Approaches for Coordinating eTextbooks, Online Programming Practice, Automated Grading, and More into One Course

Margaret Ellis, Clifford A. Shaffer, and Stephen H. Edwards
Virginia Tech
Blacksburg, VA
maellis1|shaffer|s.edwards@vt.edu

ABSTRACT

We share approaches for coordinating the use of many online educational tools within a CS2 course, including an eTextbook, automated grading system, programming practice website, diagramming tool, and debugger. These work with other commonly used tools such as a response system, forum, version control system, and our learning management system. We describe a number of approaches to deal with the potential negative effects of adopting so many tools. To improve student success we scaffold tool use by staging the addition of tools and by introducing individual tools in phases, we test tool assignments before student use, and we adapt tool use based on student feedback and performance. We streamline course management by consulting mentors who have used the tools before, starting small with room to grow, and choosing tools that simplify student account and grade management across multiple tools.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**;

KEYWORDS

tool use, CS2, instruction, course management

ACM Reference Format:
Margaret Ellis, Clifford A. Shaffer, and Stephen H. Edwards. 2019. Approaches for Coordinating eTextbooks, Online Programming Practice, Automated Grading, and More into One Course. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287487

1 INTRODUCTION

With continuing advances in CS Education Research, there is an ever-growing collection of high-quality online educational tools available to instructors for use in their courses. In this experience report we share our experience integrating several such tools into a CS2 course, and describe approaches that others can use to manage the process of adopting many such tools in an education setting. We describe the context of our course and our experiences with using WebCat [1], OpenDSA[2], CodeWorkout[3], and Lucid Chart [4], all in conjunction with more typical educational tools such as Canvas (our LMS), Eclipse and iClickers. We want students to take advantage of available tools that allow them to explore, practice, tinker, and build professional skills. Yet we do not want to overwhelm either the students or the instructional team. The overarching approaches we used can be grouped into two main categories: 1) improving instruction and student learning, and 2) streamlining course management for the instructor. To improve instruction and student learning we use scaffolding to phase in the use of multiple tools and for learning individual tools incrementally. We test tool-based assignments before student use to verify clarity and correctness, gauge difficulty, and assess the amount of time necessary to complete the assignment. We also modify our use of individual tools based on student experiences and feedback.

Our approaches to streamline course management for instructors help save time and reduce the risk of error. When integrating multiple tools into this course we have benefited from consulting mentors who have used a given tool before. We also start small with room to grow in the use of more complex tools, and we prefer to choose tools that simplify student account and grade management across multiple tools. We will report on our specific experiences and within that we describe some guiding principles that can be applied generally in various instructional settings.

2 BACKGROUND

Student success has been linked to student self-efficacy, and targeting instruction to improve student confidence with a specific task can improve student persistence [10]. A key objective of our CS2 course design is to improve student self-efficacy, so our approach to integrating and using tools is done with this in mind. Four key areas of influence impact self-efficacy: vicarious experiences, mastery experiences, social persuasion, and physiological response [2]. Tools that provide accessible practice to students can help improve their confidence in the practice area with mastery experiences, particularly when the lessons are sequenced from simple to more complex and students are able to monitor their understanding. Tools that provide accurate non-negative feedback can maintain students' confidence by social persuasion. If students' frustration with challenging concepts can be minimized while they have a satisfying experience then the physiological response positively influences self-efficacy [3]. In a study of CS majors, students reported that they gained confidence by tinkering outside of class, and a personal

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02…$15.00
https://doi.org/10.1145/3287324.3287487

[1] web-cat.org
[2] opendsa.org
[3] codeworkout.cs.vt.edu
[4] www.lucidchart.com

factor that predicts interest in Computer Science is the willingness to engage in constant practice [1].

Practice with a tool can help build student self-efficacy with that tool and improve student success in a course. Furthermore, building student confidence at learning new tools could have a broad impact by increasing students' initiative and willingness to tinker with and explore tools throughout their career.

