Learning Analytics in a Computational Thinking Course on Data Science

AUSTIN CORY BART, Virginia Tech
JAVIER TIBAU, Virginia Tech
ELI TILEVICH, Virginia Tech
DENNIS KAFURA, Virginia Tech
CLIFFORD A. SHAFFER, Virginia Tech

CCS Concepts: •**Social and professional topics** → **Computational thinking;** *Student assessment;* •**Applied computing** → Interactive learning environments;

Additional Key Words and Phrases: Learning Analytics, Computational Thinking, Computing Education, Data Mining

ACM Reference Format:
Austin Cory Bart, Javier Tibau, Eli Tilevich, Dennis Kafura, and Clifford A. Shaffer, 2016. Data Mining a Computational Thinking Course on Data Science. *ACM Trans. Embedd. Comput. Syst.* , , Article (May 2016), 29 pages.
DOI: 0000001.0000001

1. INTRODUCTION

As students of all majors and career paths are increasingly expected to learn about computing, computer science education is challenged by the unprecedented scale and diversity of the needs. Many institutions now offer courses in "Computational Thinking", which cover foundational computing concepts—including algorithm development, data abstraction, and computing ethics—to educate diverse majors from the sciences, arts, humanities, and other fields. Although the material covered is not as rigorous or in-depth as traditional introductory computing courses, thinking computationally is often new for these students and they may suffer from low self-efficacy about their ability. Further, as non-major students often fail to immediately appreciate how computing can benefit their long-term career, their resulting motivation levels can prove insufficient to see them through the more challenging aspects of the curriculum. Hence, courses that serve these students need to focus on both cognitive and motivational concerns. Ideally, new techniques should be developed to identify which students are struggling as early as possible, so that interventions can be staged to rescue and guide learners.

In this paper, we describe a new iteration of our Computational Thinking course for undergraduate non-Computer Science majors. We have previously reported on an

This material is based upon work supported by a grant from the National Science Foundation Graduate Research Fellowship, Grant No. DGE 0822220.
Author's addresses: Austin Cory Bart, Javier Tibau, Eli Tilevich, Dennis Kafura, and Clifford A. Shaffer, Computer Science Department, Virginia Tech.

earlier version of this course [Kafura et al. 2015], but with an emphasis on the tools and curriculum designed rather than on assessments, course outcomes, and our data analysis process. Not only have we iterated upon the design of the course, but we have improved the instrumentation to collect fine-grained data about our learners. In this paper, we explain the nature and mechanics of the course and explore the results of learning analytics collected during the latest offering. Our goal is to establish ground truth for the motivational, engagement, and cognitive outcomes of our course. Achieving these outcomes has previously been based on educational theory rather than driven by the results of data collection. We also report on initial approaches to finding whether these outcomes can be predicted earlier in the course, in order to identify at-risk students. Finally, we review the needed changes for our course that are being implemented in our current offering this semester, so that we can validate our results.

In particular, we make the following contributions:

(1) Identify a number of early-course behaviors (identified by student analytics) that can be used to predict student performance later in the course.
(2) Propose several new mechanisms for assessing students, and attempt to establish their validity.
(3) Describe in more detail our curriculum.

Readers interested in the design process for our curriculum will find this presented in Sections 2 (Background) and 3(Course Components). Readers interested in the data analytics aspects of this paper - the first two contributions - will find this material in Sections 4 (Data Collection), 5 (Research Questions), and 6 (Threats to Validity). Section 7 (Iterating the Course) describes how the data analytics results guided revisions to the course content and pedagogy. Brief conclusions are given in Section 8.

2. BACKGROUND

Both the design and evaluation of our course has largely been driven by theory. The course material is based on work in the subfield of Computational Thinking, with a number of notable divergences. The pedagogy and structure of the course is informed by educational theories such as Situated Learning Theory, Active Learning, and Instructional Design. Finally, our evaluation is modeled after similar efforts in the Computing Education Research literature.

2.1. Computational Thinking

When we first started designing our course to teach “Computational Thinking”, we felt it was essential to precisely define the material we would cover. Unfortunately, there is still limited consensus on *what* exactly CT is, how it should be taught, and how to assess learner’s understanding. An excellent resource for summarizing the history of Computational Thinking research is the 2013 dissertation by Wienberg [Weinberg 2013]. The term “Computational Thinking” was coined by Seymour Papert in 1993 [Papert 1996] and popularized by Jeannette Wing in 2006 [Wing 2006]. Wineberg’s comprehensive survey analyzed 6906 papers directly or indirectly related to Computational Thinking that emerged after Wing’s paper. This survey particularly criticizes the existing research for a lack of assessment and evaluation, with few papers reporting an operational definition of computational thinking (and many of those simply describing computational thinking as a “way of thinking”, a “fundamental skill”, or a “way of solving problems”).

In the past decade, a number of initiatives have established learning objectives that attempt to describe computational thinking [Google Inc. 2011; ISTE & CSTA 2011], incorporating a range of topics. Our particular definitions (given more concretely in Section 3.2) are distilled from these objectives, focusing on Abstraction (represent-ing real-world objects with quantifiable properties), Algorithms (manipulating those properties with concrete instructions), and the Social Impacts of computing (relating the implications and interpretations of computations to stakeholders and society). We chose these [three particular key learning objectives] in order to keep our [core learning objectives] simple and provide a clear overarching message to students. [The term ”learning objectives” is overloaded in the previous sentence (and the next one). Can't use the same term for obviously two different things.] Besides our [chosen learning objectives], we also diverge from much of the literature by choosing to focus on undergraduate education instead of for K-12.

An ongoing debate within the Computational Thinking subfield is the role that programming should play in such a course. Some feel that it is appropriate to avoid teaching any programming [Walker 2015b]. Others feel that it is impossible to teach meaningful computational methods without some level of programming [Walker 2015a; Cooper and Dann 2015]. We agree with the latter sentiment, believing that students can only concretely learn to think computationally when they actively engage with algorithmic materials, and programming is an excellent medium for doing so. Further, we also want our course to impart practical skills to our learners.

As do many others, we frame our course using a “theme”. In our case, the theme is “Data Science.” One popular theme incorporates games and animation as a motivating context for computing [Guzdial 2003; Repenning 2013]. [I don't know that ”media comp” is recognizable using the tag ”games and animation”. Maybe need a better term for this.] This was a deliberate contrast to presenting computing in a decontextualized, mathematical manner (though there are still attempts at this approach, see [Mneimneh 2015]). Although games and animations offer students many interesting activities, we were concerned that these do not help students to accomplish tasks in their own field, and that they will not be able to transfer their knowledge. Therefore, we wanted to use a theme that would be relevant to their own disciplinary activities. To that end, we have created a curriculum that uses data science, inspired by several other similar endeavors [Goldweber et al. 2012; Anderson et al. 2014]. [”Similar” in what way? Similar in that they use Data Science? Or do you mean similar in that they have themes? If so, this needs to be reworded.]

Perhaps the most similar effort to ours thematically is the set of four courses described in [Anderson et al. 2015], which also uses data science to convey course concepts. [This is really four courses? Does not seem so from rest of paragraph.] We share their sentiment that, “The approach is more than just a collection of nifty assignments; rather, it affects the choice of topics and pedagogy.” However, although we agree in vision, we depart in execution. In particular, we attempt to provide students with more opportunities to explore their own interests and career paths through course assignments (in particular, the final project). Meanwhile, Anderson, et al.'s course [Singular?] is restricted to particular prepackaged units on topics that they hope will be motivating for a variety of students. Another key difference is that they were developing a CS-1 level course rather than a course for non-majors, allowing them more rigor and topics.

Another course similar to ours is described in [Plaue and Cook 2015]. We share the idea of empowering learners to follow topics related to their interests and career paths. We also share a strong emphasis on manipulating and visualizing datasets. However, we have no lessons in designing or developing web-based materials. Further, we allow students to explore more general themes of data science rather than focusing on a journalistic approach.

Although we have never intended to emphasize statistics in our approach, our final course design shares characteristics with [Hall-Holt and Sanft 2015], who attempted to create an introductory computing course specifically with statistics in mind. Like us, they incorporate data science components at a deep level. However, from a pedagogical perspective, it seems that their course teaches useful data science techniques first, and introductory computing concepts second. For instance, loops were delayed until much later, since loops are of less use in their language of choice (R). We teach loops rather than other approaches to working with lists, so that students can learn the computing concept of iteration.

2.2. MUSIC Model of Academic Motivation

A major concern of ours was that students entering a course on Computational Thinking would have low levels of motivation, particularly since the students were overwhelmingly from non-STEM majors. For them, the course fulfilled a general education requirement. To better understand the complex nature of motivation, we rely on the MUSIC Model of Academic Motivation to describe the associated dangers and opportunities [Jones 2009]. The MUSIC Model is derived from a meta-analysis of other theories of motivation, incorporating only the academically relevant components, resulting in a theory that is useful for both designing and evaluating courses. There are five key constructs in the MUSIC model:

eMpowerment:. The amount of control or agency that a student feels that they have over their learning (e.g., course assignments, lecture topics, etc.).
Usefulness:. The expectation of the student that the material they are learning will be valuable to their short- and long- term goals, especially for their career.
Success:. The student's belief in their own ability to complete assignments, projects, and other elements of a course with the investment of a reasonable, fulfilling amount of work.
Interest:. The student's perception of how the assignment appeals to situational or long-term, individual interests. The former covers the aspects of a course related to attention, while the latter covers topics related to the fully-identified areas of focus for the student.
Caring:. The students perception of other stakeholders' attitudes toward them. These stakeholders primarily include their instructor and classmates, but also can be extended to consider other members of their learning experience (e.g., administration, external experts, family, etc.).

