. Ideally, their experience with chat provides a sufficent base to allow them to explore other types of interaction. We describe some our experience.



```
tim:          OK, lets take a look at this connect4 game
tim:          Everyone, try evaluating the code...
tim:          Then you can get back to your small group projects
>>
```

Special Forms and Macros of Jscheme

Special Forms and Macros inherited from Scheme

Jscheme adopts the standard special forms and macros of R4RS Scheme and adds a few (in blue) to improve the Java interface (e.g. tryCatch)

- `(quote EXPR)` or `' EXPR` -- intuitively, the quote indicates that the EXPR is to be treated as data and should not be evaluated. The precise semantics are a little more involved as we must state what syntax is allowed inside (e.g. quotes, Java Literals, vectors, ...) and how that syntax is treated.
- `(begin EXPR1 EXPR2 ... EXPRn)` -- this evaluates the expressions in order, first EXPR1, then EXPR2.

address: http://jscheme.sourceforge.net/js

```
(define (install applet)
  (define lib (jlib.JLIB.load))
  (define imp (import "java.awt.Graphics"))
  (define L (label "Red Starts"))
  (define c (canvas 500 500))
  (define zz (.addMouseListener c (jsint.Listener. (lambda(e)
     (if (equal? (.getID e) java.awt.event.MouseEvent.MOUSE_PRESSE
         (begin
            (mover (readstring color-tf)
               (.intValue  (floor (* 0.02 (.getX e))))
               (.intValue (floor (* 0.02 (.getY e)))) )))
       (.repaint c)
     ))))
  (define x-tf (choice "1" "2" "3" "4""5" "6" "7" "8" "9" "10"))
  (define y-tf (choice "1" "2" "3" "4""5" "6" "7" "8" "9" "10"))
  (define color-tf (choice "red" "blue"))
  (define (mover clr x y)
      (define g (.bufferg$ c))
      (if (equal? clr "red")
          (.Graphics.setColor g red)
          (.Graphics.setColor g blue))
      (.select color-tf (- 1 (.getSelectedIndex color-tf)))
      (writeexpr L {[(.getSelectedIndex color-tf)]'s turn})
      (.Graphics.fillOval g (* 50 x) (* 50 y) 50 50)
      (.repaint c)
    )

  (define move
   (button "move"
    (action
     (lambda(e)
       (let* (
              (x (+ -1 (Integer. (readstring x-tf))))
              (y (+ -1 (Integer. (readstring y-tf))))
              (clr (readstring color-tf))
```

Scheme Validator

```
>
Validation results for
>/cs2a03/tjhickey/app

Code with linenumbers
interspersed)

 1 |  (define (inst
 2 |     ;; define w
 3 |
 4 |     (define lib (jlib.Swing.load))
 5 |     (define ta (textarea 10 50 (HelveticaBold 16) (col
 6 |     (define tf (textfield "0" 10 (HelveticaBold 24))
 7 |       (action (lambda(e)
 8 |           (c 'send 'chat (readstring tf))
 9 |           (writestring tf " ")
10 |
```

address: http://tat.cs.brandeis.edu:8090/cs2a03/tjhickey/applet_demos/chat.scm

```
Scheme Evaluation Window
Evaluating at Sun Nov 16 12:10:56 EST 2003

>> (define (install applet) (define lib (jlib.JLIB.lo
-----> (lambda install (applet)...)

>> (install #t)
-----> #null
```

```
Sun Nov 16 12:11:10 EST 2003 (GREWP)--L22--C3--Total--L82--C60
```

Figure 1: View of the GREWPtool

2 GREWPtool

2.1 Design

GREWPtool, shown in Figure 1, is primarily composed of 3 integrated pieces of software in 4 frames (the two smaller windows in the forground are an evaluation of the code shown in the editor component, and the text output of that evaluation). Going clockwise they are: an IM-like chat, a public web browser, a synchronous shared code editor, and a private web browser. We used elastic windows to mitigate the issue of limited screen real estate [3]. Because we did not employ strict WYSIWIS [22] users could reshape the frames according to their personal preferences without effecting their partner's views.

The Chat frame provides a means for explicit communication outside the source code in the editor.

The Public Browser serves as a common resource for students which features links to webpages on syntax and usage. Each member of a GREWPtool group has an associated tab in this frame. This allows students to watch (but not control) each other's browsing activities.

