# Beyond the Flipped Classroom: Learning by Doing Through Challenges and Hack-a-thons

Michael Skirpan
Department of Computer Science
University of Colorado at Boulder
michael.skirpan@colorado.edu

Tom Yeh
Department of Computer Science
University of Colorado at Boulder
tom.yeh@colorado.edu

## ABSTRACT
Traditionally, the inverted (or flipped) classroom has students complete traditional, passive learning tasks (e.g., watching lectures) while at home and uses class time to actualize what is learned through labs, discussions, and exercises. In this paper, we present an instructional model for teaching computer science (CS) that compounds features of the flipped classroom with components of peer instruction and formative assessment. Outside of class, in lieu of a lecture, students worked collaboratively on *learning challenges* that introduced content through a series of hands-on exercises. During class time, we used *hack-a-thons* to create an active classroom environment to promote peer coding and cultivate the growth of relevant real-world technical skills. Class work was digitally synced to Google Drive in real-time to allow instructors the opportunity to customize on-the-spot feedback. Further, we used journals as a formative assessment measure to synthesize student interests and opinions into our continued design of the class. In this paper, we describe our pedagogical model and discuss the results and lessons learned from the class using mined data from Google Drive and student journal responses.

## Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer Science Education, Curriculum.*

## Keywords
Flipped classroom; inverted classroom; hack-a-thon; pedagogy; peer instruction; formative assessment

## 1. INTRODUCTION
In education, there is an age-old tension between traditional and progressive approaches to schooling [12]. The standard division is characterized between classrooms where the teacher is at the center of knowledge construction (e.g., lecturing, delegating required homework, and testing) versus a student-centered approach where student are responsible for building knowledge through their own interests and activities. While this debate has been ongoing, it has recently become central to the discussion of how to best teach STEM content and, central to this paper, computer science (CS).

Recently, in CS education, attention has been placed on the inverted, or flipped, classroom approach as an alternative to a traditional model [3, 5, 6, 8, 10]. Historically, the philosophy of learning through action derives from experiential educational models suggested by thinkers like John Dewey, William Wirt, and Lev Vygotsky. Contemporary publications primarily draw their model from work done by Eric Mazur in the 1990s. Mazur discussed using computers in the classroom to allow the teacher more time to coach students on specific struggles [9] and later developed a flipped classroom technique involving peer instruction [4]. However innovative, this style has continued to rely on students watching lectures at home while class time is spent taking conceptual quizzes and performing problem-solving exercises in groups. Although this approach takes a step forward to improve the traditional model of instruction, similar issues of engagement and retention are still inherent.

**Figure 1. Weekly hack-a-thons in class where students design, build, and demo a prototype in 150 minutes**

Video lectures often promote limited attention to the instructor because multitasking and personal distractions are in direct competition with the lecture content. Moreover, the overall class content is still centered on what is chosen by the instructor, ceding little ownership to the student in regard to the content learned. Given that student responsibility and "doing over knowing" are two goals of the flipped classroom model [5], there still appears to be room for progress.

Because we believe in the movement toward more progressive approaches to teaching CS yet are cognizant of these issues, we decided to make some further modifications to the flipped classroom model. Namely, we ushered in a formative assessment component to the curriculum to improve student buy-in to class content and scrapped passive lecturing altogether in favor of a fully hands-on approach that emphasizes the creation of tangible projects.

We explore this modified model and the results from our experience in this paper. It should be noted that, for our first implementation of this model, we chose a full adoption. That is, scheduled class time was 100% hack-a-thon and 0% lecture (see Figure 1). We recognized beforehand that the ideal adoption would likely be less extreme, but we wanted to test the limits of the model and use student feedback to reassess. In sections 2 and 3, we will delineate our model and provide a detailed example of it in action, and in sections 4 and 5, we will share our results and return to the question of ideal adoption.

## 2. INSTRUCTIONAL MODEL
From a macro-view, we built our model around core of skill building and project delivery. We selected content that was relevant a) to the skills students were most interested in acquiring and b) to the skills applicable to creating professional-quality web applications. Further, all learning was structured around projects;

both small projects that would be completed in one or two weeks and a large final project that involved creating a full application that could be incorporated into student portfolios. To accomplish these criteria, we opted to deliver the content in a dynamic manner: Though we created a full syllabus for preparatory reasons, the class content often changed as we learned more about our students and discovered learning struggles that required more time. Despite having dynamic *content*, we adhered to a class model with four major components: learning challenges, journals, in-class hack-a-thons, and a semester project. We used Google Drive as a management tool to track student work and handle submissions, the details of which we will discuss in section 3 on implementation.

