# Integrating tools and resources: a case study in building educational groupware for collaborative programming *

John Langton
Computer Science Department
Brandeis University
Waltham, MA, USA
psyc@cs.brandeis.edu

Tim Hickey
Computer Science Department
Brandeis University
Waltham, MA, USA
tim@cs.brandeis.edu

Richard Alterman
Computer Science Department
Brandeis University
Waltham, MA, USA
alterman@cs.brandeis.edu

## ABSTRACT

This paper presents design implications for educational groupware as revealed by GHT (Group Homework Tool), a same time different place groupware tool built to support synchronous, collaborative coding among novice programmers. We detail the design, implementation, evaluation, and redesign of GHT, focusing on the issues of awareness, control, evaluation and scaffolding. GHT capitalizes on trends of technology and collaboration in the traditional learning environment by supporting distance learning, remote access to TAs and tutors, and facilitating co-located and remote group work. Constructing such software for a computer science curriculum provides unique challenges as one must integrate the tools used by a programmer, the resources used by a learner, and the widgets used to support group interaction. By combining common groupware components with our own shared editor we were able to exploit the educational benefits in a modified version of extreme programming [1]. Our research informs future design efforts by building upon previous investigations of integrated, cooperative software in a learning environment [10, 12, 11].

## 1. INTRODUCTION

Educational institutions are increasingly embracing new technologies and software to aid instruction [19, 20, 21] At the same time, the concept of cooperative learning [8] is gaining acceptance and a place in the classroom [22]. We wanted to exploit the benefits of both in a same time/different place groupware tool built for novice programmers in an introductory computer science course [7]. To do this we needed to balance the needs of a learner with those of a programmer while supporting coordination. This unique set of constraints revealed equally unique design imperatives for educational groupware in the areas of awareness, control, evaluation, and scaffolding.

*This work was supported by the National Science Foundation under Grant No. EIA-0082393

Soloway, Papert, and others [17, 15, 13] have recommended a pragmatic approach to the design of educational software and prescribe a set of general guidelines. Such guidelines also exist for groupware [6, 14, 2]. While the goals of each are not mutually exclusive, they sometimes have conflicting interests. One divergence we happened upon while designing our tool is the issue of explicit communication: while commercial groupware often tries to reduce the amount of communication necessary during focal points of coordination, we wanted to utilize it as an opportunity for students to participate in joint exploration and "sense-making" [16].

To negotiate these disparate requirements we integrated various software tools, following the examples of [11]. In this paper we present

- the combination of groupware components that we found most effective for supporting students in an introductory computer science course.
- insights into issues of awareness and control in the design of educational groupware to promote cooperative learning and support synchronous collaborative coding.

The following sections cover our iterative design process, results of an experiment using GHT with novice programmers, methods of evaluation, redesign, and guidelines for future design efforts.

## 2. REQUIREMENTS ANALYSIS

### 2.1 Motivation

The potential benefits of educational groupware have been widely recognized [5]. Today's students are frequently dispersed whether they are adult learners, industry professionals seeking new skills, or off campus undergraduates. In this environment groupware can facilitate distributed lectures and TA sessions, however its advantages far exceed distance learning. GHT was built to exploit the benefits of cooperative learning during both co-located and remote groupwork. One of our main goals was to lower the overhead for a variety of students learning how to program, including non-science majors. To do this we needed to establish

a model of coordination, an effective combination of groupware tools, and sufficient scaffolding for initiating and structuring collaboration.

Some influential educational theories state that learning takes place when students are actively engaged in collaborative sense-making activities among a community of actors ; that learners' participation in constructive activities help them internally represent the knowledge they are seeking to acquire. The common denominator of these theories is that we can foster learning by providing students with the opportunity to:

1. use the tools of the targeted domain of knowledge, and
2. coordinate with peers during tool use within that domain.

In the case of supporting groups of students learning to program this meant providing an Integrated Development Environment (IDE) designed to facilitate cooperative coding. Traditional models of collaborative programming employ the use of concurrency control software like CVS. These tools, however, minimize the required communication between coders, an essential ingredient in cooperative learning. We chose instead to look for a more interactive model.

The idea of maximizing interaction and communication between coders is quite popular in the form of extreme programming [1]. The benefits of this innovative theory have been debated strongly, and are most widely recognized in industry. However, participants of pair programming have repeatedly noted its educational benefits [10].