Some tools are specifically designed to enhance learning or automate course management (e.g., CodeWorkout, OpenDSA, Web-CAT, iClicker, Piazza), and others are designed to complete a task that is expected of computing professionals (diagramming tools, IDEs). Previous research and reports discuss the use of online coding tools, visualization tools, eTextbooks, auto grading tools, student response tools, and online forums. Many instructors believe that online coding tools reduce the dropout rates in introductory programming classes [20], and these tools have been shown to be useful in helping students master syntax of programming languages [13]. There are claims that coding tools increase the self-confidence of students in Computer Science [4, 15], while visualization is useful for novices and programmers in general [11, 14]. It is widely accepted in Computer Science education research that automating processes, such as grading, can save resources and also seems to overall positively impact academic performance [7, 21]. If used thoughtfully, it is also commonly accepted that response systems such as iClickers and online forums such as piazza also improve student learning [6, 12].

Despite the benefits of using the aforementioned educational tools, some instructors still do not take advantage of tools in their courses for a variety of reasons. Educators need to feel a sense of control for successful adoption of a tool. Training and tutorials that recognize operational and pedagogical difficulties that can arise help reduce educators' anxiety [16, 17]. It can be difficult for educators to find guidance regarding effective use of educational tools such as algorithm visualization, but experiences and lessons learned from others can save time and allow educators to make better choices [19]. Educators are less willing to adopt a system unless its usefulness has been demonstrated. But even though the usefulness of online coding tools has received attention from researchers, use of such tools is still not widespread by educators [5]. Researchers recognize time as the major impediment to instructors' adoption of visualization tools which is further exacerbated by course integration issues [18]. The next wave of challenges for instructors who are already using tools is determining how many and what tools to use, and how to effectively integrate them. We share our approaches to help other instructors expand and coordinate their tool use with a greater sense of control and confidence so they can improve student learning and reduce administrative work.

3 COURSE DESCRIPTION

Our CS2 course covers an introduction to data structures, efficiency, and Object Oriented design with Java as the programming language. There are typically two lecture sections of 100-300 students who have a variety of backgrounds. Students have obtained credit for a prerequisite introductory programming course, either in our CS software engineering sequence, at a community college or another 4-year institution, or by obtaining AP Computer Science A credit.

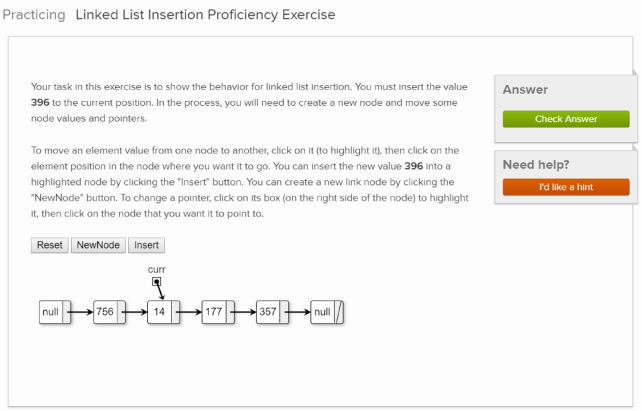

Figure 1: Example of an OpenDSA interactive exercise.

Students who took the prerequisite programming course at our institution come to our course familiar with unit testing and our style and documentation standards. They will have already used two of the tools discussed here: Web-CAT (for managing submissions of programming projects and auto-grading via unit testing) and CodeWorkout (which auto-grades small programming exercises). However, a significant fraction of the students in the course have not used these tools, are not familiar with our style and documentation standards, and have never written unit tests or used an auto-grader system. Our CS2 course is taught using Java, and there are typically a few students whose only prior programming course used a different programming language.

Students attend two traditional lecture sections and a two-hour lab each week. Labs and programming projects are submitted to Web-CAT for feedback and grading. Our current institutional Learning Management System (LMS) is Canvas. In addition to lab and projects, students have programming practice assignments in CodeWorkout and OpenDSA. CodeWorkout is used on tests for coding questions. OpenDSA is an interactive eTextbook which is customizable but already has significant content available for data structures. Students are assigned OpenDSA reading and problems throughout the semester, and they find the visualizations and practice useful. An example exercise is shown in Figure 1. Students also have design assignments for which they use a diagramming tool of their choosing. We use Piazza heavily as the course forum, and we use iClickers for in-lecture active learning.