## 2.1 Learning Challenges

Students began each week by going to a link posted on our course website. The link took them to a Google Document (i.e., the learning challenge) where new content would be introduced. These documents guided students through a series of challenges to be completed, usually starting with a simple "hello world" application. A challenge would be highlighted and numbered with a description of a deliverable we expected in their submission document. Each challenge was contextualized with links to helpful pages such as select code documentation and Stack-Overflow answers to common issues. At each of these junctures, students would be asked to reproduce a shown result or implement a solution to a problem given certain constraints. They then submitted a screenshot or code snippet as supporting material to show completion of the challenge.

Challenges would often entail working on different facets of a mini-project being built throughout the homework. Sometimes a challenge could be completed in one screenshot, but often there were a series of sub-challenges that asked them to share smaller parts of their process. For submission, we posted a template in the form of a Google Presentation that would have a single slide for each challenge or sub-challenge deliverable. Completing a submission involved copying the template and placing it in their personal folder having filled in each slide. To leverage the diverse skillsets across the class, we asked students to choose one or two partners each week and provide evidence they worked together by submitting a *selfie* photo of them working together. We imposed this restriction to promote peer instruction, allowing students to help each other.

Though the challenges imposed a strict set of functionality to implement, we opened the door for student creativity by remaining flexible about accepting different approaches to the same solution and allowing students to work on custom content (e.g., personal websites, senior design projects, open source contributions). This was our solution to an inherent tension of wanting to ensure students were pushed to learn new skills while providing them with the opportunity to work on projects within their own range of interests.

Our style of "homework" is one of the most distinguishing features between our implementation of the flipped classroom model and other methods. A common approach in the flipped model is to provide students with online video lectures that pose short-answer and multiple-choice quiz questions along with a few practice exercises along the way [8]. Our model differs primarily in that we drop the lecture component completely and instead scaffold the learning exercises with a narrative (i.e., tutorial) that replaces the didactic content traditionally found in lectures.

Further, through the use of Google Drive, we are given access to fine-grain temporal data about when student work was completed and knowledge about who worked on which documents (i.e., who collaborated with whom). Once the students completed the challenges, we also gave them a weekly journal template to fill out, which asked them to reflect on what they had learned and to provide feedback about the class. Further details on our use of Google Drive will be discussed in section 3.2 below.

## 2.2 Journals

Every week, the students were given a set of questions to submit with their challenges that encouraged reflection, criticism, and suggestions. Typically, we would survey them with a mixture of quantitative and qualitative questions. We presented both recurring questions, establishing baseline measures on the weekly curricula, and novel questions pertinent to that week's specific content.

Common quantitative questions regarded the length of time spent on and the perceived difficulty of the assignment. Qualitatively, we would usually ask students to share resources (i.e., web links) they found useful to complete the homework, explain barriers they hit during the assignment, and what else they would have liked to learn if they were to complete the assignment again. This data would then be used by the instructors to make decisions about what content to include in the weekly hack-a-thon, whom to place in teams for the hack-a-thon, and what topics required more work in the coming weeks. For timing reasons, we did not read all journal entries each week but instead set aside time to glance through them quickly as a way to determine common struggles.

Journals provided a means for us to conduct a formative assessment of the class. Formative assessment is defined as "assessment carried out during the instructional process for the purpose of improving teaching or learning" [11]. In educational research, formative assessment has been cited as one of the most powerful sources to improve student learning and class engagement [7]. Such mechanisms have been implemented in prior CS teaching using a revision cycle for homework questions to improve learning outcomes [2]. However, our focus on formative assessment was designed to improve *teaching*; thus, we used the journal as a direct, private channel for students to inform instructors about what was good or bad, difficult or easy, interesting or boring, regarding each week's assignments and content. All journal responses were submitted in the same Google Presentation document as the challenges. An example is discussed in section 4.3.

## 2.3 In-Class Hack-a-Thon

Hack-a-thons (also called "sprints") have become a popular method whereby coders come together to produce first versions of projects, create open-source content, share skills and insights in their work, and perform coding work for the public good. Beyond being a norm in the coding community, this spontaneous process is useful for collaboration, generating creative results, and learning to work with deadlines. Thus, to have an active classroom and observe our students true abilities, we used class time to "hack."