In designing GHT we decided to extend this approach to support groups of arbitrary size synchronously coding. We anticipated that during group assignments students would establish a consensus on the structure of their activity, break up the work, and then independently program in tandem. This would promote modular code while allowing questions and statements during momentary reviews of each other's work. We further surmised that students would need to agree when to compile and how to debug due to the interdependence of their code. This would provide an opportunity for them to enter into explanatory practices should their code not produce the expected result. It would also enable students to engage in self-explanations by following the example of a professor's code during class and a TA's corrections during office hours.

Having established a primary tool and framework for coordination we needed to provide students with the necessary resources for completing an assignment. These could be as simple as a task definition and API specification, however needed to impart some structure to the activity. This structure would act as scaffolding to help initiate collaboration and guide solution production. In this way the responsibility for establishing roles could be transferred from groupware design to user interpretation. We speculated that for open ended assignments a brain storming component would also be benefitial.

## 2.2 Components

Previous studies have shown effective integrations of groupware for aiding instruction [11, 10] Determining the "effective integration" for GHT would only be revealed through repetitive design cycles. For our initial design we combined the following components:

- **A synchronous code editor:** to support students writing to the same piece of code simultaneously.
- **HTML frames for an assignment definition and resource page:** to act as scaffolding in starting the assignment and structuring coordination
- **Chat:** to facilitate explicit communication
- **Shared Whiteboard:** to help the students brainstorm various aspects of their project (e.g. GUI layout).

Students were well acquainted with single user versions of editors, whiteboards, web browsers and multi-user chat.

## 2.3 Coordination

After selecting our components we had to address the following standard groupware concerns:

- **Awareness:** users need to know where they and their partners are in a shared workspace [5, 12].
- **Control:** who has control of the shared workspace can be a complicated issue. Some systems mandate turn taking [10] while others allow users to choose [20].

The difficulty in addressing these issues lay in maintaining a balance between tools and learner resources. The scaffolding provided by our HTML frames would impose constraints on how students interact. We were led to ask questions like:

- Should we let students decide how to break up the work or specify this in the assignment definition?
- If we assign roles will we constrain coordination and the resulting benefits of collaboration?
- If we don't will students spend too much time figuring out where to start and less time figuring out how to program?

To fulfill our design requirements we followed an iterative process similar to that prescribed by [13]. First we analyzed how programmers work and assessed where learners would need guidance while participating in group activities. We then designed scaffolding keeping in mind the collaborative nature of GHT and the tradeoff between ease of collaboration and structure in activity. Both of these processes were informed by repetitive groupware walkthroughs [17]. The next section describes the initial design of GHT.