In 2015 we restructured the course content in order to synchronize lab activities, lecture material and progressively more involved projects. We incorporated scaffolding throughout the course. Students are given pre-reading assignments and quizzes in preparation for lecture, and the labs are broken into pre-labs, labs and post-labs. We include iClicker questions in the lecture material. When we redesigned the course we continued to use Web-CAT but wrote new tests and hints for reference tests focusing on expected behavior instead of specific methods, with a goal of keeping the messages constructive and positive.

Table 1: Tools used in CS2

Tool Name	Purpose	Introduced	Integrates with
Canvas	Learning Management System(LMS)	week 1	
Web-CAT	auto-grading and feedback system	week 1	Canvas
iClicker	active learning response system	week 2	Canvas and spreadsheets
OpenDSA	eTextBook with algorithm and data structure visualization	week 5	Separate Canvas Course
Lucidchart	optional online collaborative diagramming tool	week 5	Google Docs, pdf
Eclipse Debugger	Debugger	week 5	Web-CAT
CodeWorkout	online coding practice and assessment	week 6	Canvas

4 LESSONS LEARNED

4.1 Approaches for Student Success

Students start with varying backgrounds, so the initial weeks can be intimidating and there can be a steeper learning curve for some students than for others. Scaffolding tool use helps alleviate the initial challenges. We scaffold tool use by starting with small introductory activities, including demonstrations in lecture and with videos, and eventually expect students to use tools in context. Pro-actively testing tools and corresponding instructions with consideration for difficulty and approximate time required improves student success and minimizes unanticipated frustrations. Assessing student feedback and performance allows us to make adaptations to tool use, activity directions, and overall course instruction.

4.1.1 Scaffold tool use. Our approach to integrating multiple tools is to phase them in throughout the semester, and to provide instruction for each when introduced. At the beginning of the course students are trying to learn the course routines, CS concepts, and new tools. To keep them from feeling intimidated and overwhelmed we phase in tools over time. We make the learning curve for individual tools more gradual by exposing students to tools at increasing levels of detail and difficulty over time. We attempt to lessen the cognitive load on students so that they are not simultaneously trying to master a new concept and a new tool. By scaffolding the introduction of tools and the learning of specific tools we give students the opportunity for mastery experiences that are sequenced from simple to more complex to build their confidence.

4.1.2 Add tools to the course in stages. In Table 1 we list tools used in the course and when they are introduced in order to ease students into the semester. In the first week of the course students start the routines of going to lab and submitting to Web-CAT. They also begin weekly pre-reading assignments with quizzes on Canvas. In the second week we expect students to use their iClickers in class, and the forum becomes more active. After a few weeks, students are comfortable with the overall course management and tools. Many of them need this time to adjust to programming expectations reinforced by Web-CAT, which includes style and unit test coverage. This gives them the opportunity to practice and gain confidence in this software development setting.

After several weeks, when students are accustomed to the course, we demonstrate the debugger. Students also have their first assignments using a diagramming tool and their eTextbook. In the following week they have on-line programming practice to help prepare them for the upcoming testing experience.

4.1.3 Introduce specific tools with increasing levels of difficulty. To lessen the cognitive load on students, we scaffold the assignments that use the tools. This eases the students into using the tool. We use a variety of techniques such as having a small assignment with the specific objective of familiarizing students with the tool, modeling use of the tool in class, and having assignments with increasingly involved use of the tool. The first day that we introduce using iClickers is a practice day, and we do not count the points so as to ease into use and make sure the logistics are correct. These smaller experiences set the students up for success and allow them to get positive feedback as they build skills.

In the week when students use the Web-CAT project autograder for the first time, we provide detailed lab instructions for doing Web-CAT submissions. The corresponding post-lab activity is an exploration of the Web-CAT results in order to further familiarize students with the tool. This eases students into using Web-CAT. Subsequent Web-CAT lab and project instructions are less detailed.

