Perspectives on Active Learning and Collaboration: JavaWIDE in the Classroom

Jam Jenkins
Valdosta State University
Valdosta, GA, USA
cjjenkins@valdosta.edu

Evelyn Brannock
Georgia Gwinnett College
Gwinnett, GA, USA
ebrannoc@ggc.edu

Thomas Cooper
The Walker School
Marietta, GA, USA
thomas.cooper@thewalkerschool.org

Sonal Dekhane
Georgia Gwinnett College
Gwinnett, GA, USA
sdekhane@ggc.edu

Mark Hall
University of Wisconsin - Marathon County, WI, USA
mark.hall@uwc.edu

Michael Nguyen
Emory University
Atlanta, GA, USA
michael.nguyen@emory.edu

ABSTRACT
The Java Wiki Integrated Development Environment (JavaWIDE) is an innovative environment that promotes active learning and collaboration in programming courses. This paper surveys how JavaWIDE’s features have been used to promote active and collaborative learning in both traditional and distance education (synchronous) in four different environments: high school, summer enrichment courses, and at two- and four-year colleges. The authors describe the context of each teaching and learning environment and the parts of JavaWIDE that are particularly well suited in each context. After discussing the active learning and collaboration techniques employed, student responses to the experience are summarized. This collection of case studies illustrates how the concurrent editing, shared environment awareness and other features of JavaWIDE can be used to promote active learning and collaboration within a heterogeneous set of teaching and learning environments.

Categories and Subject Descriptors
D.2.6 [Software]: Programming Environments – integrated environments, interactive environments.

General Terms
Design, Experimentation, Human Factors, Languages.

Keywords
JavaWIDE, concurrent editing, distance education, tutoring, web-based IDE, collaboration, active learning, case study, Java, CS0, CS1, introductory programming.

1. INTRODUCTION
Active learning and cooperative learning have been used successfully in introductory programming courses to promote retention and pass rates [1-3]. Examples of using these techniques include using enhanced clickers for providing instantaneous feedback, incorporating short student exercises within the lecture, and using short in-class exercises such as Think-Pair-Share, pair programming, group projects and others [4-9].

The Java Wiki Integrated Development Environment (JavaWIDE) is designed to support the active classroom and collaboration by supporting synchronous group programming via concurrent editing and shared environment awareness. While tools exist for supporting group programming [10-13], these tools have not been widely adopted at the introductory programming level.

JavaWIDE simplifies the adoption process by providing a zero-install web-based environment that scales well. There are currently over 120 JavaWIDE sites for schools from around the world, all supported by a single server and site administrator. This paper describes how and why JavaWIDE has been used at four different levels (high school, summer camp, two-year and four-year colleges) and in a combination of traditional co-located classrooms and in distance education. These four different use cases drawn from actual experience demonstrate how JavaWIDE can support promoting active learning and collaboration in a heterogeneous set of environments.

2. JAVA WIDE OVERVIEW
The Java Wiki Integrated Development Environment (JavaWIDE) is a web-based development environment built for collaboration and ease of use. The initial version of JavaWIDE was created in 2007 in an effort to eliminate the need for installing an IDE while including the collaborative features of a wiki. Since this time JavaWIDE has been enhanced to include many commonly desired features such as code completion, syntax highlighting, automatic indentation, code formatting, auto-import, code refactoring, integrated Java API documentation and others. In addition, the web-based interface allows the inclusion of other features uncommon to development environments including the automatic posting of all source code and executable programs, integration with social networking sites, and a shared codebase (allows everyone to see, use and modify code on the system).

Perhaps the most innovative features are the integration of concurrent editing and shared environment awareness. Using the follow feature it is possible for one user to automatically open all of the files another user currently has open. As the “followed”
user subsequently opens additional files, these same files open automatically within the “follower’s” view of files. Moreover, as each user edits the file, the updates appear immediately on all users who have the same file open. Users do not need to take turns editing – everyone can edit whenever and wherever they would like. This feature is called concurrent editing. All of the cases that follow made extensive use of the concurrent editing and the shared environment awareness (the follow feature).