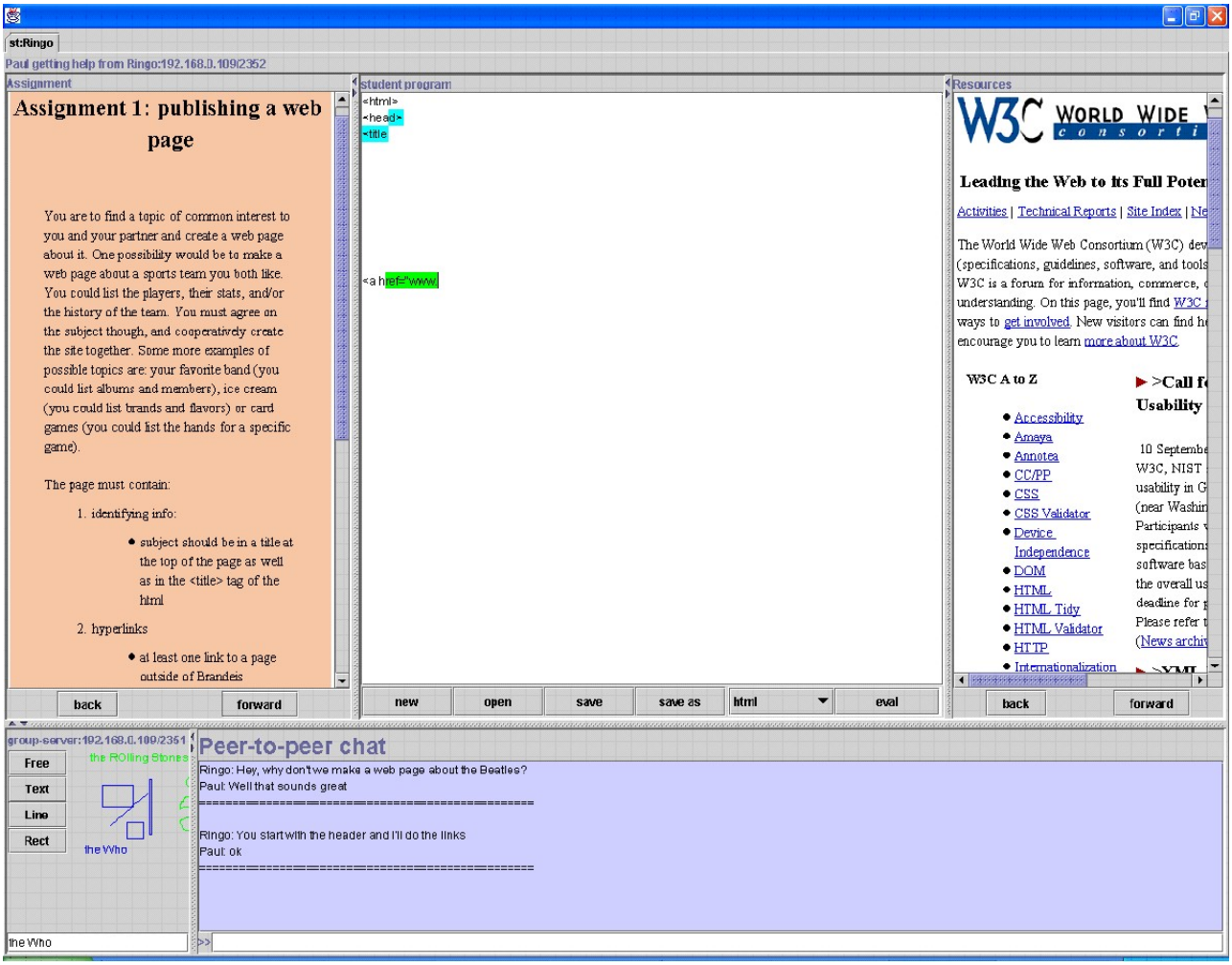


**Figure 1: View of the GHT**

## 3. DESIGN

GHT is comprised of 4 integrated pieces of software in 5 frames as shown in Figure 1. Going clockwise they are: a HTML frame presenting the current assignment, a synchronous shared code editor, a HTML frame providing links to resources on syntax and usage, an IM-like chat, and a white board. To ameliorate the problem of limited screen real estate we used elastic windows [11]. Because we did not employ strict WYSIWIS users could reshape the frames according to their own personal preferences without effecting their partners' views.

*The Assignment frame* was comprised of a bulleted list specifying task requirements. These were open ended imperatives such as "include a button with an associated action in your program". Users were free to implement the specifics in whatever way they saw fit. This allowed multiple correct solutions to each requirement thereby promoting coordination among users as they try to reach a consensus on one. Any more structure would have constrained the interaction, limiting the students' need and desire to explore possible solutions. By ordering the bulleted points in a top down fashion students could iterate through them in a stepwise procedure. This facilitated a kick start to the process and aided in partitioning the work load. We surmised that the assignment would need to be referred to more frequently than the resources. Each was therefore provided an individual frame so that users could collapse the resources without losing the assignment.

*The editor frame* supported students writing to the same piece of code simultaneously. Features were kept to a minimum to simplify implementation and use. Cut and paste operations were accomplished with keyboard shortcuts. As a secondary scaffolding support students were supplied with skeleton code files. These contained basic elements of the assignments such as the body tag of an html page, and could be loaded into the editor via a "File" menu.

*The resources frame* served as a final scaffolding support. Using it, users could refer to examples and syntax references anywhere on the internet. The starting page provided links deemed practical by the instructor.

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

*The whiteboard frame* allowed students to brainstorm various aspects of their program (e.g. GUI design). Again features were kept to a maximum. Users could draw lines, rectangles, free hand, or type onto the picture.

### 3.1 Mediating coordination:

**Awareness**: At start up each user was automatically assigned a unique color. This was used for their cursor, the last 20 characters they typed into the editor, and their activity in the whiteboard. Students could easily identify where everyone was as long as they could remember who was assigned which color. We considered adding a legend, however identity in this case was relative; knowing the name of the blue user did not matter as much as knowing that the same person who drew the blue square was coding on line 3 in the editor.

**Control**: In our initial design we intended to establish roles for students explicitly i.e. one student would program the interface while another would program the backend, or one student would comment while another would code. This schema utilized a locking mechanism so one person could not interfere with the activities of another. In the end we decided that this would put too much of a constraint on how users interact, disallowing simple operations like commenting and coding on the same line. Instead we gave users control of the same text at the same time. This opened up possibilities for coordination and thus promoted cooperative learning.

