# The active classroom: students and instructors parallel programming... in parallel

Nasser Giacaman, Simar Kalra and Oliver Sinnen
*Parallel and Reconfigurable Computing group*
*Department of Electrical and Computer Engineering*
*The University of Auckland, New Zealand*

***Abstract*—The biggest difficulty that students face when learning programming is in developing the necessary cognitive skills that allows them to apply what they have learnt. It is generally accepted that programming is one of those things that can only be learnt by doing and actively engaging with it. Parallel programming is a prime example of a programming area that students commonly struggle with. A major inhibitor is due to some of its abstract concepts, making it difficult to grasp a true understanding of the underlying principles in a traditional classroom setting. This paper discusses the underlying principles that motivated the development of Active Classroom Programmer (ACP), a tool for students to learn effective programming strategies with the guidance of their instructor. ACP aims to increase students skills in applying programming topics, by immediately engaging them with the newly introduced material. This is especially important in parallel programming, as the topics quickly progress onto the many parallelisation caveats (such as thread-safety, race conditions, and so on). While laboratory or homework exercises provide students with valuable hands-on experience (to apply newly taught concepts), this opportunity generally arrives too late after the material is presented in the lesson. To address this, a collection of parallel programming exercises are being developed for the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing (as an Early Adopter award), with the help of ACP. Instructors are welcome to utilise any of the developed exercises, or even request a private ACP account for their own courses to program with their students.**

## I. Introduction

Software is increasingly becoming a more integral aspect of our daily life, especially as the uptake of personal mobile devices continues to increase. The shortage of skilled developers in the software development industry continues, and is only exasperated as software companies continue to expand. Programming is the one dominant and essential skill in developing software. The difficulties of teaching and learning programming is nothing new. It is generally accepted that students struggle mostly with the higher level aspects of programming (rather than details such as syntax) [1]:

- Understanding how to design the program to solve a problem,
- Debugging their own programs.

These aspects of programming demand the development of cognitive skills that cannot be taught in the traditional classroom [2]. In the case of novice programmers, those that have developed their programming strategy tend to learn more effectively than those that have not developed a strategic approach [3]. This is particularly important for software engineering students, whose engineering discipline emphasises an inductive approach to problem solving [4].

An active learning approach is essential in motivating programming students to be more engaged [5]. The traditional classroom provides insufficient opportunity to develop the necessary cognitive skills in programming, as it lacks the relevant learning activities [2]. A "gap filling" exercise is a pedagogically sound way to help students build on their existing knowledge to solve problems, but is difficult for students to do alone and demands close support from teachers [2]. Debugging practice also promotes cognitive programming skills [6], and it too cannot be learnt in the traditional classroom setting (of sitting and listening to the teacher).

In recognition that practice is essential, programming laboratory sessions (labs) are the typical opportunity in which students develop these cognitive programming skills. The usual model is to hold practical lab sessions once a week, normally a few days following the classroom lesson pertaining to that topic. Albeit having good intentions, and by all means being beneficial to students grasping concepts, a number of learning deficits are inherent in this model:

- The delay between the lesson in which the material is presented and the associated practical lab is too long. Short-term working memory only holds information for 20 minutes; if students are not given the opportunity to relate previous knowledge with the new information within these 20 minutes, the information is lost [7]. Students need to interact with the new material almost immediately, otherwise it will be lost quickly [8].
- The frequency of the labs tend to be (at best) once a week, and usually only cover a subset of the material taught in the lessons so far. By the time the lab session arrives, more material had been thrown at the students before they had an opportunity to reinforce the earlier material. This exasperates the previous problem since new material is continuously building on unfamiliar material.
- It is essential to have a sufficient number of teaching assistants that circulate the room answering student questions, otherwise progress will be severely stalled. Some students may feel emotionally unsatisfied if their peers are receiving an unfair amount of support and attention compared to themselves. Merely increasing the number of teaching assistants is not only a matter of financial difficulty (i.e. limitations of the course budget), but also

in terms of finding teaching assistants with the required expertise and skills.

This motivates the necessity for a more interactive programming classroom (i.e. students code in the lesson). Not only may this help long-term memory retention [7], but it allows students to capitalise on the teacher's expertise. Promoting a social atmosphere in the classroom by working together is also valuable, where students should not feel disadvantaged. Peer acceptance and classroom belongingness aids academic self-efficacy and achievement, as well as student motivation [9]. Students should be exposed to learning from others without seeing other students as a threat to their own learning (which might be the case when one student takes more of the teaching assistant's time than others).