Students are said to be motivated when one or more of these constructs is sufficiently present, as perceived by the student. The MUSIC model has been relatively under-applied in Computing Education Research, compared to particular sub-theories involving subjects such as self-efficacy or interest. We felt that when designing a course for non-majors, however, it was important to measure and build for all five components as much as possible. We revisit the MUSIC model when describing our evaluation process for students' motivation within the course.

2.3. Situated Learning Theory

We believe that the selection of a "theme" and efforts to enhance motivation should be grounded in an underlying theory of learning. A learning experience is a complex sequence of contextualized events, and Situated Learning Theory can help to guide and explain the structure of this experience. SL Theory, originally proposed by Lave and Wenger [Lave and Wenger 1991], argues that learning normally occurs as a function of the activity, context, and culture in which it is situated. Therefore, tasks in the learning environment should parallel real-world tasks, in order to maximize their *authenticity*. The key difference is that learning is driven by the problem being solved, rather than the tools available. Therefore, the problem being solved should lead directly to the tool being taught. Contextualization is key in these settings, as opposed to decontextualized (or “inert”) settings. SL Theory separates the concepts of the course Content from the course Context.

The original work in SL Theory was not about pedagogy or instructional design. It simply described how people learn and the importance of context and collaboration, but did not recommend a particular teaching style. Subsequent research by Brown [Brown et al. 1989] and others expanded the theory so that it could be applied to the design of learning experiences. These expansions often naturally dictate the use of active learning techniques, reducing the role of lecture in favor of collaborative, problem-based learning activities. There is also a strong emphasis on authenticity in assessment. Choi & Hannafin [Choi and Hannafin 1995] describe a particularly useful, concrete framework for designing situated learning environments and experiences. The Choi & Hannafin framework has four key principles: the Context – described as the “... problem’s physical and conceptual structure as well as the purpose of activity and the social milieu in which it is embedded” [Rogoff and Lave 1984]; the Content – the information intending to be conveyed to the students; Facilitations – the modifications to the learning experience that support and accelerate learning (commonly done through Scaffolds); and Assessment – the methods used to evaluate the learning experience and measure the progress of the student.

Guided by SL Theory, we felt it was important to determine a suitable context for the course that would align with our course content. [Is that really true? Did you think about the course, and SL theory, and then hunt around for a suitable theme?] As part of the overarching goal to bring more students into Computer Science, a large number of contexts (or ”themes”) have been explored in introductory computing courses. The context of a learning experience grounds the learner in what they already know, in order to teach the new material. Some introductory computing experiences focus on presenting the content as purely as possible, which has been criticized for coming across as abstract and detached [Zografski 2007]. However, starting with Seymour Papert’s work with robotics and the LOGO programming environment in the 70s [Papert 1996], instructors have explored motivating students’ first computing experience through richer contexts. As noted earlier, some of these contexts rely on Situational Interest (e.g., Digital Media “Computation” [Guzdial 2003] and Game Design [Zografski 2007]), while others attempt to provide enduring career value (e.g., [Big] Data Science [Anderson et al. 2014]) and social applicability (e.g,. Problem Solving for Social Good [Goldweber et al. 2012]). When designing our course, we felt these last two contexts would be able to support the largest range of introductory learners while aligning with our course content.

2.4. Peer Instruction

One of the more compelling and consistent results in Computer Science education is the effectiveness of peer instruction techniques [Porter et al. 2013]. Many studies and theories suggest that students learn more and become more motivated in courses that allow structured collaboration [Zingaro 2014]. Further, collaboration techniques can allow courses to scale more efficiently by reducing the burden on the course staff, as students have more human resources to draw upon. Finally, the diversity of students in the course can be leveraged to help students better understand how computing is used across different fields.

2.5. Learning Analytics in Introductory Computing

Learning Analytics and Educational Data Mining are interrelated fields concerned with tracking, organizing, and predicting student performance in order to achieve improved learning and other course outcomes. Learning analytics and data mining cover a wide range of techniques. In a recent paper, an ITiCSE working group summarized recent research in this area [Ihantola et al. 2015]. The group suggested that few pre-CS1 courses have been analyzed (11%), even though they also include high school courses in that number. Further, they note that few papers looked at Python-based courses, compared to Java courses. They compare and contrast the different statistical approaches and techniques used by the various authors. In this paper, we rely largely on descriptive, detailed, and exploratory statistical analyses, in addition to other types of computational analyses (e.g., log analyses, program analyses).

3. COURSE COMPONENTS

Although there are a number of existing Computational Thinking curricula, we have defined a particular set of educational objectives and course topics that we believe are valuable for our students. Large portions of our course were created from scratch to achieve our pedagogical goals. To support this, we have created a range of innovative technological scaffolding and have applied modern pedagogical techniques (e.g., peer instruction, active learning, etc.) that enable us to capture and catalog a large quantity of learner analytics to guide course development. [The previous sentence jumbles together technological scaffolding and pedagogy, and drops learner analytics onto both of them. Needs a major re-write.] In this section, we describe the courses' audience, content, context, structure, technology, and assessments.

3.1. Audience

Our Computational Thinking curriculum was designed to fulfill a new general education requirement at our university, to provide "Computational Thinking" to all undergraduates. Therefore, students will come from the arts, social sciences, humanities, agriculture, and more. Although some science and engineering majors enroll, it is expected that most students will not be STEM majors. The students are not expected to have prior programming experience, and the course itself has no prerequisites. Students could be at any level, from Freshman to Senior.

3.2. Course Content

The three major topics in the course are:

— **Abstraction:** The idea that real-world entities (e.g, people, objects, events, actions, processes, etc.) can be simplified and concretely represented to suit the needs of some stakeholder.
— **Algorithms:** The idea that computational abstractions can be manipulated using formal language.
— **Social Impacts:** The idea that computing provides new powers and knowledge that can influence society.

Our core learning objective is for students to be able to create an algorithm that manipulates data to answer real-world questions. Students must express their algorithms using a practical programming language – in this case, Python. Further, they must be able to navigate and describe data in contexts related to their own careers and across other fields. Finally, they should be able to discuss the social impacts, relevant stakeholders, and ethics of the questions and answers they find.

Figure 1 illustrates the hierarchical and connected nature of two of the courses' topics, Algorithms and Abstractions. Algorithms, fundamentally, describe *actions* in a specific *sequence*. Depending on the formality of the algorithmic agent, these actions can vary in complexity and atomicity. Higher-order constructs empower algorithms with the ability to branch (make *decisions*) and repeat (*iterate*). Abstractions are concretely represented as variables or *properties*, which formally have *types* and quantitative characteristics. We refer to variables as properties to avoid confusing students with

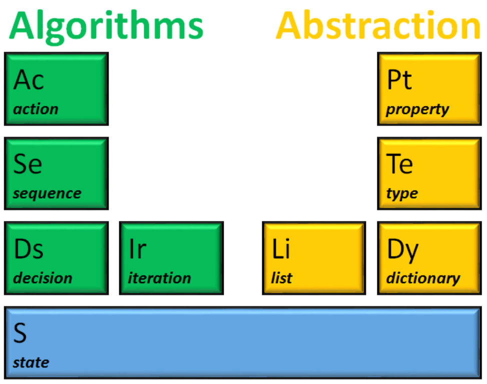


Fig. 1. Fundamental Elements of Algorithms and Abstraction

the algebraic term. Higher-order organizational structures such as homogeneous *lists* and heterogeneous *dictionaries* allow for more efficient navigation of complex data. Connecting these two ideas is *state*, the collection of properties whose structure is defined by Abstraction and whose values are used or changed dynamically during the performance of an Algorithm. Taken together, these concepts describe our ideal "notational machine" for students, unifying their understanding of control structures with the data structures being manipulated.

The topics of algorithms, abstraction, and social impacts were chosen for a combination of philosophical and pragmatic reasons. Philosophically, the algorithmic constructs in Figure 1 are embodied in the Turing machine model of computation and are elements of imperative programming languages. In addition to giving robust meaning to "computation", the algorithmic elements are sufficient for our learners to develop cognitive skills in process-oriented thinking, which we view as one component of the ”thinking” in computational thinking. Pragmatically, time constraints prevent us from including more advance programming concepts (e.g., objects, functions). Philosophically, the abstraction elements in Figure 1 carry through our commitment to presenting computation as being a tool for answering questions about the real world. These abstraction elements are sufficient to define instances of complex entities using structured collections with information-bearing properties. In addition to helping students see the world through the lens of information, the abstraction elements stimulate cognitive skills in the management of complexity through ”layers of abstraction”, which we view as the second component of the ”thinking” in computational thinking. Pragmatically, time constraints limit the number of structures that we can introduce, and the ones we did include were influenced by our choice of Python as the programming language. Philosophically, the social impacts topic encourages our learners to think about the ways in which computation shapes decisions made in the real world and influences the character of that world. This course component is in line with calls for computer science education to include the social and ethical dimensions of computing. Pragmatically, we were influenced by the requirement of our university's general education guidelines to include a components on ethics.

In the original version of the course, we also taught a unit on creating functions to encapsulate reusable chunks of code. However, time restrictions forced us to remove those lessons. Instead, modules and functions are introduced as mechanisms for reusing existing code written by other developers, without any lecture time devoted to

Module	Topics	Time Period
1	Course Overview	Days 1-4
2	Computational Modeling and Abstraction	Days 5-9
3	Algorithms and Programming	Days 10-14
4	Python and Big Data	Days 15-19
5	Mini-project	Days 20-22
6	Final project	Days 23-29

Fig. 2. [THIS TABLE IS DANGLING. It has no caption, no label, and no citation in the paper. Maybe it was somehow tied to Figure 1? But it wasn't physically part of that figure.]

creating functions. In many cases, we attempt to provide this material on functions in individual lessons with students.