The sections that follow are use cases describing actual experiences of using JavaWIDE to promote active learning and collaboration. Each use case is composed of three subsections. The first subsection will describe the educational atmosphere in which JavaWIDE was used. These vary by level, length of instruction and whether the course was in the traditional classroom, delivered entirely online through synchronous distance education, or some combination of these two. The second subsection describes the type of active learning and collaboration activities included within the course and how JavaWIDE supported them. The third subsection in each use case gives quantitative and qualitative feedback from students about the activities and JavaWIDE itself.

3. HIGH SCHOOL PERSPECTIVE

3.1 Environment for using JavaWIDE

Many high school AP Computer Science classes are small, sometimes with less than 10 students. One of the experiences that AP students should have is that of collaborating with others and completing team-based work. This is obviously difficult with a limited number of students. In spring 2010, seven high schools participated in a project called the Computer Science Collaborative [14]. The purpose of this project was to implement and assess innovative techniques to allow collaboration and teamwork at a distance.

The project needed a technological infrastructure that could allow the geographically distributed groups to collaborate in as natural an environment as possible. This included at a minimum two-way group voice communication and data sharing, in addition to the ability of collaboratively editing code. Elluminate, a webinar system, was used to provide the voice communications, an interactive whiteboard, and recording capability. Several options were available for sharing and editing the code.

JavaWIDE was selected for its ability to support collaborative code editing. It also reduced the need for version control because files were stored online in a central location. Moreover, JavaWIDE allowed us to participate in synchronous lessons. These programming lessons were truly collaborative rather than just cooperative. They allowed students and teachers to participate in developing algorithms at the same time and more easily help each other. The marriage of a webinar system and JavaWIDE provided a way for students to communicate with each other, to collaborate on programming in real time, and for us to document the process for research purposes.

3.2 Active Learning & Collaboration

During the project, students participated in several synchronous distance education lessons. The instructors used JavaWIDE to make the lessons interactive. In our cyber-icebreaker, students used turtle graphics to draw lines, make the pen lift up and down, and change the color of the pen. Many students attempted to draw their initials. Students were then placed into breakout rooms and asked to collaborate on writing basic methods to draw a shape or a word related to a computer science principle. During the time students were becoming familiar with using the webinar system and JavaWIDE, instructors monitored students’ progress by using JavaWIDE’s follow feature. Using this and the concurrent editing feature, each student’s work could be monitored at a distance.

Over the course of the project, students participated in programming increasingly difficult fractals found in natural system using recursive algorithms. These methods were applied to four different fractals. In the first case students were given a lecture on recursive methods, provided the pseudo-code for a fractal tree and shown how to code the fractal tree in JavaWIDE. In each subsequent case, students were given less information in an effort to promote active discussion, problem solving and algorithm development. The other lessons on fractals over the course of this project included programming a Koch Curve, which students were asked to extend to create a coastline fractal and a Sierpinski Triangle, which students were asked to extend to create a sea coral or sponge-like structure. In the final project, student groups were asked to apply what they learned in the online large group sessions to create a fractal of their own design. Recorded sessions, presentations and documents for the project can be found on the project website.

Between each large group session, student groups were asked to meet online to collaborate on programming the fractal that was discussed in the large group session and to extend it to an environmental or biological system. Each group consisted of 5-7 students, one from each participating school. In two instances, groups had 2 students from the same school. Students were divided into groups based on their computational thinking skills and previous programming experience.

In each online small group meeting, the instructor-mentor was in charge of taking attendance and making sure that the session was recorded. The peer leader was in charge of conducting the session, while the instructor monitored the meetings. The peer leaders were instructed to use a form of the Think-Pair-Share (TPS) model to facilitate the development of each algorithm [15]. Student groups posted their solutions to their site in JavaWIDE and shared their solution with other groups at the next online large group meeting.

3.3 Student Response

During the course of the collaborative project students were asked to keep a reflection log. In addition, teachers provided written feedback at the end of the project. Below are excerpts from students’ logs and from teachers who participated in this project. These responses indicate a very positive experience collaborating using JavaWIDE and that some minor tweaks are needed in its interface.

Students

“The tools have been very helpful [JavaWIDE, Elluminate, Google Docs]. We were all able to write code in one class on JavaWide and I shared my desktop and talked through questions with my teammates while we were writing code.”