This paper discusses a more collegial model of active learning for the programming classroom by encouraging an open-learning environment. In addition to promoting good learning practices, it also seeks to promote a sense of belongingness amongst students. It builds on sound industry practices, namely pair programming, where each and every student is (loosely) paired up to program *with* the teacher. The guidance provided by the teacher in the problem-solving, as well as the debugging training, helps develop the cognitive skills necessary for programming. This is especially valuable for parallel programming, since debugging in front of students is highly beneficial to illustrate parallelisation caveat concepts such as race conditions. The Active Classroom Programmer (ACP) is structured in a way that allows it to be a non-intrusive intervention for both instructors and students to embrace, by providing students with an opportunity to continue (spontaneous) coding snippets presented by the teacher.

As multi-core processors become the norm in ubiquitous devices, parallel programming has become an evermore important programming field. Learning parallel programming is challenging, let alone being applied correctly and efficiently, so the TCPP Curriculum Initiative on Parallel and Distributed Computing (PDC) [10] was released to guide instructors in their curriculum design of parallel and distributed computing courses. To help promote and further develop this Curriculum Initiative, the Early Adopter Program sought proposals that implement it. One such proposal was the development of a collection of parallel programming exercises meeting the most essential PDC topics in the Curriculum Initiative.

This paper serves two purposes. First, it discusses the underlying principles that motivated ACP in general. Second, it directs the reader (whether a student or teacher) to the repository of PDC exercises dedicated to addressing topics in the Curriculum Initiative, identified as being most essential due to students being able to "apply" (according to Bloom's taxonomy [11]) those concepts in practice.

Section II discusses features that would be useful in an active learning environment for parallel programming courses. The ACP philosophy and features are highlighted in section III, while directing the reader to the ACP repository dedicated to PDC topics in section IV. Section V gives student perspectives on using ACP, before concluding in section VI.

## II. Exploring desired features for parallel programming with students

A number of technologies have emerged that help in promoting an active learning approach for programming. Without attempting to be complete, this section discusses the general idea behind many of these approaches. While they have merits in their own right, they leave more to be desired before being suitable for programming classrooms. In particular, we focus on parallel programming as being a more advanced and specific topic (as opposed to common material taught in CS1 courses). Some of the valuable features in a classroom tool would be:

- The instructor should have the ability to create code spontaneously (e.g. if a student asks a question and the instructor wishes to illustrate with an example), and not be constrained to exercises prepared beforehand.
- The examples (whether premeditated or spontaneous) should be transferable immediately to the student's programming development environment seamlessly.
- Allow an exercise to be "progressed" in stages, with snapshots (as determined by the instructor) of the intermediate steps recorded so that students may replay in their own study time outside of the classroom.
- The student and instructor should have independent copies of the code, so no one is "blocked" from coding (which is the case with collaboration plugins).
- The tool should be non-intrusive and easy to use (no need for understanding or using additional tools, such as version control, since this distracts and complicates it for novice programmers).

### A. Interactive coding websites

Combining the reachability of the world wide web and the effectiveness of interactive learning, various interactive coding websites [12], [13] have emerged. These websites have gained tremendous popularity as they allow students to follow lessons that include exercises within the website. Added benefits include allowing students to learn in their own time and without setting up a development environment (such as downloading/installing an integrated development environment and software development kits).

These technologies are a nice way to teach the basics of a programming language (such as syntax) to a student. The websites tend to include predefined lessons formulated to conduct a course. Understandably, it makes sense that the lessons include topics that meet the interest of a larger population. As such, none of these lessons include advanced concepts such as parallel programming. The websites are supplementary to the classroom (students follow them in their own time separately from the classroom) and only target general CS1-like topics. The lack of an integrated development environment (IDE) further limits the potential of using them for more complex topics such as parallel programming.

In terms of addressing the particular learning outcomes of a course, something more flexible is necessary. What is necessary is the ability of an instructor to create customised exercises that specifically target the learning outcomes of the

curriculum, ensuring a constructive alignment approach [14] to teaching. Also missing is the human expert, the instructor, who is needed to guide the students in the programming strategy and to adapt the exercises as student queries are encountered.