## 4. IMPLEMENTATION

GHT was implemented in Jscheme [7], an open-source implementation of Scheme in Java with full and transparent access to the java libraries: (jscheme.sourceforge.net). We chose this language for several reasons:

- our familiarity with the language and the ease with which we were able to use it to rapidly prototype the system. Especially notable is the ability to use the read/eval/print loop to develop code incrementally.
- its ability to run on many platforms (thanks to its Java-based implementation). It is currently implemented in about 6000 lines of Jscheme code (including all of the networking libraries) and is deployed as a 600K double-clickable jar file.
- its simple, transparent interface to Java, which provides it with full access to the extensive Java libraries. The java.nio.* package was especially useful, as were the javax.swing packages.

The low level networking was implemented by sending s-expressions (nested lists of symbols, strings, and numbers) as text through sockets. This flexible interface greatly enhanced our ability to rapidly prototype the system. The socket handling was all implemented using the java.nio package. One thread was responsible for managing all sockets and socket servers. Since java.nio supports non-blocking I/O this scales to hundreds of sockets, easily meeting the requirements for the GHT project.

Each group of collaborating GHT clients is associated with a broadcast server which allows clients to broadcast messages to all members of the group. The chat panel, shared whiteboard, and cobrowsers were all implemented by sending simple messages to the associated server. The only widget whose implementation was moderately complex was the shared editor.

We implemented a low-level layer of libraries for creating group clients and group servers as well as a library of basic widgets. Each widget was built with two operational modes – the standard mode and a replay mode in which operations come from a log file rather than from user inputs. The code for this project and the libraries is available as part of the opensource groupscheme project (groupscheme.sf.net).

The shared editor was implemented using a simple variant of the well-known operational transformation approach intro-

duced by Ellis and Gibbs and used in most other collaborative editors today [3, 18]. The main difference between our approach and that of other systems is that we use a remote echo (or full duplex) method in which insertions and deletions are not executed locally on the client until after they are transformed and bounced back by the server. The server accepts all insert/delete requests and merges them (which entails some modification of the operations). The clients then apply the modified transformation to the text. This introduces a small amount of lag (perceived as type-ahead), but there were no complaints about this behavior from students that used the system. The lag time for clients within a metropolitan area (on broadband or modem connections) is generally under 0.3 seconds.

The IDE features supported by the tool include evaluation (and some highlighting) for Jscheme programs, for HTML code, and for Java code.

# 5. EVALUATION

To evaluate the effectiveness of our tool we conducted an ethnographic study of 6 students using GHT in pairs. Subjects were placed at individual terminals out of each other's sight with the frames of GHT arranged as in Figure 1. There were 2 sessions per pair, each lasting two hours and entailing a different assignment. The first assignment was to code a webpage using html, and the second was to code a simple application using JScheme.

We conducted a brief exercise with study participants during their first session to familiarize them with the features of GHT. This contained the following directions: "Please greet your partner using the chat frame. Ask your partner for the name of a shape, then draw that shape onto the whiteboard. Load the file 'assignment1.html' into your editor (only one person needs to do this). Press the evaluate button on the editor." After pressing the evaluate button users would see the results of the skeleton html code for the first assignment. This was a mostly blank html frame with one or two words saying something to the effect "code here." If there were any problems the researcher would provide guidance, however this was kept to a minimum so as not to bias the study.

GHT has a built-in logging system which records user actions during tool use. The VCR mode allows an analyst to play back a log (moving forward and backward, single step or fast forward) showing the interface from any student's view. This provided us with the ability to play back sessions [9] and conduct an in-depth discourse analysis of all chat[4]. Additional logging enumerated everything from mouse clicks to frame resizing. To insure that nothing was missed and enforce communication only through the tool, investigators were present during test sessions and kept personal notes of interesting points in student interaction.

## 5.1 Observations

**Awareness:** The colored cursors of GHT effectively informed users of who was where. In a number of surveys there were requests that there be a "panic button" or some sort of device to capture the attention of a partner who was not paying attention to chat. The obvious solution would be to add aural cues for changes in the chat panel status as this is most common technique employed by popular IM clients. In the evaluation trials, there were no aural cues as the students were in public areas where the computers are muted. One interesting point here is that users could easily go to the line of code where their partner was typing and insert a few characters to get their partners attention, however students felt that this was either too many steps, not fool proof, or considered too invasive. In either case this is a testament to the fact that group coding consists of cycles of waxing and waning collaboration.