“JavaWide is a ridiculously cool idea. Being able to have our whole group work on the same file is very useful.”

Teachers

“JavaWide is great with the ability to share code and work at the same time without a program on the computer. The lag issues are minor but somewhat annoying.”
“I really like the potential of having students editing one file together in one session; I have already begun to use it in class. Its file management system is not intuitive for such things as saving multiple versions of a file or moving a file from one place to another.”

4. ENRICHMENT PERSPECTIVE

4.1 Environment for using JavaWIDE

In summer 2011, a one-week summer enrichment course was offered, entitled the Cooperative Online Game Programming Course. The purpose of this course was two-fold: first to prepare AP teachers to deliver engaging synchronous online distance education, and second to improve the preparation of and motivate students entering AP Computer Science. During the first week the course was offered to an AP teacher and teaching assistant. During the second week the teacher and assistant offered the course to a group of 14 students entering AP Computer Science.

By recognizing motivation as a primary problem of online learning, the Cooperative Online Game Programming Course (COGPC) used a pedagogical model that emphasized online presence of the instructor, related learning to students’ personal goals, and fostered student collaboration. JavaWIDE facilitated the implementation of this model in several key ways.

First, the group editing part of JavaWIDE allowed the instructors to design activities promoting student interdependence. This interdependence helped build the online community in the brief period of the single week in which the course was delivered entirely at a distance.

Second, JavaWIDE eliminated a large amount of technical issues by providing a tool that does not need to be installed or configured, and works across web browsers and operating systems.

Third, the mode of communication in JavaWIDE between instructor and students enabled the instructor to be more accessible and responsive to students, and increased the perceived quality of the class and the effectiveness of the instructor. The features used in this communication included the ‘follow’ feature and the concurrent editing.

4.2 Active Learning & Collaboration

The integration of JavaWIDE into the COGPC allowed for the following active learning techniques to be utilized:

Leading question: Questions were framed to guide the students questioned to respond with a particular obvious answer as expressed in generating Java code. These answers were then explored in further depth which ultimately concluded with answers that were not so explicit, fostering cognitive dissonance, and/or self-questioning on the part of the students.

Brainstorming: This method of collaborative problem solving involved all members of the class spontaneously contributing code as ideas are generated.

Student-created art and video game: Students were encouraged to create abstract art and a video game in groups as contexts for extending their understanding of key course-specific concepts. This sort of exercise encouraged students to ask each other: What would a good game/program look like? How should the art/game be created? What are the strengths and weaknesses of various approaches? JavaWIDE provided an excellent resource for this sort of learning.

Problem-based learning: The COGPC enrichment course utilized this pedagogical strategy based on constructivist learning theory that simultaneously develops both problem solving strategies and disciplinary knowledge bases and skills by placing students in the active role of problem solvers.

Student creative construction: Based on a set of criteria that were determined in the course and informed by course materials, students created an abstract art piece and a video game. Students worked collaboratively producing an artistic product. In the process of creative expression, students learned basic programming skills.

4.3 Student Response

Before and after the course all students were asked to rate their level of agreement (1 is strongly disagree, 5 is strongly agree) for questions related to using JavaWIDE, interest and preparation for the Advanced Placement Computer Science course, and perceptions on group work. Out of 14 the students in the course, 13 agreed to participate in the study. All but one student (12) answered questions in the pre-survey, and all 13 answered questions in the post-survey.

The numbers following the questions indicate the average level of agreement before and after the course.

1. I know how to use JavaWIDE. (1.25, 4.54)
2. I feel confident that I will do well in the Advanced Placement Computer Science course this fall. (4.33, 4.46)
3. I am excited about taking Advanced Placement Computer Science this fall. (4.58, 4.62)
4. I feel well prepared to take Advanced Placement Computer Science. (3.58, 4.62)
5. Working with groups has benefits over working individually. (3.83, 4.15)
6. I can learn more by working with others than I can by working alone. (3.75, 4.31)
7. Working in groups is often frustrating. (2.83, 2.08)
8. Some people in a group always end up doing all of the work while others hardly do anything. (3.50, 2.85)
9. On computer related projects, I prefer working in groups over working individually. (3.58, 3.85)
10. I am interested in game programming and collaborative team programming. (4.17, 4.08)

Overall the student responses are positive and indicate students had a positive first experience with collaborative programming. Particularly, the changes in questions 5-9 all show an improved perception of group work after taking the course.