The first semester that we used the OpenDSA eTextbook, we did not initially demonstrate it in lecture. From forum and office hours interactions, we realized that students were getting stuck on those questions that require problem solving, particularly about algorithm analysis and recursion. Students actually needed help with course skills and concepts more than tool use. When we demonstrated OpenDSA problems in lecture, we found that students were more comfortable after they watched us solve just a couple problems. Providing this vicarious experience for them builds their confidence in solving these types of problem. As a result they start the homework assignments sooner and ask fewer questions. Similarly for CodeWorkout, students can use the tool and understand the feedback but might need help with some of the Computer Science context. Most CodeWorkout questions require students to write member methods for a class. This is disorienting for some students until they see a solved example. After that, they are more comfortable and successful solving future questions.

Many students do not begin the semester debugging their code with a source-line debugger. We notice that students are intimidated by the eclipse debugger until they have one-to-one coaching in office hours, at which point they value it and become proficient with it. In the first semester that we used the debugger, we did introduce it in lecture and lab, but then we realized we needed a lab dedicated solely to that topic. We have since also added more early exposure to the debugger. By week 5 we have demonstrated using the debugger in lecture to see the contents of a linked structure. In the post-lab the following week, students further explore the lab code using the debugger and are given a corresponding quiz. This

increases self-efficacy with the debugger around the time that their programming projects are becoming larger. In Week 9 we have a lab dedicated to debugging. The students find this frustrating, but more of them begin using the debugger, which we observe helps them in the projects.

The final project requires students to work in a group to iteratively design and implement a program. For each of the four programming projects that precede the final group project, the students are given design guidelines. In preparation for creating their own design with a UML diagram, students are initially asked to create a simple UML diagram of existing code containing only two Java classes. This experience helps familiarize them with using the diagramming tool when the diagram difficulty is low. We also provide our own brief video tutorials, and we vet and link to existing tutorials. Lab and project instructions provide UML diagrams as examples, and the tool is not phased in until Week 4, with the complexity of its use increasing over the semester. This is an example of phasing in a tool over time, so students are comfortable with it by the time that they use it for their more high-stakes final project.

4.1.4 Test tool assignments with consideration for clarity, difficulty and timing. Working through assignment instructions well before they are assigned is valuable because it improves the instructions and overall assignment. Actually completing an assignment allows instructors to verify that students are set up for success with clear directions, pages with working links and images, the necessary pre-requisite skills, and the appropriate time-frame to complete the assignment. Pro-actively testing instructions reduces student questions and increases their comfort, and saves time in the long run for the instructional staff. Having other instructors or teaching assistants test the assignment is ideal, but minimally the author themself can work back through the instructions with fresh eyes well after it was drafted.

As part of our lab restructuring, we broke each into pre-lab, lab, and post-lab. In the first semester we had the TAs work through the labs in advance using a rubric such as the one shown in Figure 2. As detailed in the checklist, we test the instructions on Canvas, the down-loadable code skeleton, and the Web-CAT configuration. We reduce confusion in lab when the Canvas instructions are clear and complete. Assignment testing also uncovers potential issues with our Web-CAT reference tests and feedback messages. Our expectations about student performance and the amount of time required to complete the lab is improved by having someone time themselves to complete the assignment. With assignment testing the resulting instructions are clearer, and adjusted expectations are more reasonable. This reduces student anxiety and frustration while still challenging them.

Each semester we have different assistants solve our CodeWorkout questions, and we verify that the timing, difficulty and instructions are appropriate. Even after multiple iterations, we still discover ways that we can improve instructions or how we write our unit tests. Here are examples of feedback that we gained from testing our instructions:

"The only problem I encountered is for union, the instructions didn't say you can't edit the original bags, but CodeWorkout won't let you pass if you modify the original bags."

