Towards a Data-Driven Analysis of Programming Tutorials' Telemetry to Improve the Educational Experience in Introductory Programming Courses

by

Anna Russo Kennedy
B.A., Concordia University, 2006

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Towards a Data-Driven Analysis of Programming Tutorials' Telemetry to Improve the Educational Experience in Introductory Programming Courses

by

Anna Russo Kennedy
B.A., Concordia University, 2006

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Melanie Tory, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Yvonne Coady, Supervisor
(Department of Computer Science)

Dr. Melanie Tory, Departmental Member
(Department of Computer Science)

ABSTRACT

Retention in Computer Science undergraduate education, particularly of under-represented groups, continues to be a growing challenge. A theme shared by much of the research literature into why this is so is one of a distancing in the relationship between Computer Science professors and students [39, 40, 45]. How then, can we begin to lessen that distance, and build stronger connections between these groups in an era of growing class sizes and technology replacing human interaction? This work presents BitFit, an online programming practice and learning tool, to describe an approach to using the telemetry made possible from deploying this or similar tools in introductory programming courses to improve the quality of instruction, and the students' course experiences. BitFit gathers interaction data as students use the tool to actively engage with course material.

In this thesis we first explore what kind of quantitative data can be used to help professors gain insights into how students might be faring in their courses, moving the method of instruction towards a data- and student-driven model. Secondly, we demonstrate the capacity of the telemetry to aid professors in more precisely identifying students at-risk of failure in their courses. Our goal is to reveal possible reasons these students would be considered at-risk at an early enough point in the course to make interventions possible. Finally, we show how the use of tools such as BitFit within introductory programming courses could positively impact the student experience. Through a preliminary qualitative assessment, we seek to address impact on confidence, metacognition, and the ability for an individual to envision success in Computer Science. When used together within an all-encompassing approach aimed

at improving retention in Computer Science, tools such as BitFit can move towards improving the quality of instruction and the students' experience by helping to build stronger connections rooted in empathy between professors and students.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Taylor, Isabella and Josephine for your love, support and patience.

Dr. Melanie Tory and Dr. Joe Parsons for your insightful comments and edits to my thesis draft, and for your participation at my thesis defense.

Dr. Yvonne Coady and Anthony Estey for your many contributions to this research, for mentoring me throughout my time in grad school, and for making it a whole lot of fun.

DEDICATION

To Taylor, the most wonderful husband and partner I could ever have hoped for. This has been a very long journey, and very much a team effort. Thank you.

Chapter 1

Introduction

Computer Science (CS) is a rewarding, exciting, diverse and interesting field, yet there is a well-known and growing shortage of workers in Canada to fill jobs in computing [19]. One way to address this shortage is to increase retention rates in university CS programs. In particular, increasing retention of underrepresented groups such as women (who make up only 12.9% of undergraduate CS students [46]) could further contribute to filling these labour gaps. While enrolment in CS undergraduate programs has steadily risen in recent years [46], Cohoon and Chen [28] report attrition as being a serious issue facing university CS programs since the early 1990s, if not earlier. Recent reports put attrition rates in CS programs as high as 66% [34]. How, then, can we improve retention in undergraduate CS programs, particularly for underrepresented groups?

There are many possible reasons for attrition in CS. What many theories have in common is a "distancing" in the relationship between professors and students. White [45] reports how increases in class sizes, increased reliance on web-based learning, and evaluation of teachers based on research output rather than instructional quality or performance have negatively impacted the relationship between students and teachers, and students' experiences in university. Large classrooms were found to act as barriers to building meaningful relationships between teachers and students, contributing to a feeling of "mass education" that leads students to perceive lecturers as wanting them to be merely passive attendees at their university education, rather than active participants. Seymour and Hewitt [39] studied reasons why students leave the sciences, pointing to numerous studies that have found the "apathetic environment" in STEM classrooms to be the most frequently cited reason students leave. This chilly environment is typified by a feeling that faculty do not care about

students and that students are not "gifted" enough to be successful in these fields. Cohoon [10] examined departmental conditions that cause students to leave CS, and concluded that when faculty believe student success to be a shared responsibility, attrition rates — particularly of female students — are lower. Tinto [40] states that faculty engagement, especially in first-year courses, is critical to enhancing student retention. Yet even when faculty wish to be a part of improving the student experience in their courses, the reality of large class sizes and teaching loads makes it more difficult for instructors to understand how students are faring in their courses, and to connect with those that most need their help.