The process would start with students either self-selecting or the instructors assigning teams of three to five students. It was known ahead of time that the hack-a-thon would involve an implementation using the materials seen in the previous week's learning challenge, although the specific task would be kept private until students arrived in class. Once students were in their teams, we asked them to go to the course website, where a link had recently been added, directing them to the guidelines for the

week. Similar to the learning challenge, the hack-a-thon guidelines were paired with a submission template to fill out.

The hack-a-thon guidelines would offer general design challenges (e.g., make a robot dating website) and list milestones that students needed to accomplish (e.g., take a screenshot of a robot dating profile). Milestones were similar to challenges insofar as they were pieces of functionality that needed to be proven by submitting a screenshot or code snippet. One major difference between the challenge and hack-a-thon is that we enforced a design phase at the beginning of each hack-a-thon that asked the teams to produce an outline of the work to be completed and a declaration of who would be responsible for what parts of the implementation. The design phase would be limited to the first thirty minutes of class and would allow instructors to see each team's goals in real-time by using a similar Google Drive approach from the learning challenges. Using live syncing of the files, we could then observe each team's progress, provide individualized feedback, and offer help to those who struggled.

During the hack-a-thon, the instructors would immerse themselves in discussion and consultation with the teams. The last fifteen minutes of class would be devoted to a short showcase of what each team accomplished because every team often had vastly different strategies for implementation. In the end, the hack-a-thons would be judged by external parties (often friends of the instructors, local coders, or other graduate students) to provide objective feedback on the end products. This allowed students to receive real-world critiques regarding design, usability, and coding patterns that allowed them to improve beyond the requirements of the class. Having external judges allowed the instructors to focus on evaluating student learning fairly and kept subjective judgments out of the grading scheme while still delivering insight to our students on what to expect from authentic users and employers.

The hack-a-thon component of the class is similar to other flipped classrooms in that it generates an active environment during class time. A more common way to achieve this would be to have a series of in-class exercises similar to traditional homework or to present hands-on examples that are traditionally solved by the instructor during lecture [6, 10]. Our model differs slightly in that we use the hack-a-thon to gear class time toward a single project-based learning exercise in which groups work collaboratively toward a result that has no right or wrong answer. Rather than ask students to work toward a predetermined solution, our model attempts to emulate a real-world implementation whereby the creative team is central to the outcome, and failure and iteration are expected parts of the process.

## 2.4 Semester Project

The culminating component of the class was an in-depth project for which students could use any mixture of the tools we learned about during the semester. Students also had the opportunity to integrate outside skills and interests into their final project. Students were informed of the project at the beginning of the semester and were encouraged to start thinking about what they wanted to create and find a team with which they wanted to work. At the semester midpoint, we asked students to solidify teams and submit a project proposal that outlined the purpose of the project, what functionality they planned to implement, who would be in charge of what aspects of the project, and a GitHub repository where we could find the final project. Once these were submitted, instructors read through them to ensure that they met class expectations and that the work was both reasonable and evenly divided among the team.

As a way to keep a tally on the projects and feel confident in their progress, we asked students to self-impose four milestones (submitted with their weekly learning challenges) that would keep them on track throughout their project. Although milestones were taken into consideration in the final project grades, we primarily used these as motivating factors to ensure that groups stayed on track. For our final class meeting, we held a presentation together with five other CS classes to exhibit the students' work.

Our project takes a similar approach to that described in past hands-on instructional methodologies [1] in that we asked students to develop milestones and report their progress by submitting pieces of code and screenshots. However, we did not place strong structural constraints on the project. The only requirements were based on using some of the tools learned in class and choosing milestones suitable to prove reasonable progress; otherwise, the students were free to implement anything.

# 3. IMPLEMENTATION: SPRING 2014

We now move on to describe the specific details of the class setting, student population, material taught, and evaluation methodology. The course we chose for this model was "User-Centered Design and Development," a cross-listed undergraduate and graduate course that focused on front-end development skills and creating web servers to host applications. Our class met for 2.5 hours once per week throughout a single 15-week semester, and anticipated a 10-hour weekly work commitment.

## 3.1 Class Setting and Student Population

Selecting a class setting was important, as we learned on our first day of class when we tried to manage the hack-a-thon in a traditional classroom environment. Not only were the standard single-occupant desks a hindrance for collaboration, but also the lecture-based classroom was too small for instructors to walk readily between groups, neither did it supply enough plugs for every student to code for the entire class time. Thus, we quickly scouted better campus spaces, first trying a lab space (which worked well but presented scheduling issues) and then a lounge space.