Checklist for Lab Review

1. Review Pre-lab and solutions, check if well connected to lab
2. Read lab instructions
   a. **verify all links work**
   b. **verify images work**
   c. **take a careful look at code blocks and check formatting**
3. Make sure skeleton formatting is reasonable
4. Verify skeleton downloads correctly
5. Solve the lab yourself
6. Review Solution against yours (we can reuse former solution or your new one)
7. Submit Solution(s) to WebCAT
   a. check for reasonable error feedback on WebCAT
   b. check deadline times for each lab
8. Review Lab Notes from last year regarding the lab and make it usable for team
9. Review Post-lab and solutions
   a. check if well connected to lab
   b. check deadline times
10. Send feedback to me and communicate/work back and forth
11. Report out to the team (by Sunday)
12. Make sure google drive directories have latest skeleton and solution, rename any former one with _prev. Both the Skeletons and Solutions directory

Figure 2: Example checklist for testing lab instructions

"I was of the opinion that when the second bag is null, the bag returned should be a new bag with all the elements in the current bag, rather than just returning a reference to the current bag, but if that's not a requirement then it shouldn't create any confusion for students since both approaches work."

Testing CodeWorkout questions helps us anticipate multiple solution approaches and how we might need to improve our unit tests. We have also found that we need to be mindful of what methods the students need to know in order to solve the problem (e.g. `String substring(int, int)`, `double pow(double, double)`, `int parseInt(String)`). Testing helps us to flesh this out and decide what additional information students might need.

We have found that integrating tools requires additional time spent by the instructional staff on configuring tools and testing the assignments up front in order for students to have a smooth experience. Fortunately, the tools that we use have also successfully reduced the time spent grading by our TAs and instructors, so there is more time available for configuring and testing. Even with the additional time spent in preparation, the end result is more time available for student contact.

4.1.5 Consider student input and experience. Watching and talking to students about using tools enlightens us and debunks preconceived notions about their experience with, and perception of, the tools. This allows us to make adjustments that improve student learning.

By watching students and looking at their submission history, we could see that many students were submitting to Web-CAT heavily to receive feedback while not thinking through their submissions thoroughly on their own. These students were using Web-CAT feedback to make guesses until they had full code coverage for their own tests, and likewise making small changes until they pass all of our tests, seemingly without regard for the bigger picture of the assignment or overall test cases for the problem. One of

the goals of an auto-grading system that requires student tests and provides feedback is to promote reflection-in-action, however student patterns of over-submitting are well documented [9] [8]. An adjustment that we made was to configure the use of "submission energy", which is a mechanism to limit the frequency with which students can submit to Web-CAT. This causes students to think through the problem more thoroughly and learn more from the assignment.

When we began updating this course we knew we wanted to increase student engagement. Some of our faculty had formerly experienced frustration with response systems, and others did not want to require students to make additional purchases. By talking with students, we discovered that because most of them also take Physics, they already have iClickers. So our use of iClickers was adapted to mirror how our colleagues in Physics use them. This simplified our ramp up, and was convenient for students because it was already a familiar tool. Paying attention to students and talking to them allowed us to come up with this solution.

4.2 Approaches for Instructor Success

Instructor time is valuable. If instructors can spend less time on administrative work then they have more time to spend with students and improving curriculum. We try to streamline the use of multiple tools in one course in order to save time and reduce errors. We consult mentors to head off difficulties, and we start small to ease into teaching with a new tool. We pick tools that integrate well with each other, particularly for managing grades.

4.2.1 Consult a mentor who has used the tool before. Finding a mentor who has used a given tool is valuable for avoiding pitfalls and quickly resolving logistical issues. Having a mentor is especially helpful for guidance on how to handle absences, late work, and how to accommodate students with special services needs.

When we changed the lab structure and set up all new projects for the course, it was advantageous to have a mentor for using Web-CAT. It made us more efficient in setting up configurations for assignments and in writing tests. Experienced users could provide guidance not just on syntax for test messages, but various approaches on how and when to give students feedback on failed tests. Seeing examples of a mentor's previous configurations helped to accommodate late submissions with automatic point deductions. For a tool as complicated as Web-CAT, it is ideal to build community and corporate knowledge from semester to semester in the form of experienced TAs training new TAs, and having the experienced TAs write training manuals for new TAs who will have to pick up being tool administrators.