*cobrowsing* We also observed that users spent a considerable amount of time coordinating their web browsers. For example, one group was building a web page describing their favorite basketball team and they took turns searching for information on the web and inserting it into the page. They each took responsibility for a subset of the team members (effectively modularizing the problem), but to share results of web searchs they had to type in URLs into the chat window, which frequently generated typos, confusion, and delay. This resulted from their not being aware of the contents of the other person's web browser.

**Control**: Our purposeful lack of control structure forced users to collaborate to a greater degree as they not only needed to agree on what they were coding, but at what points to stop and evaluate the code. These deliberations contributed to learning since it required a clear conceptualization of code requirements, interdependency of code segments (if one users code depended on another users unfinished code it wouldn't evaluate), promoted modularity in coding style (since students were coding different parts they only wanted to worry about how their code would access the methods of their partners code, not how those methods worked necessarily), and milestones in programming (students evaluated often instead of coding the entire program and then wasting time debugging a nightmare).

The minimalist approach to scaffolding was very effective. Users did in fact complete the bulleted points in the assignment description as a stepwise procedure. The most complicated cognitive task seemed to be agreeing on a theme for the web page or the jscheme application.

*Collaborative editing* Every group lauded the ability to edit program code in the same editor at the same time. One participant remarked that it was the first time a group programming assignment actually took half the time. Other methods presumably take more time because of the overhead of splitting up the work in the beginning and then merging the completely written pieces at the end.

The shared editor was used as a medium for communication as well as an editor: instead of chatting about what line of code their partner had in error, a user would often go to the specific line of code in the editor and write their comments there.

# 6. REDESIGN

The main issues we identified in the evaluation phase were:

- a rough transition when students moved between independent work and close collaboration. Users wanted to

be able to easily see where their partners were editting and to capture their attention.

- a need to coordinate more effectively in web browsing
- the whiteboard was never used.

To address the problems uncovered during evaluation, we added a few new features and are currently observing their impact:

- **watch mode vs edit mode:** We observed that a common dynamic in the beginning of a session is that one student would be the primary coder, while the other student would comment on the code, ask questions, make suggestions, etc. in a style very similar to pair programming in the extreme programming model. After a few minutes of this sort of programming students would often branch off to work on independent parts of the program, and cycle back and forth between close and loosely coupled interaction. This pattern of collaboration allows a less experienced student who is initiallly incapable of working independently meaningfully participate in her zone of proximal development [Vygotsky, 1978].

  To support this behavior, we added a watch-mode in which the student's editor window would automatically scroll to keep the other students' edits in view. By switching to edit mode the student could turn off this following behavior. In our current re-evaluation, the watch-mode/edit-mode component seems to be the most used modification. It indicates both a change in role and a change in functionality.

- **cobrowsing tabs:** To help coordinate browsing we expanded the help browser to include tabs for each member of the group. This allows a student to watch another students browsing activities (but not to control their browser)

- **removal of the whiteboard:** Since the whiteboard was never used, we decide to remove it and free up some screen real estate. For other problem sets it is likely that the whiteboard would be more useful, but for the problem sets we were considering it hindered the tools effectiveness by limiting the size of the other tools.

- **a panic button:** We decided to give the students what they asked for (which is not always the right choice). In this case, we added a button (with the label "panic"). Pushing this button pops up a large window with the text: "Please talk to me right now! – STUDENT'S NAME" Unsurprisingly, it appears that the "panic button" is rarely used. Students try it out in the beginning and then decide that it is too invasive.

## 6.1 Reevaluation

We are currently in the process of evaluating these changes in another experiment involving 10 pairs and 5 singles working on a 2d graphics programming assignment in Jscheme. The participants were given pre/post tests, surveys, and of course fully logged group sessions.

The participants had widely varying backgrounds, from humanities majors to graduate students in computer science. They were all able to complete the assignment within about 90 minutes with minimal training and they enjoyed using the tool. This provides evidence that the tool is simple enough to be readily mastered by novices, yet powerful enough to facilitate effective group interaction for programming assignments.

The cobrowsing seems to have answered the complaint about difficulties in coordinating around web locations. Likewise, there have been no calls for a whiteboard or other graphical communication tool. The small lag time in the editor (arising from our simple "remote echo" approach) also did not generate any comments or complaints.