3.3. Course Context

In addition to the core course content, there is also a motivating context of Data Science: extracting meaning from data using computational analyses. Our argument to students is that most disciplines benefit from data science, since the world is becoming increasingly data-driven. To that end, we have collected a large repository of datasets that can appeal to a wide range of majors (discussed further in Section 3.5.3). Students use these datasets to create visualizations (e.g,. histograms, line plots) and perform simple statistical calculations (e.g., average, maximum, counts). These kinds of analyses, while not statistically rigorous or computationally difficult, provide a range of appropriate introductory-level problems. They are tasked with interpreting these visualizations in a social context, further supporting the core learning objectives.

3.4. Course Structure

There are several distinct phases to the course: the introduction, hands-on conceptual activities, BlockPy, the Mini-project, and the Final Project.

[Is this intended to be tied to the table in some way?] Module 1 is a brief overview of the course, walking through the major topics and the cohort system (which uses peer learning). After their introduction to the course, students use NetLogo[1] and then the Blockly Maze game[2]. These two applications introduce the overarching ideas of Abstraction and Algorithms, respectively.

In Module 2, students begin learning fundamentals of computational abstractions and basic control structures, working mostly with paper-and-pencil and kinesthetic activities. In one lesson, cohorts work together to write instructions for a "card-sorting robot", executed by the course staff on playing cards in order to demonstrate the basics of algorithm design and the ambiguities of natural language. The course staff also performs a "play" that models the state of the program, the program counter, and console for a simple algorithm to process a list of data. During execution of the program, the instructor regularly stops the lesson and has students predict the next state. There are also a series of smaller pencil-and-paper activities for topics relating to decision, iteration, and abstraction.

During Module 3, students must complete a series of programming assignments in BlockPy, a block-based programming environment for Python. These assignments incorporate real-world datasets from the CORGIS project [Bart 2015], and students create charts and compute simple statistical information (averages, counts, thresholds, etc.). By the end of Module 3, students are working in the "text mode" of BlockPy,

[1]https://ccl.northwestern.edu/netlogo/
[2]https://blockly-games.appspot.com/maze

to prepare them for the transition to Module 4. This module builds on their Python experience by having them complete similar programming assignments, except this time in a professional desktop IDE named Spyder[3]. They will continue to use Spyder in Modules 5 and 6, where they complete a mini-project and then their final project. During the mini-project, students work in their cohort to answer questions about the CORGIS State Crime dataset. As a group, they create visualizations and a presentation, which they are individually responsible for presenting as a video. This is largely practice for their final project, which is a month long, individually-paced activity to create a 5-minute video presentation answering questions related to a dataset of their choosing using Python-based visualizations.

Throughout the course, most of the modules end with a day on Social Impacts. In the first module, students review final projects from previous semesters that are relevant to their career interests and must discuss and critique the social impacts of the video. In the second module, students watch a video about an ethical scenario where a computer potentially caused an accident, and they are asked to debate within their cohorts the ethical ramifications. The third module ends with a brief lesson about various ethical frameworks (e.g., Utilitarianism, Duty, Common Good, etc.) and students apply these frameworks to the debate from the previous module. In both the mini-project and the final project, students are expected to discuss the social impacts of the questions and answers that they determine.

Throughout the course, students are assigned daily readings accompanied by reading quizzes. The readings, written specifically for the curriculum by the course staff[4], introduce the material for the day with a few pages of prose and images. These quizzes are short, multiple-choice assignments that are expected to take only a few minutes. They are administered and automatically graded by Canvas. Students are allowed 3 attempts before the quiz closes

The class meets twice a week for 75 minutes. A typical day of the course begins with a short introductory lecture on the material. Students then work in their cohorts or individually to complete the interactive component (e.g., paper-and-pencil activity, collaborative discussion, programming assignment, project work). Finally, the class comes back together to discuss the lessons of the day and any challenges that occurred. Most lessons also include a required homework unit meant to reinforce students' learning and prepare them for the next class.

3.4.1. Cohorts. On the third day of the course, students are grouped into collaborative cohorts of 5-6 students. Each undergraduate teaching assistant (UTA) is responsible for two cohorts. A graduate teaching assistant is responsible for managing the UTAs, and in turn is overseen by the course instructors.

Most early classwork activities are explicitly collaborative, and students are expected to generate a single answer from their cohort. During the programming section of the course, most assignments require individual submissions, but students are allowed to work together and get help. Students are forbidden from sharing their solutions, but are encouraged to share their thought process and provide support. Near the end of the course, students complete a group "mini-project" to prepare for their individual final project. This mini-project allows a cohort to create questions and visualizations collaboratively, although they are required to submit individual presentations.

[3]https://pythonhosted.org/spyder/
[4]http://think.cs.vt.edu/book/

3.5. Course Technology

We use a variety of educational technology to scaffold student learning, including a Learning Management System, an introductory programming environment, and specially prepared dataset libraries. The online activities that students complete in the course, along with surveys, teacher observations, and interaction logs, provide a rich data source for our research questions.

3.5.1. Canvas. Although the course met twice a week in person, much of the course material was disseminated through the Canvas Learning Management System.[Need Canvas footnote.] Canvas supports the Learning Tool Interoperability (LTI) standard, so that more sophisticated tools can be embedded in the learning platform.[Need LTI footnote.] The class readings were stored as Canvas pages, which enabled us to embed interactive BlockPy canvases. [What is an ”interactive BlockPy canvas”? And is it related to Canvas?] Assignments can also include BlockPy problems. [Is ”Assignments” here means in the Canvas terminology? Subtly confusing.] Canvas has convenient support for providing feedback on student submissions, which is a major use case for our course staff. [Rewrite this last clause. What do you mean that it is ”a major use case”?]

3.5.2. BlockPy. In their first exposure to programming, students use BlockPy, a block-based environment with automatic, guiding feedback. BlockPy features a dual, bidirectional blocks-to-text coding interface that is heavily instrumented, constantly recording user interactions and code. [Rewrite. What does ”dual, bidirectional blocks-to-text coding interface” mean?] This environment facilitates the students' transition into Python, yielding a more authentic programming experience. BlockPy also integrates both real-world datasets (e.g., weather data, stock trading data) and tools for generating visualizations (e.g., line plots). Another major feature of BlockPy is the immediate feedback. When students run a program, instructor-written checks evaluate their code to guide their development (e.g., by identifying missing code constructs, incorrect output, or other anticipated errors).

BlockPy supports the LTI standard, which allows it to integrate with Canvas. Instructors embed the interactive workspaces and assignments within their courses. [How?] Students are free to work on an assignment until it is completed. Grades are passed back Canvas from the exercise upon completion.

3.5.3. CORGIS. Throughout the course, students use datasets specially designed and scaffolded for introductory computing students, with data drawn from diverse, real-world sources relevant to diverse majors (e.g., supreme court decisions, historical slave sales, incidences of diseases across the country, real-time weather forecasts, etc.). To make data conveniently available to students, we use tools from the CORGIS (Collection of Really Great and Interesting dataSets) Project. The CORGIS project makes over 40 datasets available as both raw datasets and through convenient language bindings. We use the Python language bindings and allow students to choose their own dataset.

3.6. Assessments

Our final assessment of students is a month-long project where they individually apply Computational Thinking to answer questions relevant to their own major. Students use a dataset of their own choosing, write algorithms to process and visualize the data, and then create a video report on their findings. Students are evaluated using a rubric on the quality of their report across several metrics, including the quality of their questions, their ability to explain their data abstractions, and the social impacts of their results. The programs that students create during this process are also evaluated, and the students are expected to explain automatically-extracted segments of their code

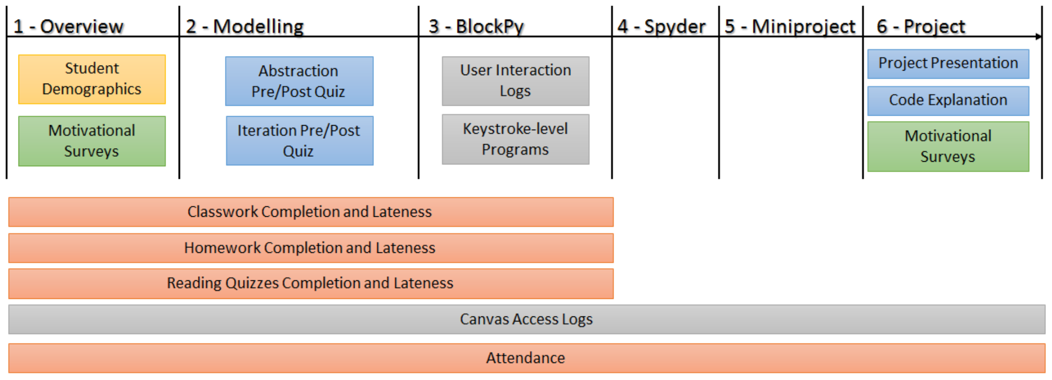


Fig. 3. Data Collection Timeline

using technical vocabulary. These two sources of data serve as ground truth of their Computational Thinking ability.

Students are assigned a final grade that is calculated based on their attendance, performance on classwork and homework, their completion of reading quizzes, and their final project scores. The weighting heavily favors effort-based work, such as their classwork and homework, but also includes their final project score.