The editor frame supports students writing to the same piece of code simultaneously. Features were kept to a minimum to simplify implementation and use. Cut and paste operations are accomplished with keyboard shortcuts while the File menu allows users to load files and save locally. Each document created is a potential group or shared document (i.e. other users can join the group and edit it) and is associated with a unique tab in the editor pane. This makes it easy for users to work on serveral different documents and/or in several different groups at the same time. In Figure 1 the user has 6 active documents (and an instance of the VCR which we describe later), with the focus on "tim–lecture". The editor also has two modes: edit and watch. Users in watch-mode have their editors scroll automatically with the activity of editing users. Users in edit-mode can type directly into the document.

The Private Browser usually starts with a bulleted list specifying task requirements. Users can then follow links or visit any page of their choosing. The "private" nature of the frame means the user can traverse the web independently of their partners.

At start up each user is automatically assigned a unique color. This is used for their cursor, the last 20 characters they typed into the editor, and the color of their chat. All activity in GREWPtool is recorded and replayable through a VCR mode which features variable delay play, fast forward, and rewind controls. The VCR is a complete replayable transcript of all user interface/domain actions. The approach to VCR technology for Groupware that we follow was first explored in detail by Landman [9, 10] and has enabled new analysis techniques [4].

Implementation GREWPtool was implemented in Jscheme (a Scheme dialect with full and transparent access to Java) as part of the opensource `groupscheme` project hosted at Sourceforge [5]. The editor uses a variation of the Operational Transformation techniques of Ellis and Gibbs similar to that of Jupiter and NetEdit [2, 23, 28, 19]. It relies on a central server to handle event serialization and is comprised of around 5000 lines of Scheme.

3 Collaborative Editor Pedagogy Patterns

In this section we describe our experiences using GREWPtool in the classroom and the pedagogical methods it affords. These can be characterized along the parameters of:

- Group Size: This can vary from having one group containing all students in a class to several groups, each containing 2-4 students and an instructor, to each student having their own group (which the instructor and TAs can join).

- Floor control: Possible options are for the instructor to edit while the students all watch, for one selected student to edit and everyone else watch, or for all students in a group to simultaneously edit.

- Group Goal. The editor can be used for several different purposes including demonstrating programming techniques, teaching debugging skills, assessing student comprehension (through quizzes or group quizzes), and practicing new programming skills.

- Class Focus: The instructor can request the focus of the students by turning on the overhead computer projector and displaying the editing window of any particular group. Alternatively, the instructor can ask all students to work in their own groups (using chat to avoid a noisy classroom).

We enumerate several successful uses of GREWPtool below. Because we are still exploring the utility of this software, this is not meant to be a comprehensive list. For each pedagogical pattern revealed by GREWPtool we present the **Method** of its application and an **Analysis** of its effects.

3.1 The Whole-Class-Group-Coding Pattern

In this pattern, the instructor creates a shared document that all students connect to.

3.1.1 Case Study 1: a 4 person HTML lab

Method The instructor creates a shared document table.html and asks all students to join. The instructor then creates a webpage template containing a table with missing rows. Each student is assigned a row. The students complete their rows and save locally. Everyone views the result of their code in their own browser. The entire group debugs the webpage with individuals taking turns editing their own rows in front of the class. The instructor saves and views the corrected page on the projector.

Analysis The Group Coding allows larger tables to be constructed in less time. The Group Debugging process allows students to learn good debugging skills, as they find problems, discuss possible causes, and try possible fixes for their own and their classmates' code.

3.1.2 Case Study 2: a 12 student Graphics Lab

Method The instructor creates a shared document SimpleTest.java and asks all students to join. Students enter into watch-mode so their editor's scrollpanes automatically follow instructor edits. This is projected on a large screen at the front of the classroom. The instructor loads initial code drawing a rotating pyramid and rewrites it to draw a multijointed arm swinging back and forth. The students save and run locally after each stage in the development. The instructor asks for volunteers to write parts of the code. They all edit the same program at the same time and debug as a whole. Students reference specific lines and ask questions by typing comments into the editor near the code in question.

Analysis Here again, the group can code larger programs in less time since all are working simultaneously. Debugging becomes a group effort benefiting from many eyes and insights. Since all students can see each others code, they are in effect, working on a virtual blackboard. This helps build their programming vocabulary and brings a sense of community to the classroom.

3.2 The Multiple-Small-Group-Coding Pattern

The class breaks up into small groups (2-4 students each). One person in each group creates a shared document that their partners connect to. The instructor and TAs join any or all groups to chat, add code comments, or edit.