The redesigned tool (renamed GREWPtool) is shown in Figure 2. The VCR component of that tool is shown in Figure 3.

# 7. OTHER APPLICATIONS

Although the GHT was designed for students collobarating on group programming assignments, we found that it was also being used to allow teaching assistants to help students with their programming assignments. To simplify this interaction we modified the tool to allow a TA to participate in several group projects simultaneously. This was implemented using a tabbed pane, with one tab providing an interface to join new groups or to create a shared document, and the other tabs representing individual shared documents. The logging/VCR feature has proved useful as well as it provides a mechanism for a TA to review what a particular student or group has been doing while the TA was working with other students. We have plans to use the logging/VCR feature to allow instructors to explicitly monitor the quanitity and quality of help that TAs provide students as well as to help mentor TAs in their interactions with students.

Once this tabbed architecture was introduced, we found that the tool worked well in a computer laboratory classroom in which students would work together in groups of two to four. The instructor could then join each group and display the work of any particular group on the overhead monitor so that the class as a whole could see and analyze the work of any group. The tool has been used in this capacity for classes ranging in size from 6 to 15. We also found that students who were not able to physically come to the class would connect from home (via modem or broadband) and interact remotely. This was only marginally successful and required a copresent participant to use the chat window to keep the non-collocated participants aware of what was happening in the classroom.

# 8. CONCLUDING REMARKS:

There are several lessons that can be drawn from our experience.

Firstly, our study validates the utility of evaluating groupware with a close moment-by-moment analysis of the interactions of real users on a real problem. This approach formed the foundation of our evaluation scheme and effectively pointed out which features were useful and which were