### B. Collaboration plugins

The next category of technologies recognise that a real IDE is desired, in addition to the value of having an instructor (such as the lecturer) that can customise the exercises for a more focused and effective learning experience. They are collaboration tools [15], [16] that come in the form of IDE plugins. The tools tend to be used for remote team programming, where programmers are physically separated from each other but need to collaborate on the same code base. The real-time qualities of these tools allow one programmer to drive the coding, while observers watch as their local IDE updates with the modifications instantaneously. In the case of the classroom, this real-time aspect is likely to be a distraction as students look down on their own IDE rather than fixing their attention to the instructor's explanations.

The collaborative tools tend to be useful for *peer* programming, where the programmers are equally contributing to the same code base. Some of the plugins provide a read-only option for the observers, which would be necessary in a classroom usage to avoid some students modifying the instructor's coding. However, to achieve active learning, the students will need to do coding at some stage in the exercise; this is not possible unless independent and local copies of the code is maintained for the students. Ultimately, these collaborative tools were not developed with the classroom setting in mind, and are usually limited in the number of concurrent connections. While good for following code in real-time, there is no versioning or snapshot ability that would be helpful for students to replay the interactivity in their own time. Such a feature would be essential, since rehearsing and reviewing material helps students remember [8].

### C. Software version control

In a software development process, software control systems (such as Git and Subversion) are typically used to manage the changes of a program over time. This allows multiple versions (snapshots) of the same code base to be recorded, with an ability to alternate between the different versions. These concepts are valuable not only in the software industry, but also in an educational sense. By being able to record the various stages of a program during its development, students would be able to reenact the steps. This is especially powerful when combined with real code, as it provides practical and active learning. The disadvantage though, is that using a standalone software versioning system would be a distraction in the classroom. This is especially true for novice programmers who do not (need to) know anything about version control, such as dealing with repositories and reverting to previous revisions). More importantly, using additional external tools create additional "overhead" steps that distract from the learning process.

## III. ACP: its philosophy and features

The technologies in section II helped identify some of the desired features that would help reinforce student learning in programming courses. Since no single solution had all the required features, ACP[1] was developed (contact info is contained on the website if you wish to consider it for your own course). Before presenting ACP features, the classroom setting that ACP promotes is explored to illustrate its philosophy compared to other programming classroom environments.

### A. Rethinking the learning environment

This section explores the progression of the learning environment for programming courses, and the benefits each may bring to developing the cognitive programming skills in students.

*Traditional classroom lessons:* The most traditional model of teaching includes the "sage on the stage" [17], where the teacher transmits knowledge in one direction with the expectation that students sit and absorb it. In most cases, there is no active learning occurring. Active learning occurs when students are engaged during the lesson with something requiring them to manipulate the new information to increase their understanding of the material [18]. There is overwhelming support for active learning [19], especially for programming courses as they tend to be rather conceptual and abstract [20]. One should of course not forget the expertise and knowledge of the teacher; the question then, is the teacher's presence in the classroom being optimised to help students learn?

*Traditional laboratory sessions:* One of the most applicable and relevant form of active learning for programming students is to do predefined coding exercises. The traditional model is to hold weekly lab sessions where students complete such exercises within the allocated times while seated at their own workstation. Students have an opportunity to fully engage with the material, while teaching assistants (usually not the official teacher, but senior students) circulate the room answering student queries as hands are raised. By the time these labs take place, students are likely to have already forgotten the material covered in lectures [7], [8]. On top of that, the low ratio of teaching assistants to students increases the chances of confused students going idle without help for prolonged periods. The students progress individually through the lab session – not collectively as a group.

From the student's point of view, there is not enough help from the teaching assistants. Even when a teaching assistant is helping, they may feel pressured to move on to the next student (especially when a large number of students are confused). The teaching assistant barely has enough time to help demystify a query, and is unlikely to fully satisfy the confused student as they rush their explanations. The teaching assistant's time is not utilised effectively, especially if they are addressing the same doubts repeatedly on an individual basis to multiple students. Some students may feel emotionally disappointed if they did not receive as much support or attention as their peers, only worsening their feeling of confused loneliness. Any shy

[1] https://acp.foe.auckland.ac.nz

students, or those that had no time to ask questions, get left behind and feel further upset and frustrated. The lab session suddenly turns into a fight for the teaching assistants, possibly with feelings of resentment towards their peers.