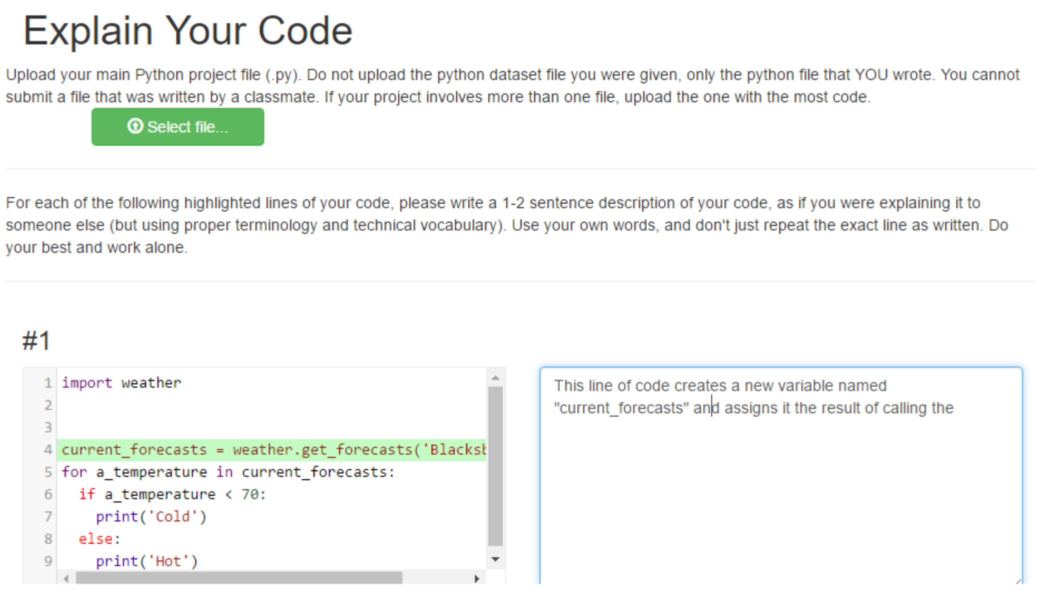
4. DATA COLLECTION

All data were collected with full compliance from our Institutional Review Board. Students were informed that data would be collected from the various computational tools. Consent was gathered using paper release forms. Only one student did not give consent to the use of their data, and so their data was removed from the analysis process. After collection, all data was anonymized, and are only reported in aggregate.

Figure 3 gives an overview of the data collection timeline. Different kinds of data are collected over the duration of the course. At the start of the semester, student demographics (orange) are collected to support our learner analysis. A pair of motivational surveys (green) are administered at the beginning and end of the course. In the second module, and then later at the end of the course, a number of cognitive assessment instruments (blue) are used to measure learner outcomes. Both throughout the course and in particular during the BlockPy module, we collect a large quantity of user interaction log data (grey). Finally, a number of effort-based classwork, homework, and other course indicators are collected during most phases of the course (pink). Little data is collected during Modules 4 and 5 because students work using their own desktop environments. [Doing what? Depending on what they are doing, it doesn't follow automatically that you can't collect data.]

4.1. Abstraction and Iteration Quizzes

Two topics that the course staff felt students particularly struggled with were Abstraction and Iteration. In the spirit of Instructional Design, the staff developed two lessons that particularly target these topics. They began the development process by identifying the key learning objectives for these topics, and used these to generate brief, 8-question pre/post quizzes for the lesson. Theses quizzes are used to measure students' initial and final understanding of the topics, allowing computation of their learning gains.



Explain Your Code

Upload your main Python project file (.py). Do not upload the python dataset file you were given, only the python file that YOU wrote. You cannot submit a file that was written by a classmate. If your project involves more than one file, upload the one with the most code.

For each of the following highlighted lines of your code, please write a 1-2 sentence description of your code, as if you were explaining it to someone else (but using proper terminology and technical vocabulary). Use your own words, and don't just repeat the exact line as written. Do your best and work alone.

#1

```
import weather


current_forecasts = weather.get_forecasts('Blacksb
for a_temperature in current_forecasts:
  if a_temperature < 70:
    print('Cold')
  else:
    print('Hot')
```

This line of code creates a new variable named "current_forecasts" and assigns it the result of calling the

Fig. 4. Code Explanation Assignment Interface

4.2. Final Project

As previously described, the final project for the course is a month-long assignment where students use a dataset [typically?] related to their career interests to conduct a computational analysis and answer questions. Once they have developed answers to their questions, students create a 5 minute video presentation (e.g., by using PowerPoint). This presentation is assessed using an 8-item rubric, which students are encouraged to reference while they are developing the presentation. The components of the rubric are as follows:

Abstraction. What is the relationship between a real-world entity and the data?
Data Structure. How is data for the project organized? What fields in the data are particularly relevant to the stakeholder?
Questions. What questions were explored?
Visualizations. How did students graphically present the results of their analysis?
Answers. What were the answers to the questions? How does the student interpret them?
Limitations. What are limitations of the data provided, and the kinds of analyses that could be performed?
Social Impacts. Who is interested in the results of the analysis, and who is affected? What potential concerns or problems can arise with regards to the analyses' impact?
Communication. Were the slides well-constructed? Did the student convey them clearly?

The complete text of the project rubric is available in Appendix A.

4.3. Code Explanations

In addition to the video, students' code is assessed using a special tool. Figure 4 shows the Code Explanation interface. Students are prompted to upload one of their Python code files (whichever one has the most code, if a student has written more than one file). The software constructs an Abstract Syntax Tree of the code and selects code lines based on presupplied criteria. In particular, the software attempts to find five code lines with examples of the following code constructs:

(1) For loop (iteration)
(2) Assignment
(3) Dictionary access
(4) Appending to a list
(5) If statement (decision)
(6) Dataset module import
(7) Plot or Print statements

These constructs are ordered so that the ones at the top will be chosen first, and selected greedily. Five lines will always be chosen, even if a single line contains multiple constructs (e.g, a dictionary access inside of an append statement). If a student submits a program with less than five lines, or with constructs present on less than five lines, all of the lines will be chosen.

The student is presented with the five lines, and charged with writing a 1-2 sentence explanation of the line using technical vocabulary. One of the instructors of the course graded all of the code explanations using a checklist. If a code construct was present in the sampled code, then students had an opportunity to gain or lose points based on correctly or incorrectly describing that construct. For instance, an assignment statement presents an opportunity for students to describe the idea of storing or setting a variable, but also presents a risk for students who incorrectly suggest that the right-hand side of the assignment is affected by the left-hand side. The complete rubric used for grading the code explanation activity is available in appendix A.3.

4.4. Surveys

Students were surveyed at the beginning and end of the course, using 26 quantitative questions and 6 qualitative questions. The quantitative questions were on a 7-point likert scale labeled from ”Strongly Disagree” to ”Strongly Agree”. The first 25 questions are a cross-product of the five core course components and the five previously described motivational aspects of the course (eMpowerment, Usefulness, Success, Interest, Caring). For the purposes of our survey, we defined the five core course components as: learning to program (content), learning to work with abstractions (content), learning about social ethics of computing (content), learning to work with real-world data (context), working in cohorts (major scaffold). So, for example, a student would be asked to rate their agreement with a statement such as, “During the course, I felt that it was useful to my career to learn how to program” or “During the course, I felt it was interesting to learn about social ethics of computing”. The 26th quantitative question asks students if they intend to continue to learn computing, either formally or informally. On the end of semester survey, students were also asked a series of 6 questions relating to the success of their cohort. These questions are based on a study done by Google that found a set of five major factors in the success of teams [ROZOVSKY 2015]. These questions ask students' perceptions of their psychological safety within the cohort, dependability of their cohort members, the structure and clarity of their assigned tasks, their perception of their cohort's sense of the usefulness of the work, and their cohort's helpfulness to themselves. [Not sure how to parse that last clause. Do you mean an individual's perception on how helpful the cohort (the cohort experience??) was to that person? Needs clarification.]

4.5. Log Data

The course was presented through the Canvas Learning Management System. After the course was completed, we obtained a detailed log of all student page accesses. This includes information not just about their time spent on course readings, but also their accesses to the assignment feedback pages. Attendance was recorded using a Canvas plugin by the Undergraduate Teaching Assistants on a daily basis.

During the middle phase of the course, students used BlockPy. We have previously published a technical description and results from a classroom pilot for BlockPy [Bart et al. 2016]. Students' interactions with with the BlockPy environment are logged at a fine-grained level.

4.6. Static Analysis

Python is a dynamic language that only gives run-time errors, as opposed to static errors. To extend our analysis of students' code, we wrote a flow-sensitive static analyzer to perform some simple analyses. Although flow-sensitivity is usually computationally prohibitive, the analyzer takes advantage of the short, simplistic nature of the types of solutions present in the course. Programs are typically less than 10 lines long and have little branching control flow. This system can detect a range of "semantic errors" that we have observed in student code:

— **Unread variables:** A variable was created but never read from, typically because a student was copying code blindly from another source.
— **Undefined variables:** The code attempts to read from an undeclared variable; a runtime error was avoided by guarding the read in an unexplored control flow path (e.g., empty for loop).
— **Overwritten variables:** A variable was written to and then written to again before it could be read from; the first write was therefore redundant.
— **Used iteration list inside body:** The iteration list (the list being iterated upon in a foreach loop) was used within the body of the foreach loop, as opposed to the iteration variable. This is never used in the types of programs students were writing, and reveals a systematic misunderstanding of the nature of iteration.
— **Did not use iteration variable inside body:** Similar to the previous problem, students will also not use iteration variable inside the body, which is only appropriate for a small subset of problems (i.e., counting).
— **Reused iteration list for iteration variable:** Although not really an error, the student has chosen the same variable name for both the list and iteration variable (e.g., `for dogs in dogs` or `for dog in dog`).
— **Attempted to iterate over a non-list:** An integer or string variable was iterated over instead of a list.
— **Iterated over an empty list:** Students often create empty lists when performing filter or mapping operations, and would sometimes be confused about iterating over the existing list vs. the new list.
— **Changed the type of a variable:** A variable initially declared to be of one type is assigned a value of a different type.