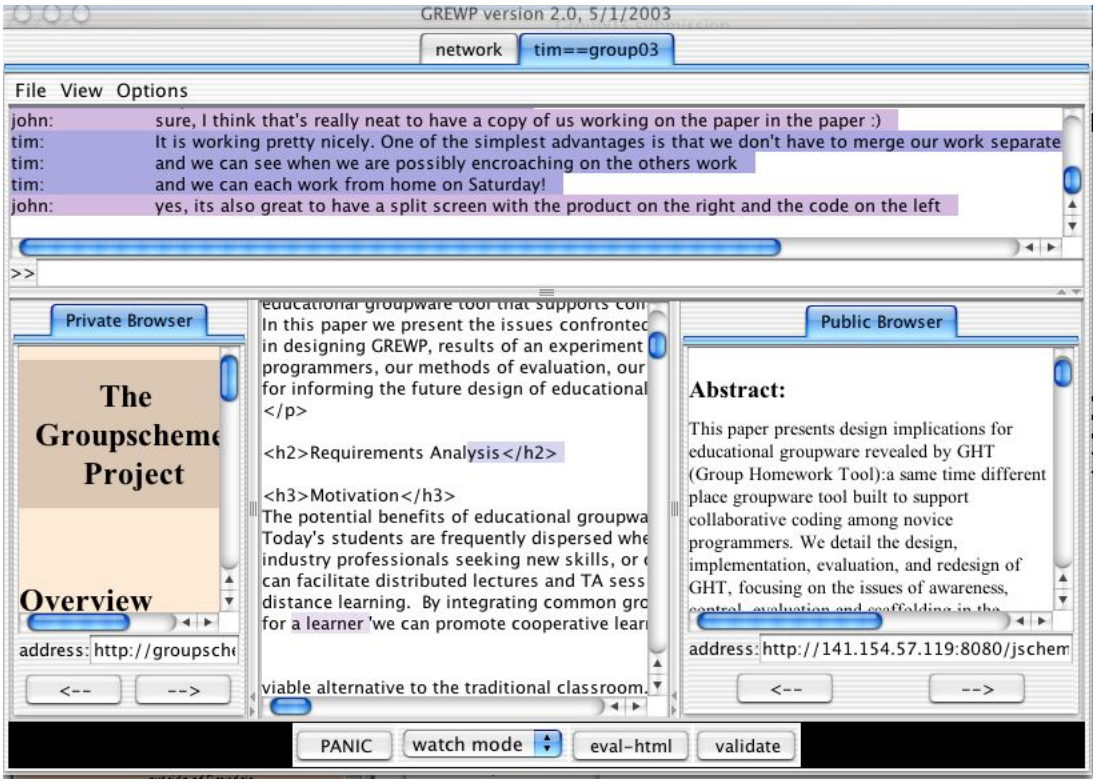


**Figure 2: Redesigned GHT tool**

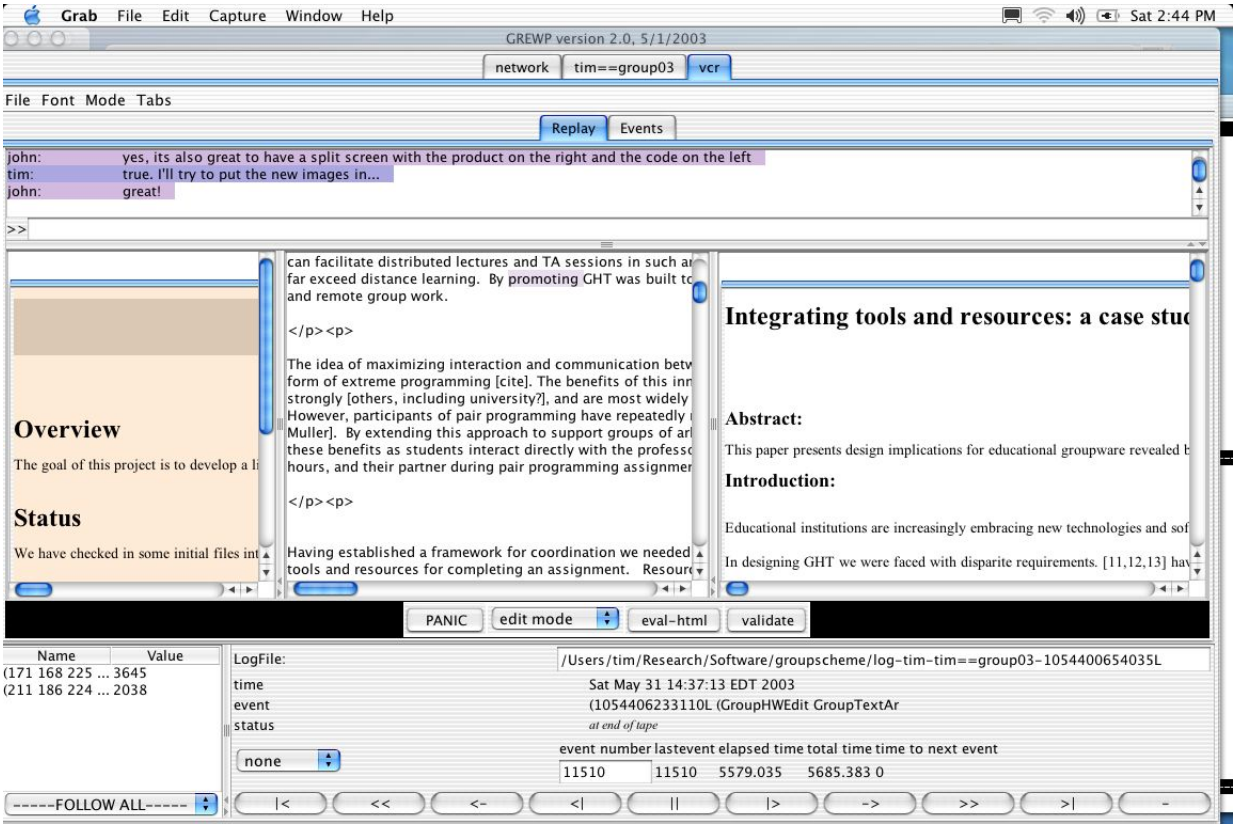


**Figure 3: The VCR tool in the redesigned GHT**

counter productive. The methodology we employed in the design was to purposefully limit the number of features, with the intent of adding them if close analysis revealed a need. Surprisingly, the analysis did not show any need to eliminate the lag time in the group editor. It did point out areas where the tool could support coordination (such as in cobrowsing). Although one could possibly reach these same conclusions with user interviews or by reviewing videotapes of the group interactions, we found the VCR method to be both easy to use and revealing. On average, an analyst can scan through a one hour group interaction in approximately five to ten minutes, although some sections require rewinding, replay, and thoughtful consideration.

Secondly, our study demonstrated that the combination of three familiar components (chat, editor, browser) with some IDE features formed an easy to use environment for working on group programming assignments. Students needed to decide how to use the tool by themselves using the tool itself. All groups were able to complete their assignments within one hour and they seemed comfortable using the tool. The only concept that was somewhat novel to them was that two people could edit the same document at the same time, without turn taking, but they quickly adapted to that. Surprisingly, the tool also turned out to be useful in a number of other contexts for which it was not specifically designed (such as facilitating student-TA interactions, and as a tool for coordinating and highlighting student work in a computer lab).