Overall, the problem seems to be one of distance — a lack of connection between students and faculty. This connection problem manifests in three ways: first, it is difficult for faculty to understand how students as a whole are faring in their courses; second, when students are struggling and at risk of failing courses, professors do not have ways of finding out early enough to be able to provide meaningful interventions to assist these students; finally, student experiences in courses are negatively impacted when they believe faculty don't care about their success. We know of the importance of CS departments having faculty who care about the success of their students, and who make that fact known in their teaching, mentoring and relationships with students. In addition to faculty considerations, student *experiences* such as confidence, motivation and belief in their ability to succeed in their CS courses and program are critical issues that need to be addressed. Taken together, the research shows that improving retention, particularly of underrepresented groups, can be achieved by encompassing an approach that focuses on building empathetic, bi-directional connections between professors and students in the CS classroom.

To address the three issues described above, this thesis presents BitFit, a web-based programming practice and learning tool for introductory programming students that allows them to work through a number of problem-based active learning activities that coincide with each week's course content. We describe our deployment of BitFit during the winter 2015 semester of CSC 110 *Fundamentals of Programming I* at the University of Victoria. The tool collects fine-grained interaction data, allowing instructors to gain insight into how students are faring with the course material, to revisit problem areas before progressing onto new material, and to answer student questions. In an introductory programming course where topics build upon each other, it is important to identify and support students that fall behind early. The data collected by a tool like BitFit can provide support for instructors to do this.

To support student confidence and metacognition, the tool provides a pressure-free environment to investigate the material, step-by-step hints, instantaneous feedback, and support for further questions.

We describe how including a tool like BitFit in courses can be an integral part of an encompassing approach aimed at improving retention in CS. From the professors' point of view, this means an approach that provides avenues for them to understand how students are faring in their courses and aims for early intervention when at-risk student warning signals arise [9]. For the students, this means a teaching team that demonstrates they care about the success of their students [10, 39]; that provides them safe places to try and to fail with the material [4]; that finds ways to answer student questions [42]; and that shows students they do belong in the field of CS [6]. Including a tool like BitFit in courses helps to show that instructors do care about student success, and contributes to fostering meaningful connections between instructors and students.

1.1 Research Questions

This leads us to the research questions that drive this thesis and arise out of the problem statement from above:

Research Question 1: What data could be gathered by a tool like BitFit to help instructors offer a better learning experience to students?

Research Question 2: What data could help instructors more precisely identify at-risk students, why they are at risk, and allow for more effective interventions?

Research Question 3: Can the use of a tool like BitFit offer students an improved learning experience?

1.2 Agenda

The following provides a map of the remainder of this thesis.

Chapter 2 examines related work in the areas of learning analytics and data collection in programming practice tools, other, similar programming practice tools, and identifying at-risk students.

Chapter 3 presents the values that guided the design of BitFit, and introduces the design requirements that rise up from these guiding principles. The final section discusses the architecture of BitFit.

Chapter 4 describes the implementation of BitFit from the points of view of the two main user groups: students and instructors.

Chapter 5 presents the raw data gathered during our observational study and deployment of BitFit.

Chapter 6 presents an analysis of the data and answers the research questions.

Chapter 7 enumerates avenues of future work for further development of BitFit and its applications, and concludes with some closing remarks.

Chapter 2

Related Work

In this thesis we present BitFit — the Programming Practice Tool, an experimental prototype of a framework to help move towards automated, data-driven analysis of student learning and course data with the goal of improving student experience in introductory programming courses. Tools such as BitFit can allow us to move from a simple "canned quizzing system" to an Internet scale model that continuously assesses and improves the quality of the educational experience [9].