Having a mentor for CodeWorkout was useful for realizing what previous content was already available. It was also important for logistical nuisances such as knowing to use the same name for the assignment in Canvas and CodeWorkout, and being sure to synchronize settings and time constraints across course sections. A mentor was also useful for demonstrating how to provide alternate test times and extended time. The begin and end dates can be adjusted and so can the time alloted. Having a mentor saved time and allowed us to anticipate and resolve logistical issues.

When we decided to use iClickers, we consulted with our colleagues in Physics. We had a meeting to discuss their approaches and steps for setting up iClickers across our institution. They showed us how they installed and configured the software and also explained their grading scheme and course guidelines. When they described some of their management efforts with grading and especially dealing with absences, we chose to take a simpler overall approach. We followed their advice to require students to only use the iClickers to respond and not their phones. Their guidance helped us head off logistical issues and made getting started with iClickers run smoothly.

4.2.2 Start small with room to grow. We avoid investing significant time and effort on a tool or corresponding assignments until we are confident it will yield positive learning experiences for our students. We will not really know this until we try the tool, so we start with a small number of assignments and keep the grading value of the assignments low. It is easier to start with a tool that provides some ready-made content, this allows you to initially spend your efforts on integrating the tool versus creating content. If the tool is also customizable then we improve and enhance our approaches either throughout the semester or from semester to semester. As discussed above, it is useful to test any ready-made content before assigning it to students.

The first semester that we used CodeWorkout, we began with pre-existing questions about arrays. CodeWorkout was familiar to some students who had used it in our CS1 course the previous semester. CodeWorkout was not familar to all the students, and it had not been used before in the CS2 course, so we eased into it. We created a couple of practice assignments with new data structure content that the students did for low point homework, and then used it in an exam. The following semester we added working an assignment during lecture. In the future we plan to create more CS2 content to use for active learning during lectures.

We do not make tool activities worth many points and we do not over-promise how we will use or grade activities with the tool. For example, there are many OpenDSA questions, but they are worth only 2% of the students' overall grade. However, they have great value to the students as practice. One semester our server running OpenDSA restarted overnight, and students who completed the assignment before the server crashed lost their grades, while students who did the assignment after the crash were graded normally. So we follow a two-level approach here. In the long term, we want to fix the underlying problem before the next semester (or work with the tool developers if appropriate). In the meantime, we can take the simple route and give everyone credit for that set of questions. This is why it is a good idea to keep the percentage of the grade and the level of instructor effort at a minimal level until you have confidence in the tool.

Integrating multiple tools is more manageable because we start small and then gradually increased tool usage, which mitigates risk and allows us to adapt and enhance how we incorporate the tools over time.

4.2.3 Choose tools that simplify management of student accounts and grades. As much as possible, we use tools that simplify course administration. This reduces errors and frees up instructor time to focus on students and instruction. We manually manage student accounts and porting grades as needed, but in most cases our tools are set up to communicate automatically.

Web-CAT now allows us to port grades directly to Canvas. In the past we would download a gradesheet from Canvas to ensure the proper column title for the grades that we were uploading. We learned to be sure to delete spreadsheet columns for other assignments so as to not unintentionally overwrite them when we upload. Keeping this spreadsheet open in Excel simultaneously with a tools' spreadsheet allows one sheet to reference the other using the Excel VLOOKUP command as needed to manually port grades.

Having the grades port directly from Web-CAT to Canvas takes significantly less time than any manual process and greatly reduces error in the transfer. We had formerly made errors such as porting grades over to the wrong assignment in the LMS or having all the student grades shifted down a row. Another key administrative benefit to grades automatically porting is that late grades are handled seamlessly. When we ported the grades manually, there would be added administrative work and more errors for the scenario where we ported grades for an assignment from Web-CAT to Canvas and then one student's grade was later changed in Web-CAT but this was not reflected in Canvas.