The occurrences of these errors in students' final submitted solutions is tabulated for each student and divided by the number of completed problems, in order to give a Semantic Error Rate (SER).

5. RESEARCH QUESTIONS

In this section, we describe our analysis of the data collected through the methods described in the previous section. Our primary research question is to determine which analytics can be used early in the course to predict student performance at the end of the course with regards to our cognitive, motivational, and engagement objectives. However, before we make predictions, we wanted to determine if our outcomes had any merit with respect to each other. [Huh?] We are also interested in identifying course performance in order to determine if students are achieving according to our own expectations. [Not sure what this means, either. Do you mean MEASURING course performance? And how does that relate to student expectation? What is special here?]

We begin by identifying ["Identifying" is a meaningless word here... what did you mean?] the demographics of our learners and ensuring they match our expected learner characteristics. [?? Demographics are what they are. Unless you designed the course to some demographic and missed it, I don't see what the issue is.] Next, we explore their [Who? The demographics?] absolute performance in the course to identify strengths and weaknesses in the curriculum. We then attempt to connect our course outcomes in the hopes of establishing ground truth. [Connect to what?] We then attempt to predict these [course?] outcomes with respect to different student groupings, in terms of demographics, cohort construction, and early course indicators.

5.1. Demographics

53 students were enrolled in the course after the first two weeks. At the end of the semester, 47 students received a grade (indicating a 12% dropout rate). Students were 14% Freshmen, 35% Sophomores, 31% Juniors, and 19% Seniors. This represents a roughly normal distribution of ages averaged midway through college. The class was largely balanced by gender with 46% female and 54% male. No students reported prior programming course experiences (i.e. college or high-school level courses). Building Construction was the most heavily represented major, with 36%, followed by English with 17%. Other majors, with 1-3 students each, represented a broad slice of the university including Public Relations, Political Science, Theatre Arts, International Studies, and Biochemistry. The college of Liberal Arts and Sciences had the largest share of colleges at 49%. Cohorts had 5-7 students in them, and each UTA was responsible for 11 or 13 students. [This last sentence is out of place.]

5.2. Course Performance

The two major assessment tools for the end of the course are an authentic final project and a code explanation activity. These represent students' ability to perform and communicate the results of a realistic computational analysis, and to correctly interpret and communicate the meaning of code. We also include the Semantic Error Rate of their BlockPy solutions as a metric of their ability to write code. [This last sentence seems out of place. It is not assessment of the course. So how does it tie to the first 2 sentences?]

Figure 5 shows the distribution of rubric grades for the eight components of the final project presentation. Although most of the class performed well across the components, there are noticeable deficiencies. In particular, there was a large percentage of students [Replace this with a number] who failed to describe the Abstractions used in their project, and a similar non-trivial percentage [Replace with a number] who struggled with describing the structure of their data. There are also few students who scored "Excellent" at describing their Questions and Abstractions. However, students performed well at describing the limitations of their data, the social implications, and in communicating their presentation.

Performance on the code explanation activity was relatively worse. Overall, students were unable to use technical vocabulary to describe their code, and made a number of mistakes when doing so. For instance, one student referred to "if loops", a particularly egregious example of a more general pattern. The only topics students performed well

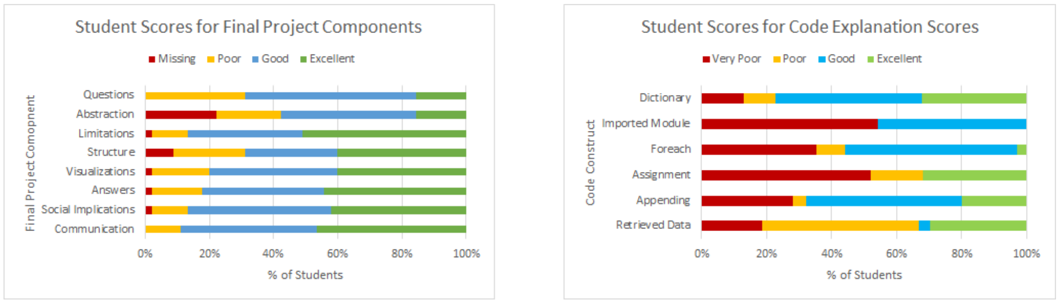


Fig. 5. Student Performance on the Final Project and the Code Explanation

on were explaining dictionary access and append statements. It is somewhat unsurprising that students performed so poorly with regards to explaining module imports and data retrieval function calls, since these topics were only lightly covered in the course material. However, students exhibited mixed performances [What does ”mixed performances” mean?] for explanations of `foreach` loops and assignment statements, despite the fact that these represent rather fundamental course topics (iteration and program state). One consideration, however, is that the code explanation rubric is less developed than other course components, making it a very raw instrument for determining student ability.

Analysis reveals that the Visualization and Answer component of the final project rubrics are possibly redundant to each other, with high correlation (.642**) [What are the ** for?] and interrelated content. [Aren't all of the measures highly correlated? Students tend to do well or poorly on nearly all measures.] Although there are a number of correlations between the scores for different code explanation constructs (Assignment was strongly statistically significantly correlated with Appending at .658** and with Foreach at .587**), we suggest it is less likely that the instrument has redundancy and more likely that it is measuring the same underlying skill to explain code, given the difference in the nature of the constructs. [Is there really anything here? What principled explanation can we give for why certain correlations matter while others do not?]

In terms of analyzing their ability to write code, we note that students had some difficulty. All students had an SER above 1, meaning that on average all students had some kind of semantic error in their solutions. The distribution of SER was roughly normal, with an average of 1.79 and standard deviation of .415. Although being able to write perfect code is not the end-goal for our course, it does represent an area where students could dramatically improve. [What would a good number be? What are we comparing against?]

Students improved measurably between [on?] both the Abstraction and Iteration topics. On both assessments, a matched-paired t-test suggests that class average went up a full point (out of the 8 points possible). [Why should we consider this good?] While an encouraging result, this leaves room for further improvement. In particular, we have to consider whether the quizzes and their respective lessons are well-aligned, which can only be done with a more comprehensive analysis of the course material. However, for now, we rely on measures of internal validity. Figure 6 shows the computed Cronbach's alpha for the quizzes: relatively low values for the items suggest little interrelation between the items. We also conducted Item Response Theory on the students results, and found that the Test Information is only high for both quizzes at around the -1 ability level; therefore, the quizzes are only particularly good at discrim-

Instrument	Pre's Cronbach's Alpha	Post's Cronbach's Alpha
Abstraction Quiz	.387	.433
Iteration Quiz	.293	.430
Interest Components	.666	.709
Useful Components	.808	.785
Success Components	.825	.727
Empowerment Components	.900	.809
Caring Components	.889	.923

Fig. 6. Cronbach's Alpha for Various Instruments [What is Pre's and Post's?]

inating very low performers from the rest. [This paragraph has too much jargon for the audience. Can this be explained more clearly?]

5.3. Outcome Ground Truth

Given the students' previously-described performance, what confidence should we have in our assessments? [I don't understand why you are raising confidence in assessment as an issue. And how does their performance affect our confidence in assessment?] In our course, we consider a number of cognitive, motivational, and engagement outcomes. The major cognitive outcomes for the course are related to the final project: their video presentation and their code explanations. We are also interested in their performance on the Abstraction and Iteration quizzes, both in terms of final score and learning gains. [Why are you interested? Or perhaps that is not the right phrasing for what you are trying to say. Do you mean that you think these are especially important for some reason?] On the motivational side, we used the responses from the beginning- and end-of-course surveys to measure students' self-reported motivation. Finally, in terms of engagement, there are a number of different outcomes: attendance, number of late assignments submitted, the amount of time spent at the Canvas course site, the amount of time spent working on BlockPy questions, and the quantity of classwork, homework, reading quizzes, and BlockPy questions completed. We attempt to correlate these outcomes with each other to determine if they support each others' conclusions.

We found a moderate, significant correlation (.510**) between students' presentation grades and their code explanation grades. It would appear that students who perform well on the final project also tend to be able to explain the code associated with their project. We find this to be a promising result, since it suggests some connection with these two different outcomes. Students' code explanation scores had a modest correlation with their SER (-.346**), suggesting a connection between students' ability to write semantically correct code and to explain that code.

There were modest correlations between students' code explanation scores and their score on the Abstraction (.376*) and Iteration (.394*) post-quizzes; there were no significant correlations between these quizzes and the students project scores, however. This suggests some separation from students' understanding of algorithmic and programming concepts versus the skills demonstrated by their project presentation. In particular, the project incorporates a number of soft skills and domain knowledge, rather than pure technical skill.

There were no significant correlations between students' performance on the Abstraction quizzes (either the before or after [What does "the before and after" mean?]) and their performance on the Abstraction component of the final project. This suggests a serious misalignment between this pair of instruments. [Why? I think that I understand (at least, they have the same name, so maybe they could be expected to be related). But spell it out.]