3.2.1 Case Study 3: a 12 person Graphics Lab

Method Continuing from case study 2, the instructor asks the class to break into groups of 3-4 (one group per row of computers) and asks each group to modify the code to put a spinning propeller at the end of the multijointed arm. The students communicate by talking rather than chatting, but stay seated at their terminals and each type into their group's program window. The instructor moves electronically from one group to another, offering help where needed. When each group completes the instructor saves, compiles and runs their code, displaying the results at the front of the room with the projector.

Analysis The smaller groups provide a more tightly coupled interaction that is most similar to paired programming. As a result, students in each group learn by asking each other questions, commenting on each other's code, and observing each other's style.

3.3 The Language-Lab Pattern

All students work on their own code. The instructor (and possibly one or more TAs) join these single-student groups to answer questions and make suggestions.

3.3.1 Case Study 4: a 10 person web programming lab

Method Students work on their individual homework assignment, which is to create a personal webpage. Each student creates a shared document and loads in the current version of their homework. The instructor and TAs join each student group and move electronically from student to student observing their work and asking if they need help. Students can also ask for help aloud.

Analysis This is similar to a standard programing lab except that the instructor has easier access to observe students electronically (whether the students are aware of this or not). If one student needs a lot of attention the rest of the class doesn't need to know.

3.4 The Group Debugging Pattern

Students write code and are asked to intentionally insert errors for each other to catch (they occaisionally insert none to keep their classmates on their toes). The entire class then attempts to find errors and suggest corrections.

3.4.1 Case Study 5: a 10 person Web Programming lab

Method Students write examples of HTML code for tables and insert various types of errors (missing brackets, incorrect style parameters, unclosed tags, etc.). They are given about 5 minutes to write their code. The class then debugs each student's program collaboratively.

Analysis This pattern removes the stigma of making mistakes as no one knows if a given bug is intentional or not. Students compete and congratulate each other for creating and finding subtle bugs. Common errors are revealed to inform future debugging.

4 Integrating GREWPtool into the classroom.... a more detailed evaluation

In this section, we discuss our experience integrating this tool into our courses and the way in which it changed the courses. An instructor contemplating the use of such a tool must make many pedagogical choices – when to introduce the tool, how to introduce it, how often to use it. In most of the courses mentioned in section 1, the GREWPtool was introduced early and used almost every class day.

In the medium sized laboratory courses (TYPCosi and the CS155a graphics class) which each had 10-20 students, the tool was introduced in the first week of class. The instructor showed the students how to download the tool as a jar file and how to start it up (double click on the jar) as well as how to create and join collaborative sessions.

In pre-GREWPTool versions of these courses, the lectures were taught with a computer projector where the instructor would demonstrate the programming techniques and relevant concepts on a screen visible to all students. The students would also use part of the lab time working on individual assignments where the instructor (and sometimes TAs) would be available to help individual students. Once the GREWPtool was introduced the courses became much more interactive and there was a noticeable increase in the level of participation among the students.

Consider for example the first type the GREWPtool was introduced in the Spr2004 TYP Cosi class, which had about 20 students, none of whom had any programming experience. The students were initially shown how to start up the tool and how to join the instructor's editing session. Once they had joined they could see the code the instructor was typing appear on the big screen at the front of the classroom, as well as on their own computer.

At the time were studying lists and so it was natural to construct a class list where each student would build their own list element. During this initial programming period, students who did not understand how to create the list element watched others (on their own computer screen) and were able to learn by example. Once everyone had completed (which required some class discussion), the instructor evaluated the HTML and there were many errors (mostly missing punctuation).

This led into a group debugging session where the students as a class would search for errors and in the process we learned some good debugging skills together. After the webpage was debugged, we repeated the cycle by expanding our entries (adding CSS style to change the color, background, and font) and repeated the group coding/ group debugging cycle.

At one point, one of the students remarked that someone had deleted some of his code and this provided an opportunity to use the VCR replay feature to roll back to a point in the class before the code had been deleted. The students were able to reflect on their group performance (in high speed) and get a quick review of the activities of the day, as well as an introduction into how to review the days lecture on their own at a later date.

In the rest of the class, the students were asked to refine their debugging skills by writing a simple HTML page and intentionally introducing one or more subtle errors into the page. To do this they each created a new document and the instructor joined each of the students groups so that he could display any student's program on the main screen. We then went through each student's code and, as a group, attempted to identify and correct the bugs.