As shown in Figure 1, we began hosting the class from a student lounge space that contained large worktables as well as couches and chairs placed around coffee tables. This space proved to be most beneficial for everyone.

The class had 31 students in total: 8 graduate students and 23 undergraduates. Among the 8 graduate students, 7 were master students and 1 was a PhD student. Of the 23 undergraduate students, 22 were computer science majors and 1 was a mechanical engineer. Most of the undergraduates were seniors or juniors (5 and 17, respectively) with the addition of 1 sophomore. Within the total population were 4 female students (2 graduate, 2 undergraduate) and 28 male students. Using student journal data, we found that 94% of the students had previous front-end experience (i.e., HTML, CSS, or JavaScript), but only 32% had experience working with web servers (e.g., Apache).

## 3.2 Use of Google Drive

To prepare for the semester, we set up Google Drive folders for every student. Permissions were set so that student folders were accessible only by that student and the instructors. This gave students the ability to share documents for collaboration while keeping their journals, project proposal, and feedback private.

For weekly learning challenges and journals, we used a public folder that contained the templates files (Google Presentations with slides for each submission) from which students made

personal copies. Submissions for hack-a-thons, on the other hand, were handled within a single, public file. Teams managed their own internal file sharing, but milestone submissions were placed onto a shared Google Presentation, which had preset blocks of slides for different teams to use. The start of the hack-a-thons involved having the instructors post a link on the class website, directing students to the hack-a-thon outline document, and following a further link from there to the submission document where each team laid claim to a block of slides by placing their names at the head slide.

We chose Google Presentations as a primary medium for student submissions because of the extreme flexibility it granted us. Slides could easily be used to add an image, type a journal entry, or work with a mock-up, and the entire document easily turned into a shareable presentation. It also allowed us easy access to an edit history on all student documents.

## 3.3 Class Content

The content students were required to learn was separated into breadth and depth aspects of the course. The depth content structured the requirements of the semester-long projects. We offered graduate students the option of learning to use the LEAP Motion or the XBOX Kinect. The only depth requirement for undergraduates was that their final project must use the ubiquitous JavaScript library, JQuery.

All weekly assignments were part of the breadth dimension of the class, familiarizing students with a wide range of tools. Core topics included creating website wireframes using Balsamiq and PopApp, website development using Bootstrap, building web-servers with Python Flask and a SQLAlchemy back-end, and application hosting via GitHub and Heroku. During the final weeks of class, when student projects were exhausting more time, we taught "fun" topics based on class interests, which included 3D Modeling in Google Sketchup and OpenSCAD, hardware programming with Picaxe Robots, and tangible interfaces with Makey-Makey. This offered students a full-scope overview of designing a user experience from the backend of a webserver all the way to a physical prototype that can be held by an end-user.

## 3.4 Course Evaluation

The final grade was weighted as follows: hack-a-thons 25%, learning challenges 30%, journal 15%, and final project 30%. All students kept their own folder on Google Drive, which was shared with the instructors. Learning challenges and journals were due by 11:59 PM the evening before class, and hack-a-thons were due by the end of class.

The weekly learning challenges were graded on three criteria: timeliness, completion, and correctness. Credit was given only to submissions that were handed in before the final deadline, which we tracked via Google Drive. Once we eliminated late submissions, we then credited half a point for turning in any submission for a given challenge and credited the other half point for challenges that were correct. Correctness was a lenient measure, as we were aware that there were always multiple ways to approach a solution thus credit was given as long as the screenshot or code snippet showed a basic competency at completing the task. Journals were graded at the same time as weekly challenges (They were submitted together.), and the only criterion was completion: 1 point for each question answered.