5.4. Engagement Outcomes

We have tracked a number of outcomes in the course that we refer to as motivational or engagement outcomes. In general, we refer to self-reported data as motivational, while engagement outcomes are collected through observation (e.g., attendance, time spent in tools). Although less important than our cognitive measures, we consider engagement and motivation to be high priorities for our students' satisfaction.

Reviewing the results of the start-of-course motivational survey, we find preliminary support for one of our major design principles: the use of data science as the learning context. We chose to emphasize the value of working with real-world data over the decontextualized learning of programming, and this decision seems to match students' interest. All numbers reported in the following are at levels on a 7-point likert scale, ranging from "Strongly Disagree" to "Strongly Agree" (0 is "Neither disagree or agree") – we interpret higher scores as students' being more highly motivated along that particular subscale. Interest in learning to program and work with abstractions was only 1.54 ± 1.31 and 1.41 ± 1.33, respectively, while interest in learning to work with real world data was $2.35 \pm .66$. Similarly, students' sense of the usefulness of learning to program was only $.84 \pm 1.54$, and to work with abstractions was only 1.43 ± 1.23, while working with real-world data was $2.33 \pm .86$.

A somewhat negative result comes from a comparison between students' pre- and post-course intent in continuing to learn computing. A matched pairs t-test suggests a significant decrease of about 1 level ($\alpha < .01, -1.03 \pm .34$) from pre- to post-course surveys. One possible explanation is that our survey question was flawed in one way or another, since it broadly covered all kinds of future learning and it may bias students [bias them how? Feels like there is a word or phrase missing?] who have a narrow conception of what computing is. Another possible, more positive explanation, is that this course helped students to an extent that they did not feel that learning more was necessary. However, it seems like this result does suggest that the course could do more to guide students in understanding how future computing courses (whether formal or informal) could benefit them. [HOw do you know that they would? Maybe they learned that they don't need to know more.] In fact, a number of other motivational attributes seem to decrease over the course of the semester. [Seems like you should state some examples here.] Further research is required to determine when and why this trend continues, and what can be done to resist it. [Why resist it?] The other questions on the survey seem to be satisfactory in terms of clarity and content, but we do note particularly high values for Cronbach's Alpha for the different MUSIC model components, which may suggest some amount of redundancy within the instrument.

Students' attendance was highly correlated with their completion of modules 1-4 (.597**), which suggests that, unsurprisingly, the students that show up to class get more work done. [Compared to what? What is the universal baseline relationship between attendance and success?] Interestingly, there was a negative correlation between attendance and students perception of the usefulness of learning computing ethics to their career (-.342*) – possibly suggesting that some students may have intentionally avoided the ethics material. Another negative (albeit weak) correlation occurred between attendance and students' perception of how useful their cohort-mates thought the material was (-.343*). [I think it would help if somewhere, maybe at the start of this section, you define what is "strong", "medium" and "weak" correlation.] A possible interpretation of this result is that students' motivation to attend class was impacted by their perception of their classmates motivation. In fact, an ANOVA calculated between students' sense of the usefulness of the material and their perception of their classmates sense of the usefulness of the material suggests a significant relationship ($r^2 = .412$**), which might mean that much of the students' sense of the usefulness of the material is drawn by their peers. Alternatively, they project their own sense of the usefulness of the material onto others. [Could do an interesting ANOVA to see if cohorts tend to correlate their performance and/or perceptions more than chance would indicate.]

One of the strongest correlations between the survey results and other course engagement outcomes is the high correlation between a student's completion of Modules 1-4 and that students' perception of how much course instructors cared that they learned to work with real-world data (.406*). Note that most other survey responses did not correlate strongly or significantly with their completion of the modules. It is curious that their sense of the instructors caring is the highest correlated, but overall suggests a theme that students perceived the course context very strongly during the modules. [Rewrite the previous sentence.] One explanation is that the motivation of the instructors had an impact on convincing students to complete problems. [I don't think that ”motivation of the instructors” is the right term.]

Student self-efficacy toward writing programs was moderately correlated with completion of BlockPy problems (.397*). Students total time spent working on BlockPy problems was negatively correlated with two different survey responses: their intent to continue learning computing (-.369*) and their perception of how useful their cohortmates thought the material was (-.443**). The former might suggest that students who were interested in continuing might also be the students who can complete problems faster; [That statement is nonsense – that's what correlation MEANS. You are looking to suggest a cause here.] the latter is more difficult to explain, but a potential hypothesis is that an implicit social pressure effect occurs where, if students believe that their peers undervalue the material, then they will devote less time to it.

5.5. Demographics, Cohorts, and Outcomes

A one-way ANOVA test was conducted between genders across the outcome variables established in previous sections. There were no significant differences between genders for cognitive outcomes such as final project score, code explanation scores, Semantic Error Rate, or any of the quizzes. However, engagement outcomes were considerably different. Women had significantly more BlockPy coding sessions where they worked for at least 15 minutes on a problem (49.96 ± 16.49 sessions), median session lengths when in Canvas (14.99 ± 7.15 hours), higher completion rates of assignments ($1.07 \pm .47$ assignments), and higher attendance (3 ± 1 days). However, it is worthwhile to note that there were no differences between genders in any of the results of the motivation surveys, with the exception of students' perception of their empowerment to work with data of their own choosing (which women very weakly rated higher than men, at a $\alpha < .05$ level). One-way ANOVA tests were also used to compare student levels (freshmen, sophomore, etc.) and course outcomes, but there were no significant differences across levels.

Another research question is whether cohorts or TAs have a direct impact on outcomes. A one-way ANOVA test was performed between the cohorts and the course outcomes. However, it appears that there were no significant differences between cohorts' performances, their self-reported motivation, or engagement behaviors. Clustering students by TA yields a similar lack of significant differences. Interestingly, UTAs were distinguishable very strongly by the amount and frequency of their feedback to students through Canvas. However, there were no statistically significant correlations between amount of feedback received and course outcomes, despite educational theory suggesting the value of feedback. This does not prove that cohorts and UTAs have no value, but it does make it more difficult to understand their impact on students' learning. Further research is required to determine whether other aspects of feedback (e.g., quality or timeliness) might have a more significant impact on student course outcomes.

5.6. Predicting Outcomes

A major research goal was to establish what outcomes could be predicted at the beginning of the semester. Student log data and other early course data was analyzed, with students grouped into ”above average” and ”below average” levels for each predictive variable. One-way ANOVA tests against course outcomes were used to identify significant predictable outcomes. Then, Cohen’s d was calculated to determine the effect size on the outcome; we report these two statistics along with the change in means and standard error.

Students who loaded an above-average number of Canvas pages (which contained the course textbook) in the first two weeks had significantly more completed assignments ($\alpha < .05, d = .73, 1.21 \pm .485$ assignments). Students who loaded the gradebook an above-average number of times in the first two weeks also had significantly more completed assignments ($\alpha < .01, d = .79, 1.30 \pm .455$ assignments). Both of these outcomes also had significance in the Final Course Grade, [Rephrase – I am not sure what this means] most likely because the Final Course Grade score is heavily based on the number of completed assignments. [Then calculate correlation between initial assignments and the rest of the course assessments.] However, students who loaded the feedback for individual assignments an above-average number of times in the first two weeks scored significantly higher on the Final Course Grade ($\alpha < .05, d = .66, 6.17 \pm 2.65$ points) but did not complete significantly more assignments. These three outcomes suggest that students who were conscientious about their classwork grades and readings early in the course were able to consistently complete more of their classwork throughout the semester. Students who loaded the feedback for individual assignments an above-average number of times in the first two weeks also had significantly longer median session lengths in BlockPy ($\alpha < .05, d = .76, 2.65 \pm .98$ minutes). Above-average page accesses and submission checks in the first two weeks were also strongly associated with total BlockPy session length ($1.61 \pm .679$ hours and $1.57 \pm .653$ hours, respectively).

Completing an above-average number of classwork assignments in Module 1 (the first two weeks) was associated with a number of improved course outcomes. Divided into these two groups, roughly one-quarter of the class (the below average students) did not complete all of the work in the first module; these students performed 13.94 ± 5.35 points lower on the final project ($\alpha < .01, d = .88$). Most importantly, these students scored roughly one level higher on the Abstraction component than students who did not ($\alpha < .01, d = .90$). Curiously, these associations did not carry over when Modules 1 and 2 were considered together. However, above average number of completed classwork assignments in Modules 1 and 2 combined was significantly associated with an increased intent to continue learning computing, at an increase of roughly one-half level on the 7-point likert ($\pm$ half a level, $\alpha < .05, d = .83$). [These two paragraphs not only make my eyes glaze over, but I’m not convinced that they mean anything.]

Semantic Error Rate was only associated [What does ”associated” mean?] with one predictive variable: attendance. Above-average attendance for the first two weeks was associated [?] with a modest decrease in SER ($\alpha < .05, d = -.78, -.32 \pm .16$ errors per solution). The significance and effect size increased as attendance is considered further out: attendance for the first four weeks had an effect size of -.88** and attendance overall had an effect size of -1.20**. Students’ code explanation scores exhibited similar behavior. Although there was no association for explanation scores with attendance in the first two weeks, attendance in the first four weeks had a modest association ($\alpha < .05, d = .76, 1.16 \pm .46$ points higher) and attendance overall had a strong association ($\alpha < .01, d = .83, 1.28 \pm .51$ points higher). [Doesn’t everyone attend in the beginning?]