Learning to program is hard [27]. Introductory programming courses are typically taught in the first year of an undergraduate CS degree, and positive student experiences and outcomes in these course would directly impact student retention rates for the remainder of their degrees. The following sections show how a tool like BitFit that leverages active learning [20], continuous, bi-directional feedback, scaffolded help systems [42] and fine-grained, detailed interaction logging [9] not only can help students achieve mastery learning [7], but also can help to improve their overall experience in introductory programming courses. This telemetry is considered as being synergistic to traditional feedback mechanisms, such as grades on labs, assignments and exams. This chapter also shows how use of tools like BitFit can help teachers improve the quality of instruction and intervention for struggling students in introductory programming courses.

2.1 Background: Data Collection and Learning Analytics

Instructors benefit from early indicators of potential problems students have with understanding course material, plus accurate ways to gauge the difficulty levels in their courses. It is important that instructors be aware that students are struggling with material as soon as possible.

The research literature contains a number of examples of collecting data in introductory programming courses to improve student outcomes. Jadud used BlueJ, a beginner's programming environment, to report at compile-time the complete source code along with other relevant meta-data [22], and observed that a small number of different syntax errors account for the majority of errors. More recent work [41] has proposed using BlueJ to gather even finer granularity of data as students work through programming assignments, on a much larger scale. As an extension to BlueJ, Norris et al. used the ClockIt BlueJ Data Logger/Visualizer to monitor student software development practices. Students and instructors were then able to view the collected data, the goal being to discover "patterns" of student practices shared by successful students that are not present in struggling or unsuccessful students [30].

Retina also records compilation attempts and compiler error data [29], along with a student ID and a timestamp. Instructors can later access the aggregate student data, while each student can later view information about their own monitored activities, and compare their data with that of the rest of the class. A novel feature in Retina is that it provides students with recommendations based on data collected as students progress through a homework assignment: when Retina determines a recommendation is in order (due to a high rate of errors per compilation, or the same error made multiple times), a message is sent to the user so the student is able to get a better understanding of what to do next.

Edwards et al. [14] compared effective and ineffective programming behaviours of student programmers, using assignment submissions gathered from an automated grading system. Submissions were classified by grade received, and the significant finding of the study is that when students started and finished their assignments earlier, they received higher grades than when they started an assignment the day before it was due, or later.

Koedinger et al. [24] wrote a compelling piece addressing the goal of improving online learning and course offerings through data-driven learner modeling. They rec-

ommended designing online learning environments that collect fine-grained, complex data about a learner. This will enable researchers to use well established principles on learning and cognition, and avoid re-inventing the wheel when it comes to evaluating and improving the learning experience. They stressed the importance of learning from the decades of research that have been carried out in the Intelligent Tutoring Systems and Artificial Intelligence in Education fields, and recommend including cognitive psychology expertise to guide the design of online learning activities. They also strongly recommended adopting a data-driven approach to learner modeling with the goal of improving the learner's experience, and shifting course design away from solely an expert-driven paradigm to one that is self reflective and learns from past interactions of learners with the system.

Robling et al. [38] provided recommendations and consider pedagogical implications of extending current Learning Management Systems (LMS) to better support the needs of Computer Science programs. One of their suggestions was to integrate systems of "drill and practice" with automatic assessment into current LMSs. However, the authors noted that of one of the risks of including such systems is that, since they don't have any type of structured hint system, students may become discouraged due to lack of feedback if they are struggling with any of the material. They recommend building systems with the ability to provide layered feedback to students as a way of raising student motivation. This is one of the strong motivating factors for the hint system and built-in question support at an exercise level in BitFit.