Hack-a-thons were graded on both completion of the basic milestones and effort. We did not take off points for students who had broken pieces of their websites or for unfinished functionality because each week presented a brand new project and punishing

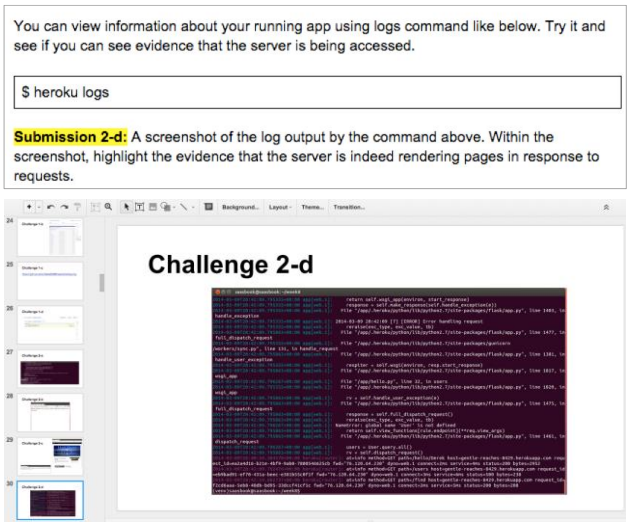


**Figure 2. Example of a learning challenge and a journal entry by a student using Google Presentation**

students for these shortcomings would likely have promoted less ambitious work. Therefore, although a baseline amount of effort was required to gain credit, the hack-a-thon was primarily about attending and attempting. We encountered almost no issues with students' not working hard in class, possibly due to the Google Drive's syncing and thus causing them to realize that we were aware of their efforts. Due to hack-a-thons, attendance was a major part of each student's grade. Class policy allowed students only one unexcused absence, which could be made up at the end of the semester when we held a makeup hack-a-thon.

Semester projects were graded for completion of milestones and final functionality, both of which were agreed upon in the project proposal. Student roles were also outlined in the proposal, so if there were any missing submissions or broken functionality, the student responsible lost points rather than the whole group.

## 3.5 One Week Up Close

To solidify how separate working parts of our class model came together in practice, we provide a short tour through a single week. All examples come from week 7 of the semester (March 2–9, 2014) when we taught website hosting using GitHub, Heroku, and Python Flask.

**Learning Challenge:** This week started by ensuring students knew how to use GitHub. They had the choice of following a GitHub tutorial to which we supplied a link or to demonstrate to us that they already know how to use it by submitting screenshots of a new repository and a fresh commit to the repository. We then linked them to material that introduced Heroku web hosting and set the challenge of using Heroku to host the Python Flask server they had built in previous weeks. Figure 2 (above) provides an example student submission of Heroku server logs. Once students understood the Heroku service, we asked them to take a website mockup from a previous week and generate a static HTML page using Bootstrap (a CSS library taught in a previous week). The page was to be viewable both on a desktop and mobile device. Finally, they were to create a serviceable route to this static page via Python Flask and serve it live on the public web via Heroku.

**Journal:** Once the learning challenges were complete, we asked students to use our journal template to report on several questions. As mentioned above, some questions were consistent through the

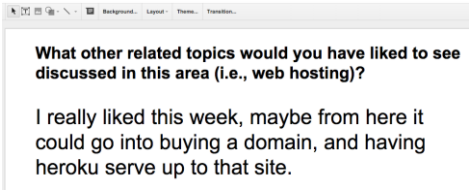


**Figure 3. Student's answer to a survey question.**

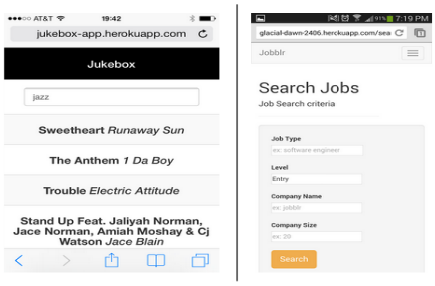


**Figure 4. Two prototypes built by hack-a-thon teams**

weeks, and others were specific to the topic at hand. We then used this content to construct the hack-a-thon and improve on the following weeks' challenges. Figure 3 offers a sample suggestion a student offered after completing the web-hosting challenge.

**Hack-a-thon:** For this week's hack-a-thon, we asked students to create a mobile application similar to what they would pitch for a startup. We asked them to plan, implement, and demo their prototype and submit the planning information, user scenario descriptions, and screenshots. Figure 4 shows two smartphone screenshots of the actual mobile applications submitted by students as a result of the hack-a-thon.

# 4. RESULTS

Many metrics were collected throughout the semester. We used Google Drive's API and continually read student journals to gather both qualitative and quantitative data about the class.

## 4.1 Attendance and Completion Rates

Under our model, attendance was a prominent indicator of success in the class. At the beginning of the semester, we expected students to drop the class once they realized that attending and participating in class would be a huge portion of their grade. To the contrary, we maintained a large class size and observed little diminishment of attendance rates. The overall class attendance rate was 92% (excluding excused absences).