*Pair programming laboratory sessions:* A simple way to mitigate some of the concerns of traditional laboratory sessions includes pairing students together in completing the exercises. This effectively reduces the pressure on teaching assistants, as students are able to solve many queries together [20], [21]. The elegance of this solution is especially appealing for programming courses, since *pair programming* is a proven industry practice [22], so this provides students with an opportunity to work with others. The pair programming philosophy is based on one person playing the role of the *driver* (doing the coding), while the second plays the role of the *observer* (guiding and reviewing the code written by the driver); to be effective, the roles are alternated frequently. The danger of students pair programming relates to incompatible personality pairings, as well as some students favouring one of the driver or observer roles without alternating [20], [21]. By being paired with fellow peers, students still do not have the guidance that would be most effective should they be paired with someone more experienced (such as the teacher).

*Studio-based lessons:* The primary problem with the traditional laboratory approach is that it is combined with the traditional classroom lesson. More recent learning models include replacing the standard lectures with lab sessions, known as the studio-based model [23]. This has the heightened benefit that students are immediately active during lessons and overall improves their programming skills [24]. While effective, the studio-based approach largely relies on a redesigned course outline with the development of new material suitable for the studio classes; in addition, it requires an increase in the number of lecturers and teaching assistants to help conduct lessons [25].

*The Active Classroom Programmer model: bridging the classroom and laboratory:* The ACP model aims to combine many of the benefits of the above models together. Most important is that of promoting active learning, which is achieved by requiring students to partake in actual coding exercises within a fully-featured IDE. However, rather than having predefined exercises (as in laboratories and studio-based lessons), the exercises are spontaneous and developed gradually during the lesson with the teacher's guidance. This allows for "on the spot" creation of exercises, not only customised for the particular needs of the course, but also for any particular queries that may arise during the lesson.

This model differs from the studio-based model in that lessons remain the standard classroom lessons; there is still only one instructor (no additional teaching assistants required) teaching much the same way (e.g. using presentation slides, etc), and it is optional for students to bring along a laptop (i.e. the coding exercises are not compulsory). These points are crucial, since it allows for a non-intrusive intervention that allows both the instructor and students to gradually accustom to. For students opting not to bring a laptop to class, they still watch the exercises on the screen (focus on the instructor), or pair with a neighboring student, or do the exercise on paper (as if it is an exam practice exercise).

The idea is that the teacher covers the "theoretical" contents in much the same way they would normally teach (using presentation slides, etc). At some point in the lesson, the teacher decides to spontaneously create a code snippet to help reinforce the material (either preempted from a student question, or somewhat premeditated). The snippet is purposely left incomplete, and is uploaded (via the instructor's IDE) to allow students to instantaneously download it (via the student's IDE) to immediately complete the snippet. These uploads are automatically recorded as snapshots of the current project, allowing the student to reinforce the material at a later stage.

The benefits of using this model include:

- Students receive guidance on the *programming strategy*. Through the process of encountering errors and debugging them in a strategic manner, the students are guided through the thought process by using an inductive teaching approach. The teacher is careful to give attention only to the important points (whereas weaker students may dwell on the irrelevant details when working alone and lose sight of the "bigger picture").
- The atmosphere of the classroom becomes *collegial*. All students receive equal attention from the instructor. If any student asks a question, all will hear it and its answer (as opposed to the lab setting, where the same question may be repeated multiple times). In some cases the question may be a "good one" that other students may not have thought of, yet they benefit from hearing its answer. There is no student that disadvantages other students by having an unfair amount of the teaching assistant's time (as in the lab/studio-based model). This creates an open-learning environment, with all students feeling they are given an equal chance to engage and ask questions.
- Students are working using the *standard IDE* that they use in their day-to-day work. This helps them set up easily, encouraging them to be more proactive outside of the classroom. Students feel assured they are off to a good start, since they build on the same code base from the teacher. Rather than feel they are coding on their own solution, the lesson theme becomes "let's all be active on the same code together".
- The benefits of *pair programming* are promoted, except now the students are (somewhat) *paired with* the teacher. Of course, this is not pair programming per se, but students play an active role of *observer* (while the teacher is coding, students watch and ask questions) while also be given opportunities to be the *driver*. This provides a form of scaffolding for the students, as they play these pair programming roles with someone knowledgeable.
- Another advantage, from the financial point of view of the institution, is that this teaching model is more cost-effective. In some cases, it is simply not possible to have a high number of teaching assistants due to budget limitations (many international universities pay research students to be teaching assistants, as it is not an obligation of their research). This, of course, should not be the primary reason to use this model – but it is a welcomed bonus nonetheless.