Below are some of the comments from students after the course:

“It was not what i was expecting at all. I enjoyed it a lot more than I expected to and my partner was a great help”

“It made complicated programs easy to do.”

“I really enjoyed the class, and it has opened my eyes to an enjoyable career path.”

“I had a lot of fun participating in this program. I learned a lot and I hope to keep in touch with all the instructors involved.”
“I learned a lot about how to program. Especially on how to program in Java. I learned how to cooperate with people in programming and I got to experience programming with others. Very fun experience.”

“It was a fun and informative course. I would have preferred more classes or more time.”

5. TWO-YEAR COLLEGE
5.1 Environment for using JavaWIDE
The thirteen 2-year University of Wisconsin (UW) Colleges serve students over the entire state of Wisconsin [16]. The computer science (CS) program in the UW Colleges is designed to be easily transferable to the 4-year institutions within the UW System. The nationwide decline in CS enrollments reported in the 2003-2004 Taulbee Survey [17] has affected the UW System, and has been particularly hard on the 2-year UW – Colleges (UWC), which are relatively small campuses, with a combined enrollment of 12,452 students.

In addition, 70% of our students are in the bottom 50% of their high school rank. The majority of our students enroll to prove that they can achieve at the University level in order to be accepted at a 4-year University. Currently, CS2 and CS3 are only offered via distance education once per year for all campuses.

Seven of the campuses have enrollment so low in computer science that we had to offer a distance education CS1 course to support the smaller campuses. Each of the distance education campuses has between 1-5 students enrolled in the lecture component of the CS1 course. These students face isolation fears due to low number of students available to discuss problems and to an instructor who is not on the same campus.

The UW-Colleges provide funding for the CS1 DE (distance education) lecturer while each campus had to hire a local lab instructor. Some campuses had difficulty hiring a local Java instructor or had financial issues with paying an instructor for small enrollments. The computer science department firmly believes that CS1 students need the face to face presence for the CS1 labs.

During spring 2011, these problems forced two campuses with combined CS1 enrollment of 8 students to drop the CS curriculum from their campus. The UW-Colleges administration along with the computer science department chair made the decision that the CS1 DE course must be able to offer not only the lecture via DE but also the labs for spring 2012. The CS department insisted on having final approval on exactly how the CS1 labs would be offered and still emphasized the necessity of maintaining the ability of DE students to show code and ask questions of the lab instructor. These DE labs must also take into consideration the type of students that would take this course. Retention of CS1 students is very critical for CS2 and CS3 when you start with small enrollments in CS1.

As I looked at the guidelines established by the UW Colleges for teaching the general concepts for the CS1 course, the emphasis that course coverage for designing algorithms should take at least a minimum of six hours. And the reality is that every programming assignment typically forces students to design a totally different algorithm. In order to be successful in this course, for the students and the professor, somehow, someway, the critical thinking skills of students have to be improved. Improvements in critical thinking along with the understanding of the basic control structures will allow students to be successful writing simple programs using any language. It is obvious that more time on task can be very helpful for developing these critical thinking skills.

5.2 Active Learning & Collaboration
In early January, 2011, I was first introduced to JavaWIDE. Very quickly it became obvious that this tool would be perfect for dealing with programming issues for DE students. There wasn’t enough time to get familiar with the IDE and implement this feature into my spring 2011 CS1 DE course as part of the syllabus. So about 4 weeks into the semester, I started offering optional programming sessions picking random assignments chosen by myself or students, usually from the back of the textbook. These assignments would mimic the upcoming lab assignment. We would meet on Monday and Wednesday using JavaWIDE from 5-7 pm and students could join one or both sessions. Attendance varied as attendance ranged from five to fourteen students, with certain students attending both sessions.