Curiously, only one of the motivation pre-survey questions had a strong association with code explanation scores: above-average positive response to students' self-efficacy towards learning about the social ethics of computing was associated with a decreased code explanation score ($\alpha < .05, d = -.68, -1.05 \pm .57$ points). It is difficult to interpret this result, but a possible explanation is that students who lack confidence with respect to social ethics may make up for this with better technical ability. [That makes no sense. I can believe that they are correlated (under the argument that students good at school are just generally good at school) or not correlated (since they have nothing to do with each other). But why would they be negatively correlated?]

Previously, we found correlations between attendance and completion of classwork assignments. This is true even from the beginning of the course, as an above-average attendance in the first two weeks has a strong association with increased completion of assignments overall ($\alpha < .01, d = .89, 1.47 \pm .67$ assignments). [Sanity check: This is considering only students who completed the course, right?] There is also a strong association with decreased number of late assignments ($\alpha < .05, d = -.77, -4.35 \pm 2.23$ assignments). The total final score in the course overall was also strongly associated with attendance in the first two weeks ($\alpha < .01, d = .92, 8.57 \pm 3.46$ points). [Please remove all instances of the word "associated" and replace with something more precise.]

Looking more at the results of the motivation pre-survey, we find a particularly interesting result with respect to students' attitudes towards cohorts. In particular, three survey questions related to cohorts (sense of empowerment, sense of successfulness, and interest) were negatively associated with several course outcomes. For instance, students who expressed an above-average interest in working in cohorts scored on average 11.34 ± 4.26 points *lower* ($\alpha < .05, d = -.72$) on the final project. An above-average interest in working with a cohort was also associated with decreased average score on the abstraction pre-quiz ($\alpha < .05, d = -.74, -.94 \pm .33$ points) and iteration post-quiz ($\alpha < .05, d = -.6, -.93 \pm .46$ points). Similar results occur across these variables. [What does that mean?] A possible explanation is that students who are eager to work in cohorts may be doing so because they struggle with more individual work.

Another interesting result with regards to the motivation pre-survey is a negative association between students' sense of the usefulness of learning to program and the median session length spent in BlockPy. Note that this is not the total amount of time spent in BlockPy, simply the amount of time spent per session. Also note that these students did not complete significantly more problems. Students who reported an above-average sense of usefulness spent on average 2.8 minutes less per session, $\pm$ .97 minutes ($\alpha < .01, d = -.80$). Sense of usefulness towards learning to program was the only survey item that had a significant association, but it is not entirely clear why.

Although many of our predictive results are not surprising (students who do well on early assignments do well on later ones, higher attendance leads to more completed assignments and fewer late assignments), it is helpful to see the results borne out through data. [Why? I don't see the point, except possibly to confirm that your data collection is not broken.] It is also particularly important to note that not all predictive variables are equal to each other; some were found to have a larger effect size or more statistical significance.

6. THREATS TO VALIDITY

The largest and most serious threat to validity of all these results is the relatively small student population (N=45 students by the end). More students take the course each semester, and we expect to double the number of enrolled students in the next offering. In fact, long-term, we hope to scale this course to handle potentially hundreds of students every semester. However, in the meantime, we must be cautious about the confidence of these results. Although we feel that we do have a nice level of student diversity that is representative of future students, we recognize that further studies will be required to establish full validity. During Fall 2016 we are teaching 100 new students, and we intend to use this new data to validate our collected results.

The motivational surveys pose a threat in that they are self-reported data. Although we explain carefully to students that all results are anonymized, they are required to identify themselves on the surveys so that they can be matched with other results. It is possible that this process does not lead to completely honest answers (although we have no evidence to suggest this), and may affect our results.

Both of the major final course assessment outcomes (the final project presentation and the code explanation activity) rely entirely on human grading. Although this allows us to present students with more authentic assignments, their results become more questionable and more difficult to generalize. Future effort in developing the course must focus on proving the veracity of the rubrics used to assess the project.

7. ITERATING THE COURSE

With these predictions and confidence in our outcomes, we can answer our secondary research question: How one can design new interventions that can guide more students to success? [Where did this come from? Did I just miss it?] These interventions can be remedial lessons that target particular misconceptions, recommended interactions with the teaching assistants or professors, and improvements to tools to enhance the support that students receive. We intend to deploy a number of these changes in the next offering of the course, and continue to make improvements as we continue to scale and disseminate the course materials.

A recurring theme in our analyses of predicting course outcomes was that much of students' final performance could be predicted in the first two weeks. [And this is new how?] In general, we intend to use the data collected during this time period to help identify students who need particular attention. UTAs can be directed to pay attention to these students more closely during classwork activities, to send out more reminders on missing assignments, and give more feedback on completed assignments. In terms of course design, we intend to impose stricter deadlines on assignments: course modules will close a week after the last assignment in them is due, at which point students will no longer be allowed to make submissions. [What possible reason is there to expect that this matters either way?] We hope that this strikes a balance with the mastery learning approach to improve student self-regulation while still giving the learners adequate flexibility to complete assignments.

The biggest concerns from a course performance perspective is students' performance on the Abstraction subscale of the final project. We intend to improve each of the lessons on Abstraction in hopes of raising that score. In particular, the first lesson on Abstraction in the course uses the NetLogo environment. In the original version of the course, the entire first module was oriented around using NetLogo. However, as we increased the role of our block-based coding environment, the NetLogo piece became less and less relevant. This semester, we reduced students' time with NetLogo to a single day. In the next course offering, NetLogo will be completely replaced with a new interactive visualization tool that allows them to work with their final datasets.

In the second iteration of the course, we introduced a "mini-project" to help prepare students for the final project. Building on that success, we are now introducing a "micro-project" and "nano-project". These projects grow incrementally through the course, as students increase their computational toolkit. Our expectation is that by continually giving students opportunities to practice (and receive feedback on) the skills required for the project, they will perform better on the project.

A large category of improvements include more automated tutor features for the block-based programming environment to provide better guidance for beginning programmers—immediate, real-time feedback that can catch problematic behavior, correct students' misconceptions, and forestall their frustration before it impacts their long-term course motivation. We intend to integrate the static analyzer directly into the BlockPy environment, introducing a new class of error messages to students. Ideally, this should decrease the Semantic Error Rates and improve the quality of programs that students write.

Student performance on the Code Explanation activity calls for a more dramatic approach. In particular, there are few opportunities in the course for students to practice and receive feedback on using technical vocabulary to describe programs. New lessons and activities will need to be created and integrated to help train these skills. Currently, it is unclear where these will fit exactly.

A major limitation for scaling the course is the final project. Students create a 5-minute video that must be graded against a rubric. Presently, both teaching assistants and course instructors grade these videos at the end of the semester over a week-long period. The instructors feel that it is important for students to have two layers of grading to ensure fairness and to account for variability across graders. Although this is tractable for 50 students, it becomes an unbearable burden on the instructor for 100-500 students. A potential remedy for this is to move to a more probabilistic model. UTAs will remain responsible for grading every one of their students, while instructors will grade only a sampling of each UTAs' students (perhaps a low grade, high grade, and medium grade). The discrepancies between the UTAs and course staff can then be adjusted to normalize grades. Further, a system for appealing grades could be instituted, so that students can ask the instructor to grade their project too – with the understanding that the new grade might be less friendly.

In order to continue improving the course, we must also adjust the types of data that we collect in future iterations. Data collection so far has relied largely on automatic data logs and coursework data. As the course scales, tracking and leveraging the reports of the human course staff could be a valuable source of data about individual students, especially if it can be quantified and incorporated into formal models of student learning. In the meantime, we will continue to collect motivational (i.e., self-report surveys) and engagement data (i.e., attendance, time spent in various environments) alongside the more cognitive-related data, as this has also proven to be a valuable resource. We do intend, however, to increase the frequency of the motivational surveys to include at least one more survey in the middle of the semester. This might help us keep track of course engagement drop-off more closely.

Ideally, more assessments can be conducted throughout the course to gather more data on students' learning trajectories. For example, the earlier abstraction and iteration quizzes could be repeated to determine if students retain their knowledge about these subjects. Ideally, we would also refine these instruments to align more closely with the final course assessments. The instructors also wish to create new instruments to gather data on student learning of other topics that they have anecdotally recognized difficulties with: students' understanding of program state and how data can be organized into nested data structures, for instance.

8. CONCLUSIONS

In this paper, we have described the learning analytics we have used in our introductory computing curriculum for non-majors centered around the theme of real-world data science. These analytics have been found, in some cases, to be strong predictors of student performance even within the first two weeks of the course. We established several different forms of assessment, and gave evidence for their validity. Finally, we also described the mechanics of the course in greater detail than in previous publications. We hope that these results will be useful for those implementing introductory computing courses, and serve as a guide for those attempting to track analytics in order to improve and iterate.

9. ACKNOWLEDGMENTS

We gratefully acknowledge the support of the National Science Foundation under Grants DGE-0822220, DUE-1444094, and DUE-1624320.

A. APPENDIX A

A.1. Presentation Evaluation

The project's video presentation is evaluated by a rubric that rates as missing, poor, good, or excellent each of the following 8 statements.

(1) The presenter explained the questions to be answered and their importance.
(2) The presenter explained the role of abstraction in defining the information properties relevant to the projects questions.
(3) The presenter explained the limitations of the available data in answering the questions.
(4) The presenter explained the structure of the data.
(5) The presenter explained the meaning of the visualizations used to answer the questions.
(6) The presenter explained the answers to the questions based on the visualizations.
(7) The presenter explained the social implications of the project.
(8) The presenter communicated effectively.

A.2. General Guidelines

In all cases a rating of Missing is given if the required element is not recognizably present. It is the responsibility of the presenter to present each element clearly.