Regarding the completion of weekly learning challenges (i.e., homework), we observed high rates of completion. Overall, 89% of challenges were submitted before the deadline and given full credit. The two weeks for which we recorded the lowest scores were when we first asked students to have a fully functional web server (81%) and when they learned OpenSCAD 3D modeling (72%). The first dip was expected, given the incoming experience of the students with web servers, and the second dip in student work may have occurred because it coincided with students' turning in their first project milestone.

Journal completion rates were almost precisely correlated with weekly learning challenges, with an overall average of 89%. This relationship is likely because journals were due at the same time and in the same document as the learning challenges. Thus, if students gave up on completing the homework or chose not to attempt it at all, they were likely not to complete the journal.


**Figure 5. Revision history of a hack-a-thon submission**

## 4.2 Student Revisions

One novel affordance of our use of Google Drive was the ability to see fine-grained revision histories on student documents, including the timestamp, username, and restorable revision for each and every edit made to a student document. This is contrasted with a traditional, coarse-grain work history in which student work is seen only upon homework/exam submission. Figure 5 shows what this revision history looks like for hack-a-thon document. This data allowed us to observe how students collaborated. (Each student's activity has a different color.) Also, we could restore old revisions to gain insight into their process.

Further, using the available revision history, we were able to conduct analyses and render graphics that provided instructors with information about student work patterns. These graphics guided us toward improved instructional strategies. Figure 6 is a sample graph from our class data that shows three kinds of student work patterns: those who started and finished early, those who started earlier but finished closer to the deadline, and those who waited until just before the deadline to finish. In light of these patterns, we responded by making a single submission due in the middle of the week to encourage students to start the assignments early because the early workers tended finished more often.

## 4.3 Student Reporting

Qualitative and quantitative information from the journals were used to determine areas in which the class needed to improve. For instance, in week 7 during the introduction to web servers, we discovered that several students had spent 10+ hours on homework and provided feedback stating they were still very confused about how the routing system worked. In response, we used the next week as a review week and separated the class into "student instructors" and "learners," whereby students who were comfortable with the material helped struggling students.

Another use of student journal reporting was to provide instructors with broad feedback about the class and collect metrics pertinent to the improvement of future iterations of the class. These metrics could be analyzed on their own or in comparison with more objective metrics via the data available from Google Drive. In our reflection section below, many of our suggested improvements for our model are distilled from these student reports. Figure 7 is one such outcome of analyzing student journals. It shows that the time spent on an assignment correlates well with its perceived difficulty. The lines are less correlated in the final weeks when we worked on 3D modeling, a topic students considered easier than coding but more time consuming.

# 5. REFLECTION & CONCLUSION

## 5.1 Generalizability

To what extent could our instructional model prove successful for other classes? Our model relies on having instructors willing to devote a large amount of time to the initial course design and to

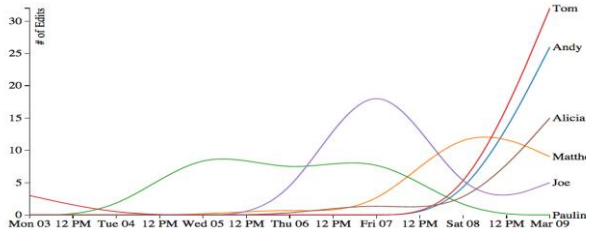


**Figure 6. Edit patterns of students' documents for a learning challenge throughout the week it was assigned**

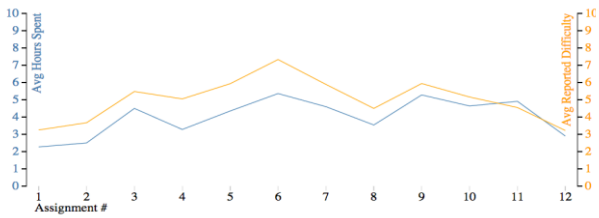


**Figure 7. Average hours spent on 12 weekly assignments (blue) compared to reported difficulty (orange)**

the development of learning challenges and hack-a-thons on a weekly basis. If full implementation were too difficult, a partial implementation (e.g., using the model every other week) would be possible and useful for most CS classes.