### B. ACP features

While section III-A discussed the philosophy behind ACP, this section will focus on the particular functions and features that have been implemented. ACP is an Eclipse plugin specifically designed to motivate student-teacher interaction in a programming class. As such, it is intended to be easy to use and very minimal to avoid distraction from learning.

The idea behind ACP is that the teacher will create an example during the class, which is then uploaded. The student will then be able to access that example and work on it themselves during the lesson (as well as after the lesson). ACP extends the Eclipse IDE by introducing a few new menu entries. ACP offers the following key functionalities to the users, some of which are restricted for teachers only.

*User accounts:* ACP requires a user account. The user must create a new user account through the ACP website to be used inside the plugin. There are two types of user accounts, student and teacher. The user account determines whether certain functionalities are available to the user, as discussed below. To support multiple institutions, registering a new user involves using a signup code specific to that institute. This allows instructors maintain their own repository of projects.

*Upload project (teachers only):* The teacher is able to upload a project inside the Eclipse environment to be shared with the students. This is a very simple procedure, where the teacher simply right clicks on a project and chooses the "Upload Project" option. A simple dialog box allows the teacher to choose a folder to place the project in. The project is immediately uploaded to the server, including any libraries contained within the project directory.

*Download project:* A user (either student or teacher) may then download a project that has been uploaded by a teacher. This is accessed via the ACP menu, using the "Download Project" option. A dialog box is used to display all the current projects available. The plugin downloads the selected project to the computer and imports it into the workspace seamlessly, ready for the user to immediately run.

*Sync project:* ACP allows teachers to upload new versions of an existing project for the students (where each version can be given a meaningful tag). The goal behind this feature is for the teacher to progress through an exercise all the while uploading different versions of the same project. The students would then be able to progress through one exercise effortlessly by switching between the different versions of the project (figure 1). In the context of classroom usage, the students will sync to the latest version of the project as the teacher adds a new version. This is similar to switching between different revisions in a version control software, expect it has been made very simple through ACP. This is especially important when students are not familiar with using version control systems, as well as the unnecessary distraction they would introduce to the entire workflow.

*Version diff:* Since each project contains multiple snapshots (i.e. the sync uploads made by the instructor), a diff compare feature is integrated so that students can see modifications of source code compared to another version of that same project. This works on a file basis (i.e. diff is applied to a particular file). Figure 2 shows the version diff of file `Counter.java`, showing how a Lock was used to solve the race condition.

*Admin controls (teachers only):* Teachers have a few extra options available to them for managing the uploaded projects. The four options available are Delete Project, Delete Version, Hide/Unhide Version, and Hide/Unhide Project. Teachers also have access to an admin control panel on the website with similar functions as well as the ability to delete users and make them a teacher. Unlike the collaboration plugins, ACP enforces this control to provide teachers with the power in maintaining student attention during the lesson. From the student point of view, the limited functionality available to them reduces distraction during the lesson.

## IV. ACP supporting the PDC Curriculum Initiative

As part of an Early Adopter award, a collection of exercises relevant to core parallel programming topics identified in the PDC Curriculum Initiative [10] are being developed. Although they will include a collection of predefined exercises (unlike the spontaneous usage of ACP discussed in section III), they will nonetheless be useful for parallel programming students and instructors alike. The exercises are accompanied by a website[2] explaining the contained ACP projects and their alignment to the Curriculum Initiative. Instructions also include how to seamlessly access and run the projects directly through Eclipse. The projects focus on those identified as expecting students to "apply" (according to Bloom's taxonomy of learning levels [11]).

## V. Student perception on their learning

This section discusses various forms of feedback from students that have been exposed to the learning environment that ACP promotes. All student feedback has been in the form of anonymous questionnaires or summative lecturer teaching evaluations. The primary focus discussed here is on student perception in the guidance they receive on the programming strategy, in addition to their impression of the overall effectiveness of helping them grasp particular concepts.