These types of data monitoring features are important to both students and instructors, and something we have begun to incorporate into BitFit. Similar to the tools above, BitFit collects data about time-on-task, plus compilation and runtime data, though at a coarser granularity (currently only numbers of compiles and runs, though future work is planned to capture compiler errors as well - see Chapter 7). One of the significant differences of BitFit to the systems described above is the scaffolded hint system, and the data collected about how students use these hints. One of our driving goals in building BitFit is to provide a feedback loop for instructors to act upon. The data gathered by the tool plays a critical role in creating this opportunity for moving towards supportive, dynamic, student-driven learning paradigms.

2.2 Identifying At-Risk Students

How do instructors and researchers identify and intervene to help students who struggle? Falkner and Falkner [15] sought ways of identifying at-risk students early enough to provide meaningful interventions, while minimizing overhead to faculty. They found that the timing of a student's first assignment submission in a CS course — particularly late submissions — was a strong predictor of which students were likely to under-perform in their classes. Another data driven approach analyzed logins to a course's Learning Management System (LMS), exam grades and clicker points (as a determinant of attendance in lectures) to identify and intervene with at-risk students [13]. Riordan and Traxler [37] studied the impact of using blended technologies such as SMS messages to student cell phones when students were identified as being at risk of failing. Ramesh et al. [35] developed a data-driven approach to modeling student engagement and success in MOOCs (Massive Open Online Courses). Through analysis of student interaction and language used in forums, plus other variables, the authors were able to accurately predict which students would stay engaged through to the end of the course, and which would achieve successful completion certificates.

Kurtz et al. report that data on novice programmers collected by their tool, WebIDE, showed concerning behaviour in at-risk students: students not struggling would complete tasks in the system with only two or three compile attempts. Struggling students, however, would approach the questions without a strategy to solve them, and end up with 20 or 30 compiles, and very little time between each compile.

2.3 Programming Practice Tools

The tools described in this section share a number of traits with BitFit: online environments that need no special software or system, and require very little overhead on the part of the student. Their aim is to give the student access to a simple, straight-forward environment providing lots of opportunity to practice doing the act of programming.

CodingBat [33] is an online tool for students to practice small coding problems. Each question asks a student to write a single Java (or Python) method that takes some parameter values, does a computation, then returns a result value. Student code answers are placed inside a frame that is compiled and run against several test cases when the student checks their answer. If there are compile time errors, the system

displays these; if not, the pass or fail results of the test cases are shown. The goal is for the student to write the methods in such a way as to pass all of the test cases. Some of the warm-up problems have solutions available within the tool, while many do not. Many links to videos or other resources are provided alongside each question, should a student need additional assistance.

WebIDE [23] is a scalable, open framework for creating, hosting and rendering labs with rich, rapid feedback. The WebIDE architecture is centred around the concept of Test Driven Learning, where students must demonstrate their understanding of a question by first writing small test cases before writing the implementation code. The platform provides a simple, web-only programming environment that novice students can use from day one.

CodeWorkout [9] provides multiple choice and coding questions for students to work through. Student answers to coding questions are run against test cases, and based on these results, students are provided with hints tailored to where their code fails tests. What is novel in this system is that after a student successfully answers a question, she has the opportunity to write a hint to help future students solve the problem. After solving a sufficient number of questions, students can author their own questions in that topic area, thus demonstrating mastery of the topic [7], and contributing to the body of learning exercises. The tool evaluates the effectiveness of hints as well as the contributed exercises to continually improve the system and its effectiveness for learners.

CloudCoder [31] is an open source platform for creating and sharing short programming exercises. Motivating its design is the authors' desire to create a programming practice environment that is completely free to adopt, with very low overhead for professors wishing to use it. The system allows the student to practice short programming exercises in an online environment, while collecting fine-grained metrics on student interaction, in order to study how students learn to program. The authors have also created a repository of programming exercises that can be freely shared amongst professors using the tool. Further details on the CloudCoder platform can be found in Section 3.3 of the chapter on BitFit Architecture, where this framework is used to compare and contrast to BitFit's infrastructure and design.