One factor that made our class exceptional was the amount of work that was visual in essence. In contrast, a class that focuses exclusively on code would likely scale down the use of screenshots as submissions and instead require links to GitHub repositories. However, screenshots could still prove useful for showing compilation and runtime errors, and Google Drive syncing could help instructors understand student work patterns. Most importantly, the use of class time for authentic code production seems to be generalizable to almost any coding classroom. In classes teaching large-scale software production (e.g., software engineering), hack-a-thons could build on one another in such a way that a final submission takes many sessions.

Another factor is class time. Our model requires longer class time (2.5 hours) for hack-a-thons to be viable. In CS1 courses, the coding is likely simple enough that 50 minutes would be an effective time, but for advanced courses a short class time would be a severe hindrance to implementing our model.

## 5.2 Class Discussion

One major change we will make to the class in future iterations is to reserve 20–30 minutes at the beginning of each class for group discussions. Student feedback revealed that, although students loved the hack-a-thon as the fundamental class component, they wished a small amount of time were devoted to overviewing common errors, fielding questions about the week's content, and talking about ideal use cases and competing tools. Using this feedback, we now realize that the ideal class time balance should be closer to 20% discussion/lecture and 80% hack-a-thon.

## 5.3 Consistency and Structure

Another improvement we will make to the class is on the overall structure of the course and assignments. The students wanted content to build upon itself instead of being asked often to replicate designs to complete work with new tools. This would have taken much more planning but would have improved the number of polished deliverables.

A lack of structure was a trade-off we knowingly made to implement the extreme, no-lecture model of the class. This created a learning curve for students to get comfortable with the weekly class cycle. Although it only took 2–3 weeks for students to settle into the class model, we believe this could be mitigated by having a better class website where all documents, links, and templates are centralized and the submission process is explained.

## 5.4 Next Implementation

Beginning in fall 2014, we are running a second implementation of this model in a data science class entitled "Big Data." We now provide a 20-minute window at the beginning of class for discussion and 5 minutes at the end of class for students to seek help with the semester-long project. Also, the semester project has become a larger component in the class so that students can use early-semester data collection from their project as content for class hack-a-thons, allowing them to work with data they are interested in and refine their project design. This class will provide further insights into the generalizability of this model.

# 6. REFERENCES


[1] Balasubramanian, R., York, Z., Doran, M., Biswas, A., Girgin, T., & Sankaralingam, K. Hands-on instroduction to computer science at freshman level. In *SIGCSE '14*. 235-240. DOI= http://doi.acm.org/10.1145/2538862.2538889.

[2] Blaheta, D. Reinventing homework as cooperative, formative assessment. In *SIGCSE '14*. 489-494. DOI= http://doi.acm.org/10.1145/2538862.2538915.

[3] Campbell, J., Horton, D., Craig, M., & Gries, P. Evaluating an inverted CS1. In *SIGCSE '14*. 307-312. DOI= http://doi.acm.org/10.1145/2591708.2591752.

[4] Crouch, C.H. & Mazur, E. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970, 2001.

[5] Gannod, G. C., Burge, J. E., & Helmick, M. T. Using the inverted classroom to teach software engineering. In *ICSE '08*. 777-786. DOI= http://doi.acm.org/10.1145/1368088.1368198.

[6] Gehringer, E. F., & Peddycord III, B. W. 2013. The inverted-lecture model: a case study in computer architecture. In *SIGCSE '13*. 489-494. DOI= http://doi.acm.org/10.1145/2445196.2445343.

[7] Hattie, J., & Timperley, H. The power of feedback. *Review of educational research* 77(1), 81-112, 2007.

[8] Horton, D., Craig, M., Campbell, J., Gries, P., & Zingaro, D. Comparing outcomes in inverted and traditional CS1. In *SIGSCE '14*. 261-266. DOI= http://doi.acm.org/10.1145/2591708.2591752.

[9] Mazur, Eric, Feb. 1991, *"Can We Teach Computers to Teach?,"* Retrieved September 1, 2014, from http://mazur.harvard.edu/sentFiles/Mazur_256459.pdf

[10] Rutherfoord, R. H., & Rutherfoord, J. K. Flipping the classroom: is it for you?. In *SIGITE '13*. 19-22. DOI= http://doi.acm.org/10.1145/2512276.2512299.

[11] Shepard, L.A. Formative assessment: caveat emptor. In *The future of assessment: shaping teaching and learning, ETS invitational conference*, New York, 2005.

[12] Tyack, D. B. *The one best system: A history of American urban education*. Harvard University Press, 1974