OpenDSA is now fully integrated with Canvas, so students do not need a separate account, and grades are automatically ported. Before OpenDSA automatically ported the grades over to Canvas, the student accounts were not necessarily the same. It is important that the roll from your LMS and the roll from your tool share some attribute that uniquely identifies each student. This is typically the student ID number or the institutional email. We formerly had grade errors because, despite instructions, some students would use a personal email to self-enroll in OpenDSA and then there was no shared unique key with the Canvas roll. Furthermore, some students would enroll in OpenDSA multiple times with various emails, so grades could be scattered across OpenDSA accounts, and various subsets might or might not get ported to Canvas. Having synchronized student accounts eliminates these issues and saves instructor time.

CodeWorkout can now be launched via Canvas, and grades are automatically ported over. However, we experienced similar issues with CodeWorkout as we formerly did with OpenDSA if we allowed students to self-enroll in CodeWorkout. Issues arose when students created multiple accounts and used different emails for their accounts, and additionally we had complications when students enrolled in the wrong or multiple sections of the course, which then affects their assignment settings. Using student accounts directly from Canvas in CodeWorkout, and not allowing students to self-enroll, saves time and reduces errors.

4.3 Potential Pitfalls

Integrating multiple tools into a course can positively impact student learning and instructor routines, but there are increased opportunities for complications. Our approaches have helped us proactively avoid many frustrations but there remain situations that can easily be mis-managed, especially when configuring a variety of tools independently.

One interesting example of a configuration issue and potential pitfall is whether students can rework the problems to study for a test. Some questions we ask ourselves are: Do the students have access to previously worked problems? If they rework them will they get regraded? If they rework them will they count as late? Currently we handle this in OpenDSA by closing assignments when they are due and then downloading grades before reopening questions the week before a test so students can use them for more practice.

Another interesting example of a configuration issue and potential pitfall is whether to record the grade for the last or best attempt from a student. This is the kind of option that may go unnoticed at first without a mentor. We find that on CodeWorkout we should take the best attempt since they might run out of time, but on Web-CAT we should take the last attempt since they are under less time pressure to make thoughtful decision about their submissions.

Hurdles such as these support the case that it is advisable to test your process with colleagues, teaching assistants, or by setting up multiple accounts of your own to enable testing. Versions, installation processes, interfaces, and security issues can change between semesters, so it is best to re-test to avoid unexpected student frustration. New issues can arise, but also, former issues might be resolved with updates.

5 CONCLUSIONS AND FUTURE WORK

By maintaining focus on student success and trying to streamline the process for instructors we have been able to integrate many tools into one course. We improve student success by scaffolding tool use, testing tool assignments before student use, and conscientiously paying attention to student feedback and behavior. We improve the experience for instructors by consulting mentors, starting small, and choosing tools that simplify course management. These approaches can be applied across educational settings. We reported on our experience with a large, in-person lecture. But many of these strategies can be applied to smaller classes, labs, and on-line courses. Our checklist in Figure 2 can be found at http://people.cs.vt.edu/~maellis1/teaching_resources/ and adapted to various circumstances.

Our future work includes integrating more active learning during lecture by expanding our use of CodeWorkout. We hope to improve student comprehension by using new content form OpenDSA with interactive questions for linked structures. We will also be adopting a version of Web-CAT with enhanced student feedback. While students benefit from scaffolding the use of the debugger, we plan to reduce the cognitive load for the debugger lab by making it about a programming concept that students have previously mastered. We are currently applying these approaches for integrating multiple tools into an updated version of our Problem Solving in CS course.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of the National Science Foundation under Grant DLR-1740765. We thank our colleagues Jack Lesko and Manuel Pérez-Quiñones for their constructive discussions during the course of the work described here.

REFERENCES

[1] C.T. Amelink, K. Davis, B.G. Ryder, and M.O. Ellis. 2018. Exploring Factors Influencing the Continued Interest in a Computer Science Major. In *2018 ASEE Annual Conference & Exposition.*

[2] A. Bandura. 1977. Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review* 84 (1977), 191–215.

[3] M. Beier, J. Lesko, and C. Amelink. 2018. Engineering Self-Efficacy: What it is, Why it Matters, and How to Encourage it! https://www.wepan.org/page/SelfEfficacy.