We used D2L Online rooms (Elluminate) so we could use the VoIP feature for students who had microphones and headphones. Other students would use the chat feature of JavaWIDE. We would start using the whiteboard feature to discuss the objects in the assignment to create UML diagrams for these objects. Then we moved to JavaWIDE to start programming. I allowed students to pick which methods for the object that they wanted to program, and for those students that did not get methods, they were assigned certain methods to watch and help fix any syntax errors. At first, this approach seemed very chaotic as control was lost to the students in the development process. If students struggled, they could ask questions, and at times, I asked for all programming to stop so we could discuss the issue. Students would provide the answers and then “chaotic programming” would start again.

If there were questions, students could ask out loud but some students preferred using chat to ask the question. I would verbally tell the group what the question was and ask for responses. Inevitably students would provide the correct response. If we started using a Java API that we had not discussed in class, I told the students to google to find out the answer. They would copy and paste into JavaWIDE their ideas and we would discuss them. This approach seemed to open doors for students to ask questions if they were not sure about any part of the code. At the end of the first pass at writing the code for the object, coding would stop and we would do “code inspections” starting at the top to the bottom and ask questions to see if the code makes sense as a method and as part of the object.

It was very interesting using this approach as I felt at times more of a conductor than a teacher as we wrote this piece of music (algorithm). After two hours though, I was mentally drained trying to keep up with student’s ideas and questions. Since I worked in the commercial world prior to becoming a teacher, I firmly believe in introducing good programming practices and style early into the semester so students could understand these practices and be able to write “good code”, or as I sometimes label it “pretty programming.” As we went on, students who would be watching students write a method would start modifying the code to enforce these good practices like descriptive variable names. And if there was a dispute, we would discuss different variable names and then vote which one we would use. As part of my teaching, I tell students they are NOT writing code for
themsevles, they are writing code for the next programmer, and that programmer was usually me.

At times, we would take the previous week’s source code and add new features to the code that were being discussed in the lectures so that we would not have to start over from scratch. This approach showed students that the software development process is cyclical and that “good code” can be made even better as new knowledge is gained.

Using group assignments or peer-programming can be beneficial for at-risk students to increase their success rate. However, the remote distances between students make this near impossible.

5.3 Student Response
JavaWIDE allowed the development of a different type of lab. Instead of having traditional closed labs and students work individually on their assignments, we will use this 2 hour lab to develop a different programming assignment but one which is similar to the current lab. At the end of the lab, students are allowed to keep the electronic copy of the in-class lab source code. Instead of having one lab time for the DE students, multiple sections will be offered to keep student numbers lower for each session of the in-class labs. This will allow each student to have more participation in each session. There will be a pre-lab assignment where students will have to “discover” the objects and methods in each lab assignment and these assignments must be turned in prior to the lab. The assessment rubric for these pre-labs will be very low-key as they will get a “Exceed expectations” (bonus points), “Meets expectations” (passing grade) and “Does not meet expectations” grade. This will be part of the overall lab grade for their assignments.

“JavaWide was very fun, the interactivity demanded you to know your stuff, and if you didn't you were shown how. I feel that JavaWide was more valuable than the slideshows, but the slideshows taught much needed vocabulary at the same time, so it is hard to say which is the most important.”

“This course was very good overall and as a first timer to programming I surely learned quite a lot. I think less time should be used on the lectures and more time should be spent on using the JavaWide program. I learned basically all of my coding skills from JavaWide and it helped me get practice in and it helped me see what things I was doing wrong. JavaWide is more hands on which is better for the learning experience overall in my opinion. I wouldn't know as much as I do today if it wasn't for the use of JavaWide. The program should be used more next semester and I think it should be mandatory.”

“The use of JavaWide was also very helpful in learning how to program.”

6. FOUR-YEAR COLLEGE

6.1 Environment for using JavaWIDE
Georgia Gwinnett College (GGC) is an open access, four-year, liberal arts college. GGC requires two ITEC courses in their core curriculum of all graduating students. The first is an Introduction to Computing and the students may choose their second from Digital Media and Introduction to Programming. Therefore, Introduction to Programming is a general education course that contains many non-majors in the enrollment.