An individual required element may be presented over multiple slides and a single slide may relate to multiple required elements.

It is not important in what order the required elements are addressed in the presentation. However, the presentation should have a logical sequencing that makes the presentation understandable and easily relatable to the rubric elements.

Here are detailed descriptions of these elements.

Specific Explanations

(1) The presenter explained the questions to be answered and their importance.

Element: The presenter demonstrated problem-solving skills by describing the questions the project is intended to answer and identifying the significance of these answers to some group.

Rating:
— Poor: states the questions and claims that answering these questions is important.
— Good: states the questions and gives an argument for the importance of answering the questions.
— Excellent: states the questions and an argument supported by evidence (expert opinion, examples, economic factors, etc.) for the importance of answering the questions.

The presenter explained the role of abstraction in defining the information properties relevant to the projects questions.

Element: The presenter demonstrated an understanding of the concept of abstraction by explaining the information properties used to model the real-world entities relevant to the projects questions.

Rating:
— Poor: gives a definition of abstraction and names the real-world entities.

— Good: defines abstraction and describes characteristics of the real-world entities.
— Excellent: defines abstraction, identifies specific information properties of the real-world entities, and explains how these properties are relevant to the projects questions.

The presenter explained the limitations of the available data in answering the questions.

Element: The presenter demonstrated quantitative reasoning by identifying factors that restrict the scope (comprehensiveness, completeness) or accuracy (precision) of the answers that can be obtained from the available data.

Rating:
— Poor: General factors related to the data are described without clear indication of their impact on the answers.
— Good: Specific factors are identified but without clear indication of their impact on the answers.
— Excellent: Specific factors are identified and the impact of these factors on the answers is clearly explained.

The presenter explained the structure of the data.

Element: The presenter demonstrated an understanding of data structures by explaining the technical organization of the data.

Rating:
— Poor: A diagram or other visual representation of the datas organization is presented and only partially or confusingly described.
— Good: A diagram or other visual representation of the datas organization is presented and well described using general language.
— Excellent: A diagram or other visual representation of the datas organization is presented and described using correct technical language.

The presenter explained the meaning of the visualizations used to answer the questions.

Element: The presenter demonstrated the ability to analyze complex data by explaining the form, content, and significance of the generated visualizations that answer the projects questions.

Rating:
— Poor: Inappropriate, incomplete or ambiguous visualizations are presented whose relationships to the questions is not clear.
— Good: One form of visualization appropriate to the data and the questions is used. Visualizations are not always appropriate titled or labelled. The relationship of the visualizations to the questions is reasonably clear.
— Excellent: Multiple forms of visualizations appropriate to the data and the questions are used. Visualizations are appropriately titled and labelled. The relationship of the visualizations to the questions is clearly identified.

The presenter explained the answers to the questions based on the visualizations.

Element: The presenter demonstrated a quantitative reasoning ability by explaining how conclusions based on the visualizations answer the projects questions.

Rating:
- Poor: The implications of the visualizations are not explained or are misinterpreted.
- Good: The implications of the visualizations are clearly explained but not directly related to the answer for each question.
- Excellent: The implications of the visualizations are clearly explained and directly related to the answer for each question.

The presenter explained the social implications of the project.

Element: The presenter demonstrated an awareness of the social impact of computing by explaining how the results of the project could have positive or negative effects on individuals, groups, or society.

Rating:
- Poor: states generally that there are social impacts
- Good: identifies appropriate stakeholders and specific impacts that may be experienced by these stakeholders.
- Excellent: identifies stakeholders, impacts, and explains possible conflicts among the stakeholders.

The presenter communicated effectively.

Element: The presenter demonstrated oral and visual communication skills by presenting the project in an effective manner.

Rating:
- Poor: The presentation was of inappropriate length for the content or the presenter was not clearly understandable.
- Good: The presentation was of appropriate length, the presenter was clearly understandable, but the slides lacked some degree of organization or clarity.
- Excellent: The presentation was of an appropriate length; the presenter was clearly understandable, and the slides conveyed information in an organized and clear manner.

A.3. Code Evaluation

The Python code developed to analyze the data and produce the visualizations is the second major outcome of the project. The code submitted for the project will be evaluated in the following way. Up to five lines of code will be selected. The selected lines of code will typically involve a significant aspect of the computational. For each selected line of code the developer will be asked to provide a short explanation of the meaning and purpose of this code in the computation. Each explanation will be assigned a rating of missing, poor, good, or excellent according to this rubric:

- missing: a meaningful explanation is not given.
- poor: a partial explanation of the technical meaning of the code using general language is given with no or only a vague description of the purpose of this code in the overall computation.
- good: a partial explanation of the technical meaning of the code using correct terminology is given together with a general, though not specific, description of the purpose of this code in the overall computation.
- excellent: a thorough explanation of the technical meaning of the code using correct terminology is given together with a clear description of the purpose of this code in the overall computation.

An explanation which otherwise meets the requirements for a rating will be downgraded if the explanation also contains incorrect or irrelevant statements, or is inappropriately long.

REFERENCES

Ruth E. Anderson, Michael D. Ernst, Robert Ordóñez, Paul Pham, and Ben Tribelhorn. 2015. A Data Programming CS1 Course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. 150–155.

Ruth E. Anderson, Michael D. Ernst, Robert Ordóñez, Paul Pham, and Steven A. Wolfman. 2014. Introductory Programming Meets the Real World: Using Real Problems and Data in CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. 465–466.

Austin Cory Bart. 2015. Situating Computational Thinking with Big Data: Pedagogy and Technology (Abstract Only). In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. 719–719.

A. C. Bart, J. Tibau, E. Tilevich, C. A. Shaffer, and D. Kafura. 2016. Implementing an Open-Access, Data Science Programming Environment for Learners. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 1. 728–737.

John Seely Brown, Allan Collins, and Paul Duguid. 1989. Situated cognition and the culture of learning. *Educational researcher* 18, 1 (1989), 32–42.

Jeong-Im Choi and Michael Hannafin. 1995. Situated cognition and learning environments: Roles, structures, and implications for design. *Educational Technology Research and Development* 43, 2 (1995), 53–69.

Stephen Cooper and Wanda Dann. 2015. Programming: A Key Component of Computational Thinking in CS Courses for Non-majors. *ACM Inroads* 6, 1 (Feb. 2015), 50–54.

Michael Goldweber, John Barr, Tony Clear, Renzo Davoli, Samuel Mann, Elizabeth Patitsas, and Scott Portnoff. 2012. A Framework for Enhancing the Social Good in Computing Education: A Values Approach. In *Proceedings of the Final Reports on Innovation and Technology in Computer Science Education 2012 Working Groups (ITiCSE-WGR '12)*. 16–38.

Google Inc. 2011. Exploring Computational Thinking. (2011).

Mark Guzdial. 2003. A Media Computation Course for Non-majors. *SIGCSE Bull.* 35, 3 (June 2003), 104–108.

Olaf A. Hall-Holt and Kevin R. Sanft. 2015. Statistics-infused Introduction to Computer Science. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. 138–143.

Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H. Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, Miguel Ángel Rubio, Judy Sheard, Bronius Skupas, Jaime Spacco, Claudia Szabo, and Daniel 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.

ISTE & CSTA. 2011. Operational Definition of Computational Thinking. (2011).

B. D. Jones. 2009. Motivating students to engage in learning: The MUSIC model of academic motivation. *International Journal of Teaching and Learning in Higher Education* 21, 2 (2009), 272–285.

Dennis Kafura, Austin Cory Bart, and Bushra Chowdhury. 2015. Design and Preliminary Results From a Computational Thinking Course. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 63–68.

Jean Lave and Etienne Wenger. 1991. *Situated learning: Legitimate peripheral participation*.

Saad Mneimneh. 2015. Fibonacci in The Curriculum: Not Just a Bad Recurrence. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. 253–258.

Seymour Papert. 1996. An exploration in the space of mathematics educations. *International Journal of Computers for Mathematical Learning* 1, 1 (1996), 95–123.

Christopher Plaue and Lindsey R. Cook. 2015. Data Journalism: Lessons Learned While Designing an Interdisciplinary Service Course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. 126–131.

Leo Porter, Cynthia Bailey Lee, and Beth Simon. 2013. Halving Fail Rates Using Peer Instruction: A Study of Four Computer Science Courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. 177–182.

Alexander Repenning. 2013. Making Programming Accessible and Exciting. *Computer* 46, 6 (June 2013), 78–81.

Barbara Ed Rogoff and Jean Ed Lave. 1984. *Everyday cognition: Its development in social context.*

JULIA ROZOVSKY. 2015. Re:Work - The Five Keys to a Successful Google Team. (17 Nov 2015).

Henry M. Walker. 2015a. Computational Thinking in a Non-majors CS Course Requires a Programming Component. *ACM Inroads* 6, 1 (Feb. 2015), 58–61.

Henry M. Walker. 2015b. Priorities for the Non-majors, CS Course: Programming May Not Make the Cut. *ACM Inroads* 6, 1 (Feb. 2015), 46–49.

A. Weinberg. 2013. Computational Thinking: An Investigation of the Existing Scholarship and Research. In *School of Education*.

Jeannette M Wing. 2006. Computational thinking. *Commun. ACM* 49, 3 (2006), 33–35.

Daniel Zingaro. 2014. Peer Instruction Contributes to Self-efficacy in CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. 373–378.

Zlatko Zografski. 2007. Innovating Introductory Computer Science Courses: Approaches and Comparisons. In *Proceedings of the 45th Annual Southeast Regional Conference (ACM-SE 45)*. 478–483.