### A. Guidance on programming strategy

One of the aims of using ACP is that students are able to develop their cognitive programming skills by learning from the instructor good programming strategies that is difficult to learn on their own. When asked "**What was most helpful for your learning?**", the following statement emphasises the power of programming in front of students by promoting inductive learning:

> *"The live examples. The practical examples of using the Eclipse/Android IDE in class, as well as the kinds of explanations and discussions held in class. Errors and mistakes made were helpful in the sense that we can not only see how they were caused, but the thought process behind the coding that caused the error, the consequence, and the workaround/fix to correct the error"*

[2]https://acp.foe.auckland.ac.nz/pdc

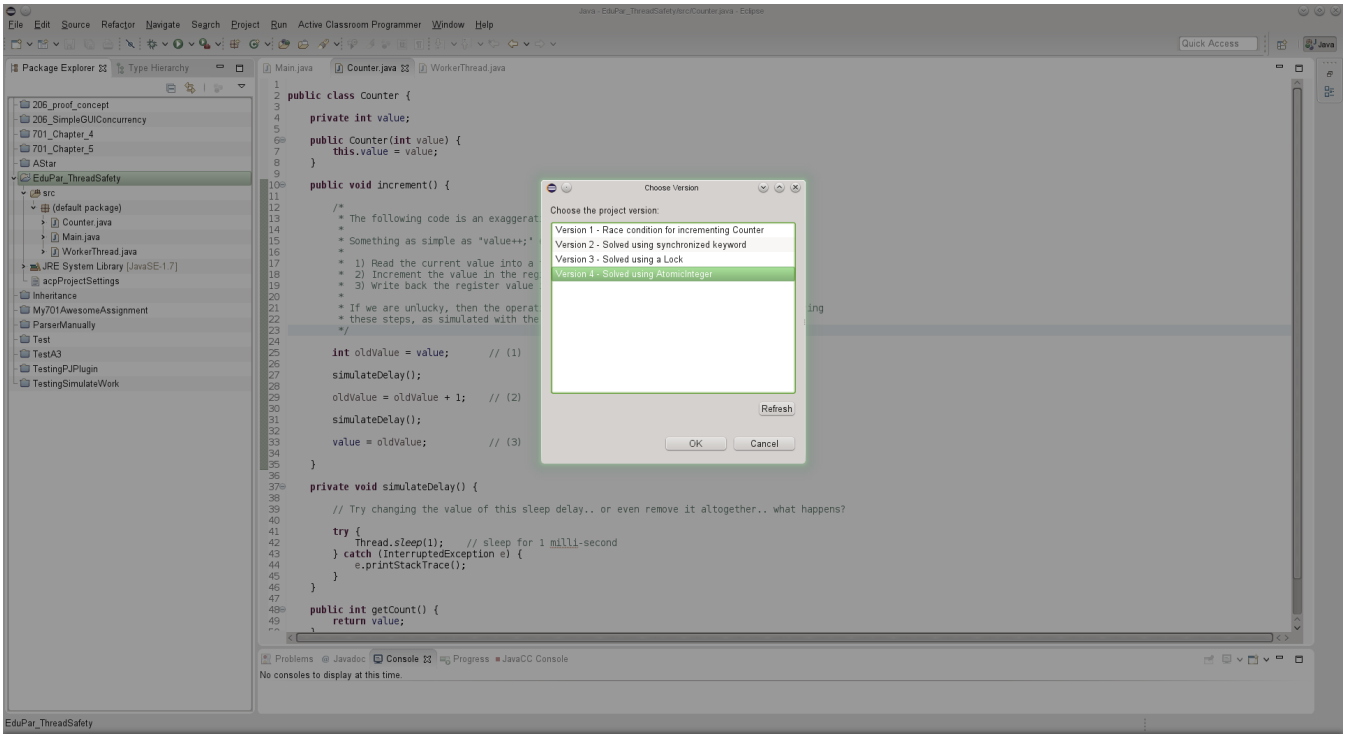


Figure 1. The ACP plugin running in Eclipse. The existence of the plugin is non-distracting. Interactions include right-clicking on an existing project (from the left menu) to sync between versions, or to download a new project altogether (from the Active Classroom Programmer menu at the top). Here, the user is using the Sync project feature to select one of the solutions for this simple race condition exercise.