[4] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J. Gomez. 2018. The Effect of a Web-based Coding Tool with Automatic Feedback on Students' Performance and Perceptions. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. 2–7.

[5] P. Brusilovsky, S.H. Edwards, A. Kumar, L. Malmi, L. Benotti, D. Buck, P. Ihantola, R. Prince, T. Sirkiä, S. Sosnovsky, J. Urquiza, A. Vihavainen, and M. Wollowski. 2014. Increasing Adoption of Smart Learning Content for Computer Science Education. In *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference (ITiCSE-WGR '14)*. 31–57.

[6] A.T. Chamillard. 2011. Using a Student Response System in CS1 and CS2. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. 299–304.

[7] B. Cheanga, A. Kurniaa, A. Limb, and W.C. Oonc. 2002. On automated grading of programming assignments in an academic institution. *Computers & Education*, 121–131. Issue 41.

[8] S.H. Edwards. 2004. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '04)*. 26–30.

[9] S.H. Edwards, J. Snyder, M.A. Pérez-Quiñones, A. Allevato, K. Dongkwan, and B. Tretola. 2009. Comparing Effective and Ineffective Behaviors of Student Programmers. In *Proceedings of the Fifth International Workshop on Computing Education and Research (ICER '09)*. 3–14.

[10] J.V. Ernst, B.D. Bowen, and T.O. Williams. 2016. Freshman Engineering Students At-Risk Of Non-Matriculation: Self-Efficacy For Academic Learning. *American Journal of Engineering Education* 7 (June 2016), 9–18.

[11] E. Fouh, M. Akbar, and C.A. Shaffer. 2012. The Role of Visualization in Computer Science Education. *Computers in the Schools* 29 (2012), 95–117. Issue 1-2.

[12] D.D. Garcia, L. Aaronson, S. Kenner, C. Lewis, and S. Rodger. 2016. Technology We Can't Live Without!, Revisited. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. 236–237.

[13] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S.H. Edwards, E. Isohanni, A. Korhonen, A. Petersen, K. Rivers, M.Á. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, and D. Toll. 2015. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports (ITICSE-WGR '15)*. 41–63.

[14] R. Ishizue, K. Sakamoto, H. Washizaki, and Y. Fukazawa. 2018. PVC: Visualizing C Programs on Web Browsers for Novices. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. 245–250.

[15] A.N. Kumar. 2008. The Effect of Using Problem-solving Software Tutors on the Self-confidence of Female Students. *SIGCSE Bull.* 40, 1 (March 2008), 523–527.

[16] R.B. Levy and M. Ben-Ari. 2007. We Work So Hard and They Don'T Use It: Acceptance of Software Tools by Teachers. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '07)*. ACM, New York, NY, USA, 246–250. https://doi.org/10.1145/1268784.1268856

[17] R.B. Levy and M. Ben-Ari. 2008. Perceived behavior control and its influence on the adoption of software tools. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education (ITiCSE '08)*. ACM, 169–173. https://doi.org/10.1145/1384271.1384318

[18] T. Naps, S. Cooper, B. Koldehofe, C. Leska, G. Rößling, W. Dann, A. Korhonen, L. Malmi, J. Rantakokko, R.J. Ross, J. Anderson, R. Fleischer, M. Kuittinen, and M. McNally. 2003. Evaluation the educational impact of visualization. In *Working Group Reports from the Symposium on Innovation and Technology in Computer Science Education (ITiCSE-WGR '03)*. 124–136.

[19] C.A. Shaffer, M.L. Cooper, A.J.D. Alon, M. Akbar, M. Stewart, S. Ponce, and S.H. Edwards. 2010. Algorithm Visualization: The State of the Field. *ACM Transactions on Computing Education* 10 (August 2010), 1–22.

[20] C. Watson and F.W.B. Li. 2014. Failure Rates in Introductory Programming Revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14)*. 39–44.

[21] C. Wilcox. 2015. The Role of Automation in Undergraduate Computer Science Education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. 90–95.