Chapter 3

BitFit Architecture

This chapter discusses the overall design goals of BitFit, and compares and contrasts these with other, similar systems. Sections 3.1 and 3.2 describe the values driving the construction of BitFit, while Section 3.3 dives deeper into the specifics of BitFit's architecture and framework.

3.1 Design Goals for Improving the Quality of Instruction

In order to encourage widespread adoption of BitFit beyond the single institution at which it was first created and deployed, we chose to make the whole of the project open source [1]. As detailed in Section 2.1 on Data Collection and Learning Analytics, we wished to build a tool that would collect fine-grained data about the students' interaction with the material. These data needed to be able to show, on a high level, how the students as a whole are faring with the course material, as well as interactions at an individual student level to enable instructors to precisely identify at-risk students early enough to make meaningful interventions.

These factors lead to the initial core design requirements (DR's):

Design Requirement 1: Build an easy-to-deploy, scalable and extensible open source tool using current, industry-standard open source technologies.

Design Requirement 2: Collect fine-grained student interaction data.

[1] The BitFit source code is available on GitHub at https://github.com/annark/coding-app

3.1.1 Design Decisions in Support of DR1 & DR2

BitFit is an open source tool. The reasons behind this choice were two-fold: first, any kinds of strings attached or payments required would be likely to dissuade instructors from using the tool. Adoption must be lightweight and require little overhead on the part of the instructor. Second, the tool was built using some of the industry's latest development tools, with the goal of attracting future graduate students and/or community members interested in building upon their skills in these areas.

BitFit is built in JavaScript via Node.js, using the MEAN framework (see Section 3.3 for details). We made the choice to adopt the Node.js framework because BitFit is a web-based tool, and Node.js specifically was built to handle the operations of a web application. Node.js employs an event loop, which is a single thread that per-

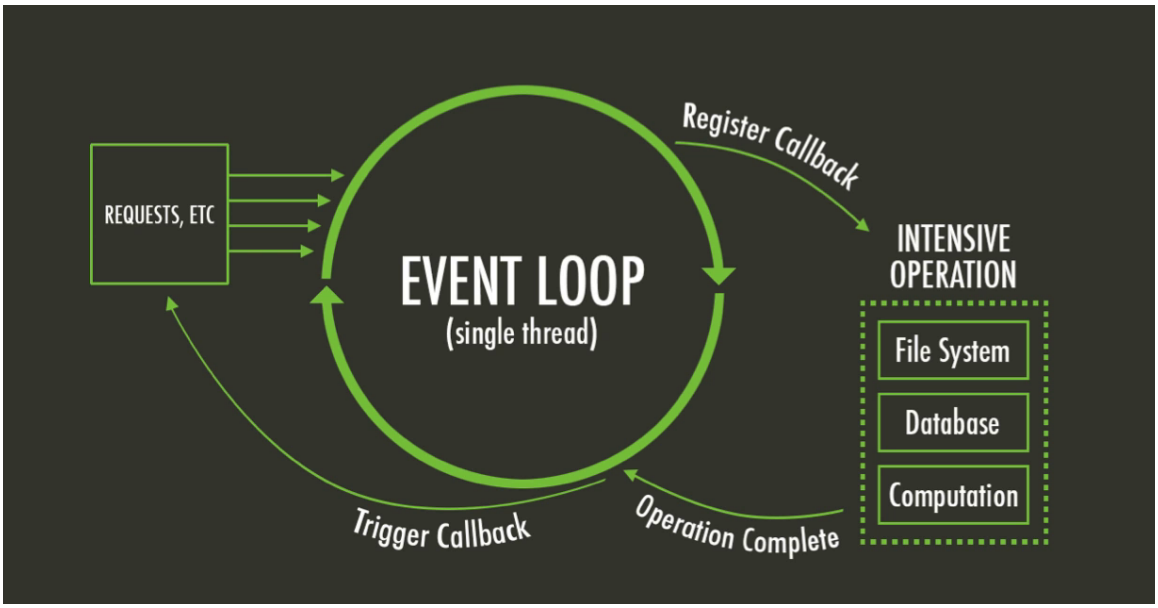

Figure 3.1: Node.js Event Loop [11]