Because many students are not majors, and owning a computer is not a requirement, finding access to properly configured and installed computers on which students can program is a challenge. Instead of offering individual technical support to students and asking for installed software in labs and school computers, it was easier to use JavaWIDE since the interface is entirely browser based (i.e. requires no installation). As long as the students have an Internet connection available and a browser they have the capability to code, compile and test. JavaWIDE supports Google Chrome, Firefox, IE, and Safari browsers and can be run on the Mac, PC and Linux operating systems. The no installation, easy to use environment is in bold contrast to many of the other client desktop IDEs, such as Eclipse and Dr. Java that require the correct installation of the JDK and the IDE software. JavaWIDE requires no requests to the Office of Educational Technology, making the life of the instructor and the support personnel’s lives easier!

Many students find object oriented coding paradigms difficult to grasp at first, and the non-linear execution of the code tends to be confusing. Many times students will follow what has happened in class, but then find they cannot reproduce the results on their own at home. Using JavaWIDE, they find it very helpful that they can view all of the coding completed in class, without having to download it. JavaWIDE documents all revisions and keeps track of who changes what and when. This feature also aids the instructor if cheating is suspected, as all changes are stamped with a user name.

6.2 Active Learning & Collaboration
Active learning begins on day one when the instructor employs JavaWIDE. On the first day of class students actually produce their very first program and they are encouraged to share this program with friends or family via JavaWIDE’s integrated social networking capability. This provides immediate positive feedback and gratification on the very first day of class.

The fact that students can concurrently edit the same source file together, much like in Google Docs, in class supports an extremely interactive environment. Students can pose a problem, then they work on a solution in concert. The professor is not the driver of the keyboard, but the facilitator and motivator.

Another active learning technique used is to talk about the problem, break it up into workable pieces (i.e. methods) and assign groups of students the different pieces in class. This is especially useful when teaching the concepts of parameters and method invocation; these concepts become imminently clear. The concurrent editing also allows the instructor the offers the students remote tutoring.

6.3 Student Response
To evaluate the effectiveness of JavaWIDE in the classroom students were recruited to participate in end of course surveys. 30 students participated in the survey in Fall 2009, 15 in Fall 2010 and 44 in Spring 2011. All 100% of the students participating in the surveys said that they had a computer and access to the Internet at home. This result is significant in that JavaWIDE requires students to have access to a browser and an Internet connection, which all of them have readily available at home.

In spring 2011, we asked students to choose from two different options given specific scenarios:

- 2% preferred to download the programs and then use them in a stand-alone, desktop IDE instead of accessing the program code on the Internet.
- 7% preferred to take notes in class, while 55% preferred to access the step-by-step revisions of the program on-line.
• 38\% reported equal satisfaction with both in-person tutoring and online tutoring (a feature offered by JavaWIDE).
• 58\% preferred the revision-tracking feature of JavaWIDE instead of having to manually do version control in many of the novice programming IDEs.
• 62\% preferred the concurrent editing feature, allowing students and instructor to solve problems together, to other IDEs that do not support collaboration in such a fashion.

In summary, the student responses to features of JavaWIDE and its usefulness in learning programming were positive. When given various scenarios and options to choose from, the students invariably picked JavaWIDE features over features of other IDEs.

7. CONCLUSION
The high school, summer enrichment, and two-year and four-year colleges comprise a heterogeneous set of environments, all with different needs but with the same desire to include active learning and collaboration in the introductory programming experience. JavaWIDE was selected as one of the tools that would support this type of learning while also meeting the other goals and restraints of the course/project. In each environment the ability to use JavaWIDE without installation or configuration eased adoption.

JavaWIDE supported a wide range of collaboration and active learning activities, primarily by using the concurrent editing and the follow feature. These allowed instructors to monitor student progress on individual and small group work, allowed students to participate in creating example code, and to work together.

The student responses to the active learning and collaboration were largely quite positive. Students enjoyed being able to work together and share code easily. In student comments during and after the course, students enjoyed receiving help, not only from their teachers but also their classmates.

8. ACKNOWLEDGMENTS
Our thanks to all of our students who agreed to participate in the surveys before, during and after our courses, and to the John & Mary Franklin Foundation for supporting the Cooperative Online Game Programming Course.

9. REFERENCES