At the end of this first 90 minute exposure to GREWPtool, the students had a good introduction to the use of the GREWPtool and also came away with a good understanding about how to create lists and use CSS in HTML, as well as a better understanding of the kinds of mistakes people tend to make and the strategies that can be used to debug programs.

All of our students were regular users of Chat programs and this exposure to network interaction is probably the reason they had no trouble mastering the basic features of the collaborative environment (co-editing). A negative consequence of their familiarity with chat was that they need to be reminded that the chat panel was to be used for academic concerns and not for social conversation. This had to be stressed several times as it was counter to their experience with that user interface, but eventually they learned to chat as if they were raising their hand and making a public comment in class.

While GREWPtool has been quite successful in the classroom, the experience for remote users has been less effective. In the cases where students participated from off-campus, we have found it necessary to assign one student in the class to be the "scribe" for remote students. Their role is to use the chat panel to keep remote students appraised of what is happening in the class (what the instructor is saying, what the class is doing, etc.) The addition of an audio channel might partially alleviate this problem, but even when remote users were connected by telephone there were awareness issues (wondering if they were still listening, etc.) It is likely that classes with all students being remote would remodel communication to alleviate these issues. As it is, one or two remote students are simply left behind the curve of in-class, face-to-face interaction.

5 Final Remarks

In all classes in which we have used the GREWPtool, we have found the students need very little instruction on how to use it. It appears that collaborative editting (two or more keyboards for one document) is a very natural process. We've also found that the classroom takes on a more active feel as everyone is engaged. The students know that their work is either being seen by everyone at the time, or will be shown to the class later, and so participation is mandatory. We have not compared classes that use GREWPtool to classes that use pair programming or extreme programming as the collaborative editing is just one aspect of the way in which this tool changes the classroom dynamics.

Although we have just begun to explore the pedagogical opportunities made possible by this tool, we have observed several key advantages that it offers over traditional programming laboratory techniques.

- Group programming allows students to benefit from each other's insights.
- Group debugging allows students to learn from each other's mistakes.
- Instructors and TAs can rapidly monitor and support multiple individuals and groups with the click of a button.
- Interactive lectures provide students a sense of agency and promote awareness.
- Multiple coders allow larger problems to be tackled in one class session.
- Whole class coding exercises help build a sense of cohesion in the classroom.

References

[1] Timothy H. DeClue, Pair programming and pair trading: effects on learning and motivation in a CS2 course, J. Comput. Small Coll., 18, 5, 2003, 49–56, The Consortium for Computing in Small Colleges.

[2] C. A. Ellis and S. J. Gibbs., Concurrency control in groupware systems., In ACM SIGMOD89 preceedings, Portland Oregon, 1989..

[3] Eser Kandogan and Ben Shneiderman, Elastic Windows: evaluation of multi-window operations, Proceedings of the SIGCHI conference on Human factors in computing systems, 1997, 0-89791-802-9, 250–257, Atlanta, Georgia, United States, http://doi.acm.org/10.1145/258549.258720, ACM Press.

[4] Feinman, A., and Alterman, R., Discourse Analysis Techniques for Modeling Group Interaction., Ninth International Conference on User Modeling (in press), 2003.

[5] Timothy J Hickey, Incorporating Scheme-based Web Programmming into Computer Literacy Courses , Proceedings of the Scheme2002 workshop.

[6] Timothy J. Hickey, J. T. Langton, Kenroy Granville, Richard Alterman, TA Groupware, Tech. Rep. CS-02-222, CS Dept., Brandeis University, 2002.

[7] Klaus Marius Hansen and Anne Vinter Ratzer, Tool support for collaborative teaching and learning of object-oriented modeling, Proceedings of the 7th annual conference on Innovation and technology in computer science education, 2002, 1-58113-499-1, 146–150, Aarhus, Denmark, http://doi.acm.org/10.1145/544414.544458, ACM Press.

[8] Seth Landsman and Richard Alterman, Analyzing Usage of Groupware, Tech. Rep. CS-02-230, CS Dept., Brandeis University, 2003.

[9] Seth Landsman and Richard Alterman, Building Groupware On THYME, Brandeis University Tech Report CS-03-234

[10] Seth Landsman and Richard Alterman, Analyzing Usage of Groupware Brandeis University Tech Report CS-02-230.

[11] J. Langton, T. Hickey, and R. Alterman, Integrating Tools and Resources: a case study in building educational groupware for collaborative programming to appear in CCSCNE 2004, (23-24 April 2004) Schenectady, NY.