forms all I/O operations asynchronously (see Figure 3.1). In addition to using less memory and being simpler to program than traditional methods of I/O, this event loop means that a node application, when faced with performing an I/O operation such as reading/writing to a data store or reading/writing to a network connection, simply dispatches an asynchronous task along with a callback function to the event loop, then continues execution of the remainder of the program. When the asynchronous operation is complete, its callback is invoked and the application finishes any processing required for the I/O task. The importance of this asynchronous I/O

event loop is that these operations, which occur very frequently in web applications, can happen very quickly. This means that Node.js applications are capable of handling a huge number of simultaneous connections with high throughput [16]. In other words, BitFit's framework is capable of scaling to handle simultaneous use by many hundreds (or more) of students (DR1).

BitFit is designed to be easily extensible. It is straightforward for an instructor or researcher to add functionality to the tool, say to allow users to practice exercises in a different programming language — C++ for instance. To do so, they would only need to add a function in the Command Line Interface module, which would call upon a C++ compiler rather than the Java compiler. Even the structure of the persistent data store was chosen with flexibility in mind. MongoDB is a noSQL datastore, with one major bonus of its design being that changes can be made to existing schemas on the fly, without having to first stop and then restart an application for the changes to take effect. This means that students could be working through exercises at the same time as a developer is adding improved functionality — for instance, further support for Test Driven Development (see Future Work chapter for more examples) (DR1).

Throughout a student's use of BitFit, interaction data are collected and stored on a secure server. These data include numbers of compiles and runs for code writing questions, numbers of hints requested, time spent on each exercise, and total correct versus incorrect attempts to answer questions. Details of data collection, and how it is used to derive insights to improve the quality of instruction and student experiences, are described in Section 5.1.5 of the Results Chapter (DR2).

3.2 Design Goals for Improving the Student Experience

It has been well established that the careful introduction of tools such as BitFit into introductory programming courses can have a positive impact on student academic outcomes (see, for example [23, 18, 17]). However, that is only a subgoal of our motivation for building the tool. With BitFit's design and implementation we sought ways to improve the student *experience* in introductory programming courses. This forms part of a greater strategy aimed at building empathy between students and teachers, with the goal of improving retention rates.

Active learning [20] encourages student engagement through active participation

in the learning process. Astin's student involvement theory [5] suggests that increases in student engagement may lead both to improvements in student performance and increases in student satisfaction levels. Papert's theory of constructionism [32] posits that people learn by creating tangible objects — understanding through constructing. Resnick [36] notes that computing-based learning provides a way for learners to actively create and manipulate these kinds of artifacts.

Lahtinen et al. [26] conducted an international survey of over 500 professors and students on the difficulties faced by novice programmers. Respondents, when asked about the learning situations they found most useful, described activities that require learning by doing. Lahtinen et al. recommended active learning be a fundamental part of introductory programming courses all the time.

VanLehn defined an *Interaction Hypothesis*, which states that the greater the degree of interactivity in a computer assisted learning environment, the greater the learning on the part of the student [43]. This notion was corroborated by another study [8], which recorded one-on-one tutoring sessions and investigated whether greater tutor initiative or greater student initiative during these sessions impacted student learning outcomes. The authors found no significant differences between the approaches observed, yet noted the mere fact of the high level of interactivity between tutor and student, as compared to lecture formats, positively impacted student learning.

While studies have shown that syntactically simpler languages such as Python can lead to improved student performance in early programming courses [25], many institutions still teach Java as a first language. The benefit of BitFit is that it allows instructors to strip away the burden for students of writing much of the more complicated syntax in the language by providing that as starter code, allowing students to focus on the small problem at hand.

Hoffman et al. [18] presented a web-based tool that not only focused on learning programming by doing, but specifically aimed to improve code reading ability - a term the authors call Active Code Reading. Hoffman et al. argued that while computer science students write a lot of code, the quality is often poor, as a result of students not fully understanding what the code they are writing or adding to actually does.