Finally, our evaluation verified our assumption that the degree to which students interacted would naturally wax and wane over the course of the groupware session. Adherence to a strict role-based model or to a turn-taking model would have prohibited this type of interaction and resulted in a much different educational experience for the participants. We are currently evaluating the educational effectiveness of the tool, but it is too early to tell whether this open approach to collaboration will prove more effective than a traditional lecture style of teaching.

## 9. REFERENCES


[1] K. Beck, Embracing change with extreme programming. IEEE Computer, pages 70-77, Oct. 1999.

[2] Ehrlich, K. Designing Groupware Applications: a Work-Centered Design Approach. Computer Supported Co-operative Work, Beaudouin-Lafon, M. (ed.), Wiley, Chichester, 1999, l-28.

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

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

[5] Melissa Glynn and Doug Vogel and Robert Briggs and Howard Brown and Debra Cunningham, Issues in technology supported learning (panel), Proceedings of the international ACM SIGGROUP conference on Supporting group work : the integration challenge, 1997, ISBN: 0-89791-897-5, pp. 7–8, Phoenix, Arizona, United States, http://doi.acm.org/10.1145/266838.276983, ACM Press.

[6] Grudin, J. 1994. Groupware and Social Dynamics: Eight Challenges for Developers. Communications of the ACM, 37, 1, 92-105.

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

[8] Johnson, D, Johnson, R, Smith K, Active Learning: Cooperation in the College Classroom, ISBN 0-939603-14-4, 1998

[9] Seth Landsman and Richard Alterman Analyzing Usage of Groupware: THYME Is On Your Side, Brandeis University Tech Report CS-02-230.

[10] 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.

[11] Joan Manuel Marquis and Leandro Navarro, WWG: a wide-area infrastructure to support groups, Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work, 2001, ISBN: 1-58113-294-8, pp. 179–187, Boulder, Colorado, USA, http://doi.acm.org/10.1145/500286.500314, ACM Press.

[12] Lisa Neal, Virtual classrooms and communities, Proceedings of the international ACM SIGGROUP conference on Supporting group work : the integration challenge, 1997, ISBN: 0-89791-897-5, pp. 81–90, Phoenix, Arizona, United States, http://doi.acm.org/10.1145/266838.266868, ACM Press.

[13] Papert, S. 1993 The Children's Machine: Rethinking School in the Age of the Computer, Basic Books, NY.

[14] Pinelle, D., and Gutwin, C. 2002 Groupware Walkthrough: Adding Context to Groupware Usability Evaluation. In Proceedings of CHI 02 (Minneapolis, Minnesota, April 2002) ACM Press, 455-462.

[15] Chris Quintana and Joseph Krajcik and Elliot Soloway, A Case Study to Distill Structural Scaffolding Guidelines for Scaffolded Software Environments, Proceedings of the SIGCHI conference on Human factors in computing systems, 2002, ISBN: 1-58113-453-3, pp. 81–88, Minneapolis, Minnesota, USA, http://doi.acm.org/10.1145/503376.503392, ACM Press.

[16] Schoenfeld, A. H. (in press) On mathematics as sense-making: An informal attack on the unfortunate divorce of formal and informal mathematics. In D. N.

Perkins, J. Segal, and J. Voss (Eds.), Informal reasoning and education. Hillsdale, NJ: Erlbaum.

[17] Elliot Soloway and Shari L. Jackson and Jonathan Klein and Chris Quintana and James Reed and Jeff Spitulnik and Steven J. Stratford and Scott Studer and Jim Eng and Nancy Scala, Learning theory in practice: case studies of learner-centered design, Proceedings of the SIGCHI conference on Human factors in computing systems, 1996, ISBN: 0-89791-777-4, pp. 189–196, Vancouver, British Columbia, Canada, http://doi.acm.org/10.1145/238386.238476, ACM Press.

[18] 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.

[19] "Logiciels libres et education". Retrieved 5/27/03, from http://libresoftware-educ.org/fr/carteFrance.html

[20] Retrieved 5/27/03, from http://www.rittershofer.de/info/linux/linout.htm

[21] "Software libero e educazione". Retrieved 5/27/03, from http://libresoftware-educ.org/it/carteItalieit.html

[22] "Cooperative Learning Response to Diversity". retrieved 5/27/03, from http://www.cde.ca.gov/iasa/cooplrng2.htm.