[12] Lydia M. S. Lau and Jayne Curson and Richard Drew and Peter M. Dew and Christine Leigh, Use of Virtual Science Park resource rooms to support group work in a learning environment, Proceedings of the international ACM SIGGROUP conference on Supporting group work, 1999, ISBN: 1-58113-065-1, pp. 209–218, Phoenix, Arizona, United States, http://doi.acm.org/10.1145/320297.320322, ACM Press.

[13] Charlie McDowell and Linda Werner and Heather Bullock and Julian Fernald, The effects of pair-programming on performance in an introductory programming course, Proceedings of the 33rd SIGCSE technical symposium on Computer science education, 2002, 5-58113-473-8, 38–42, Cincinnati, Kentucky, http://doi.acm.org/10.1145/563340.563353, ACM Press.

[14] C. McDowell, B. Hanks, L. Werner, Experimenting with Pair Programming in the Classroom, Proceedings of the 8th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, 2003, 60–64, Thessaloniki, Greece, ACM Press.

[15] Alex Mitchell and Ilona Posner and Ronald Baecker, Learning to Write Together Using Groupware, CHI, pp. 288-295, 1995.

[16] Nachiappan Nagappan and Laurie Williams and Miriam Ferzli and Eric Wiebe and Kai Yang and Carol Miller and Suzanne Balik, Improving the CS1 experience with pair programming, Proceedings of the 34th SIGCSE technical symposium on Computer science education, 2003, 1-58113-648-X,

359–362, Reno, Navada, USA, http://doi.acm.org/10.1145/611892.612006, ACM Press.

[17] Neuwirth, C., Kaufer, D., Chandhok, R. And Morris, J., Issues in the design of computer support for co-authoring and commenting. , Proceedings of the third conference on CSCW, 1990, pp. 183-195, Baltimore, MD, USA, ACM Press.

[18] J. T. Nosek, The Case for Collaborative Programming, CACM, pp. 105-108, 1998.

[19] D.A. Nichols, P. Curtis, M. Dixon, and J. Lamping, High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System, Proceedings of ACM UIST 95, November 1995, 111–120.

[20] Lucia Rapanotti and Canan Tosunoglu Blake and Robert Griffiths, eTutorials with voice groupware: real-time conferencing to support computing students at a distance, Proceedings of the 7th annual conference on Innovation and technology in computer science education, 2002, 1-58113-499-1, 116–120, Aarhus, Denmark, http://doi.acm.org/10.1145/544414.544451, ACM Press.

[21] Denis Riordan, Towards an integrated learning laboratory environment for first-year computer science students, SIGCSE Bull., 34, 4, 2002, 0097-8418, 112–116, http://doi.acm.org/10.1145/820127.820180, ACM Press.

[22] Stefik, M., Bobrow, D., Foster, G. Lanning, S., and Tatar, D., WYSIWIS Revised: Early experiences with multi-user interfaces, ACM TOIS, 5(2), 147–167, 1997.

[23] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen., Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems., ACM Transactions on Computer-Human Interaction, 5(1):63?108, Mar. 1998.

[24] Svetlena Taneva and Richard Alterman and Kenroy Granville and Michael Head and Timothy J. Hickey, GREWPTool: A System for Studying Online Collaborative Learning, Brandeis Computer Science Tech Report, 2004.

[25] Laurie Williams and R. R. Kessler, Experimenting with Industry's 'Pair-Programming' Model in the Computer Science Classroom, Journal on SW Engineering Education, Dec. 2000.

[26] Laurie Williams and Richard L. Upchurch, In support of student pair-programming, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, 2001, 1-58113-329-4, 327–331, Charlotte, North Carolina, United States, http://doi.acm.org/10.1145/364447.364614, ACM Press.

[27] Teresa Hubscher-Younger and N. Hari Narayanan, Constructive and collaborative learning of algorithms, Proceedings of the 34th SIGCSE technical symposium on Computer science education, 2003, 1-58113-648-X, 6–10, Reno, Navada, USA, http://doi.acm.org/10.1145/611892.611919, ACM Press.

[28] Zafer, A., NetEdit: a Collaborative Editor, Master of Science, University de Virginia, USA, 2001, 82 pages.

[29] Bernd Zupancic and Holger Horz, Lecture recording and its use in a traditional university course, Proceedings of the 7th annual conference on Innovation and technology in computer science education, 2002, 1-58113-499-1, 24–28, Aarhus, Denmark, http://doi.acm.org/10.1145/544414.544424, ACM Press.