Introductory programming courses are typically built around a framework that introduces fundamental programming topics that build upon those learnt in previous weeks [25]. If a student fails to master one or more of the early concepts, they will have significant difficulty in later portions of the course that focus on using these

fundamental topics in concert with each other to problem-solve.

Vihavainen et al. [44] described an approach they called Extreme Apprenticeship, that combines the practices of Extreme Programming (wherein the best practices of software development are taken to an extreme level), with Active Learning. Their approach uses guided programming exercises to promote learning by doing, continuous bi-directional feedback between student (apprentice) and teacher (master), and a "no compromise" approach where the skills are practiced for as long as the individual needs, until the apprentice becomes a master. The authors stress the need to learn the craft of programming by actually doing programming, and plenty of it.

The above paragraphs provide the motivation for the first of the student-focused values we sought to encompass by building and deploying BitFit. These give rise to the next set of core design requirements (DR's):

Design Requirement 3: Create a programming practice environment requiring very low student overhead to use.

Design Requirement 4: Build a space for students to actively engage with the material: allow plenty of opportunity to practice both code writing and active code reading.

Design Requirement 5: Foster an environment of high interactivity in the tool, to build a continual feedback loop between student and instructor.

Design Requirement 6: Design a tool where questions are broken down by topic area.

3.2.1 Design Decisions in Support of DR3 through DR6

Students access BitFit via a webpage in a browser. All that is required is an Internet connection and their login information. BitFit is system and platform independent, and abstracts away all the environment details so students can dive straight into the actual act of programming from any machine or device (DR3). In other words, BitFit encourages students to train the routine of programming: carry out a high number of interactive exercises that are somewhat repetitive, in order to master the skill needed to tackle bigger problems elsewhere [44] (DR4). Students interact with and receive feedback from BitFit in multiple ways (DR5). The types of interaction include reports on compile and run time errors, step by step hints, and instantaneous feedback. These

points are described in detail in the next section, Section 3.2.2. Finally, addressing DR6, we built BitFit to offer as much flexibility as possible to instructors in how they structure the programming exercises. Instructors create a new topic, provide some background information on the topic area, and create a set of questions to be associated with that topic. Our goal was to allow for an environment where students can practice both individual, building block programming skills, as well as work on topic areas that build upon the ones that came before.

Another important finding of Lahtinen's study showed that students overestimate their understanding of programming concepts [26]. In other words, students think they know the material better than they actually do. Students need ways to accurately gauge their true understanding of course materials, in real time.

Dempsey et al. [12] sought to understand different factors that influence men and women students to pursue computer science. Interestingly, they found that two factors — whether the student felt they were able to use CS techniques to problem solve (self-efficacy); and whether the student saw themselves as a computer scientist (identity) — emerged as markers for intention. Females reported having lower CS self-efficacy and CS identity than males. Thus, the authors conclude that initiatives to increase women's participation in CS might have more success if they also focus on changing the way women perceive themselves in the CS field.

A central component of many programming assistance tools and Intelligent Tutoring Systems is a hint framework [42]. Our goal with the hint system of BitFit is to allow the instructor to construct a sequence of hints that provide a step by step method of reaching the correct answer — a so-called scaffolded hint system. A guiding principle for this type of sequence is the Point, Teach and Bottom-out approach [21, 42]. Pointing hints seek to remind students of knowledge they already possess that will help them to solve the problem. Teaching hints describe the knowledge pointed at in the previous hint, and show how to apply it. Bottom-out hints tell the student what to do next.

Code Workout [9] provides hints to students based on the results of running their code attempts against test cases. One novel feature of Code Workout is that after students have correctly answered a question, they are given the opportunity to provide a hint for future student attempts at that question, thereby improving the system and moving closer towards mastery learning of the material [7].