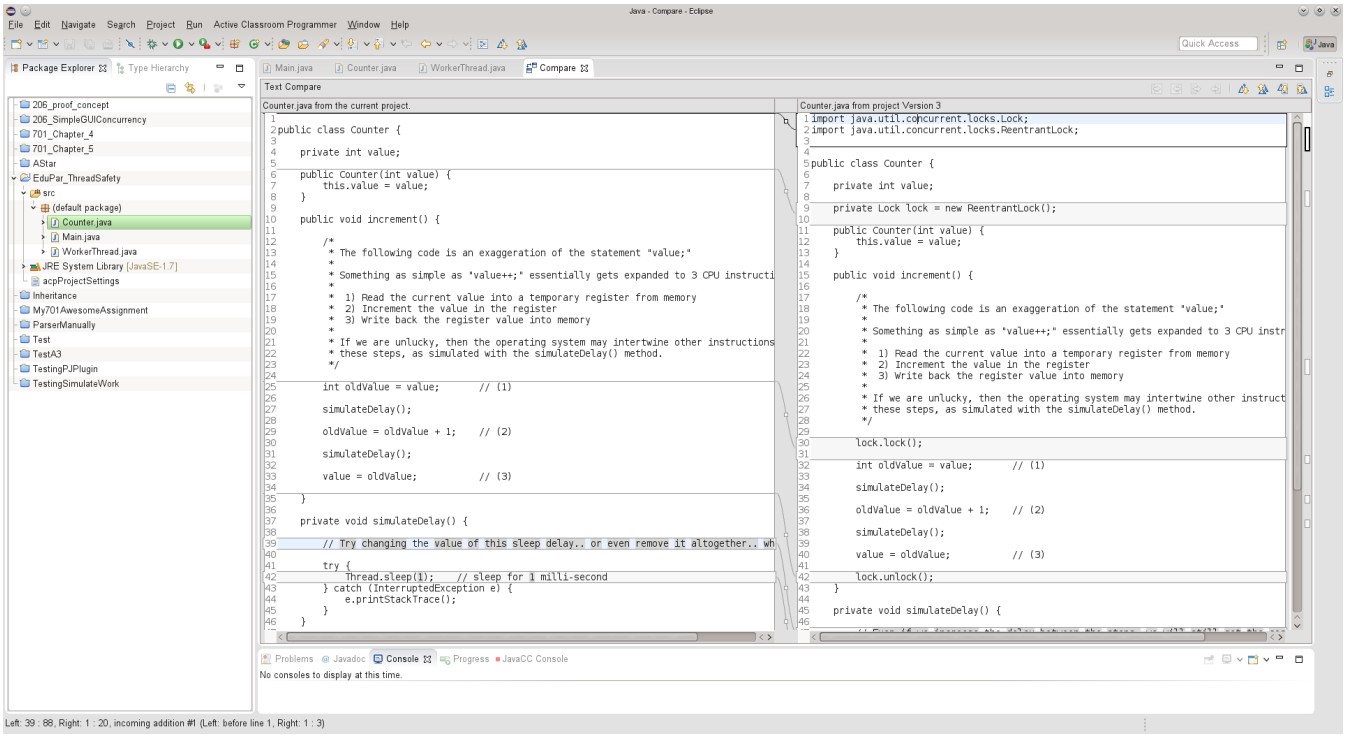


Figure 2. Here the user has selected to view the differences between two snapshots of a project. In this example, the current version is version 1 on the left (with the race condition), while version 3 (solved using a Lock) is on the right.

This is especially important in learning parallel programming, since debugging is a vital skill when it comes to ensuring thread-safe and efficient parallel code. Many exercises can be developed in conjunction with the students, during the lesson, where students begin to see the value of debugging and how it could be done. Another student highlights that instructors should not underestimate the appreciation students feel for the programming guidance when programming in front of them:

> *"I enjoyed the coding sessions in class. It is really helpful because a lot of the content we learn is only helpful when we actually code it ourselves but seeing the lecturer code in class gives us a better understanding of what you are talking about and how to actually implement it"*

### B. Grasping of course material

When asked "**In what way, if any, do you feel the tool assisted your learning?**", a lot of students commented on how ACP helped them grasp the theoretical concepts:

> *"Helped deepen my understanding and grasping of concepts"*

The versioning feature of ACP becomes a form of scaffolding that assists the grasping of concepts by allowing students to gradually progress through the project code in their own time:

> *"Better understanding of theory that was taught. Exercise examples are really good, especially step by step versioning is helpful"*

The following students emphasis how ACP allowed them to better understand the concepts by being active:

> *"Look through parts and fiddle/modify the code to see differences and understand what happened"*

> *"Being able to run through examples in my own time outside lectures so I can grasp/understand concepts better"*

> *"It helped me understand theory through a practical approach"*

## VI. Conclusions

In order to retain more of newly presented material, students need to interact with the material within 20 minutes of it being presented. In programming courses, both lectures and labs are equally as important to the learning process; students need the lecturer to explain new material, while they also need a lab-like environment to immediately practice the new concepts. Active Classroom Programmer combines the lectures and labs into one and supports a fluid and flexible approach to creating meaningful on-the-fly exercises for students to instantaneously attempt. This is especially important for abstract and difficult programming topics, such as parallel programming. To help instructors of parallel and distributed computing courses, a collection of projects are available targeting core topics in the PDC Curriculum Initiative. The accompanying website helps students and instructors work through the projects, hosted on an ACP repository dedicated for this initiative. Alternatively, instructors may create their own projects in the ACP framework and utilise it for their own courses.

## Acknowledgment

We would like to thank the NSF/TCPP CDER Center Early Adopter Awards. Their support funded the development of ACP repository of exercises for the PDC Curriculum Initiative.

## References

[1] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," *SIGCSE Bull.*, vol. 37, pp. 14–18, June 2005.

[2] W.-Y. Hwang, C.-Y. Wang, G.-J. Hwang, Y.-M. Huang, and S. Huang, "A web-based programming learning environment to support cognitive development," *Interacting with Computers*, vol. 20, no. 6, pp. 524 – 534, 2008.

[3] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003.

[4] M. Prince and R. Felder, "Inductive teaching and learning methods: Definitions, comparisons, and research bases," *Journal of engineering education*, vol. 95, no. 2, pp. 123–138, 2006.

[5] T. Jenkins, "Teaching programming – a journey from teacher to motivator," in *The 2nd Annual Conference of the LSTN Center for Information and Computer Science*, 2001.

[6] G. C. Lee and J. C. Wu, "Debug it: A debugging practicing system," *Computers & Education*, vol. 32, no. 2, pp. 165 – 179, 1999.

[7] J. Willis, "Review of research: Brain-based teaching strategies for improving students' memory, learning, and test-taking success," *Childhood Education*, vol. 83, no. 5, pp. 310–315, 2007.

[8] M. Sprenger, *How to teach so students remember*. Association for Supervision & Curriculum Development, 2005.

[9] A. A. Mouratidis and G. D. Sideridis, "On social achievement goals: Their relations with peer acceptance, classroom belongingness, and perceptions of loneliness," *The Journal of Experimental Education*, vol. 77, no. 3, pp. 285–308, 2009.

[10] "NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing." http://cs.gsu.edu/~tcpp/curriculum/?q=home, 2010.

[11] L. W. Anderson and D. R. Krathwohl, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Pearson, 2000.

[12] Codecademy, "Learn to code." http://www.codecademy.com, 2014.

[13] Code School, "Learn by doing." https://www.codeschool.com, 2014.

[14] J. Biggs, "Enhancing teaching through constructive alignment," *Higher Education*, vol. 32, no. 3, pp. 347–364, 1996.

[15] IDEtalk, "Collaboration tool for IntelliJ IDEA Java IDE." http://www.idetalk.com, 2014.

[16] Saros, "Real-time distributed software development." http://www.saros-project.org, 2014.

[17] A. King, "From sage on the stage to guide on the side," *College Teaching*, vol. 41, no. 1, pp. 30–35, 1993.

[18] C. Bonwell and J. Eison, *Active learning: Creating excitement in the classroom*. School of Education and Human Development, George Washington University Washington, DC, 1991.

[19] M. Prince, "Does active learning work? a review of the research," *Journal of Engineering Education*, vol. 93, no. 3, pp. 223–231, 2004.

[20] K. J. Whittington, "Infusing active learning into introductory programming courses," *J. Comput. Sci. Coll.*, vol. 19, pp. 249–259, May 2004.

[21] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik, "Improving the CS1 experience with pair programming," *SIGCSE Bull.*, vol. 35, pp. 359–362, Jan. 2003.

[22] L. Williams, R. Jeffries, R. R. Kessler, and W. Cunningham, "Strengthening the case for pair programming," *IEEE software*, vol. 17, no. 4, pp. 19–25, 2000.

[23] C. D. Hundhausen, N. H. Narayanan, and M. E. Crosby, "Exploring studio-based instructional models for computing education," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, (New York, NY, USA), pp. 392–396, ACM, 2008.

[24] D. Hendrix, L. Myneni, H. Narayanan, and M. Ross, "Implementing studio-based learning in CS2," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, (New York, NY, USA), pp. 505–509, ACM, 2010.

[25] M. Barak, J. Harward, G. Kocur, and S. Lerman, "Transforming an introductory programming course: From lectures to active learning via wireless laptops," *Journal of Science Education and Technology*, vol. 16, no. 4, pp. 325–336, 2007.