Vihavainen et al [44], in their Extreme Apprenticeship approach to teaching programming, view the process as similar to that of a master teaching an apprentice

their craft. One phase of this approach is called the scaffolding stage, where students are given exercises devised by the master to practice their skill. Support is then given to students in the form of scaffolding, whereby students are given just enough hints to be able to figure out the answers on their own.

Another guiding principle for including a hint system was that we want to ensure a student doesn't ever feel "stuck" with the material, at a loss for what to do next. While it is up to individual instructors using BitFit to decide the number and types of hints they will provide for each question, with our deployment and testing of BitFit, we chose to create hints that gradually pointed students towards the answer — "next-step hints" [42], without ever actually providing a "bottoming-out" hint, or giving the students the actual answer. Based on student feedback, however, in future we plan to add functionality for BitFit to provide the correct answer after the student has exhausted all available hints for a question, plus made some minimum number of attempts to solve the question on their own (see Chapter 7, Future Work).

These points make up the last of our motivating values, and give rise to three final core design requirements:

Design Requirement 7: Provide support for all student questions.

Design Requirement 8: Offer a safe place for students to try and fail with the material.

Design Requirement 9: Provide a way for students to accurately and rapidly gauge what they do and don't know, letting them become capable of seeing themselves succeeding with the material.

3.2.2 Design Decisions in Support of DR7 through DR9

BitFit was built to offer support for all student questions via a scaffolded hint system and additional *Ask a Question* buttons associated with every programming exercise (DR7). Our system allows students to work through as many or as few questions as they wish, with no limit to the number of times they may attempt exercises (DR8). Instantaneous feedback is provided in multiple ways: on student answers in the form of direct feedback when students wish to check their answers; through a *Compile Output* dialogue box if student code contains errors; through a *Run Output* dialogue box if infinite loops or other runtime errors are detected in their code. This rapid

feedback helps students to gauge areas they have mastered or where they need to do some more work (DR9).

3.3 BitFit Framework

BitFit is an open source, web-based programming practice tool, implemented in JavaScript using the fullstack MEAN framework. MEAN stands for *MongoDB*, the backend persistent NoSQL storage; *Express*, a Node.js web application framework; *AngularJS*, a frontend framework; and *Node.js*, a platform built on Chrome's JavaScript runtime. BitFit uses a client-server architecture where students work through programming exercises in a browser (the client), which sends their answer attempts and logging data to the server that hosts the application and database. There, the server compiles and runs their code, capturing the standard output and standard error streams and displaying those in the client browser to help students with debugging. In terms of security, the system has strict limits on size of user programs, input and output, and protects against infinite loops or excessively long-running programs with a timeout function. If a student's code has an infinite loop or runs beyond the maximum time limit, the system will interrupt their program, inform the student of the type of error, and advise them to check their code for issues. As well, to access any of the tool's functionality, a user must be logged in to the system. As a measure of added security, user accounts can only be created by the administrator.

CloudCoder [31] is an open source platform for creating and sharing short programming exercises. It is the tool in the existing literature on the subject that shares the greatest number of similarities with BitFit, differing mostly in the long term goals of the project and motivations behind the tool. Table 3.1 summarizes the points of comparison between CloudCoder and BitFit. Given these similarities and the open source nature of CloudCoder, one might well wonder why we chose to create yet another programming practice tool. The reasons are that in order to meet the full set of design requirements we recognized as essential to the success of BitFit, we considered both the options of extending CloudCoder, or developing a brand new tool. Because several of our requirements demanded changing the core architecture of CloudCoder, extending it would have required longer development cycles in an already tight timeline.

What is new about BitFit, and that differs significantly from CloudCoder, is

that 1) The hint system is built into the tool as a core part of the architecture, resembling more closely the ITS systems described above than does CloudCoder, yet requiring much less overhead than an ITS would to set up and deploy; and 2) Code reading/tracing questions, similar to the types of questions presented by CQG are a fundamental part of the tool's core architecture, in addition to code writing questions. Figure 3.2 demonstrates how BitFit sits at the intersection of the various tools discussed in this section.

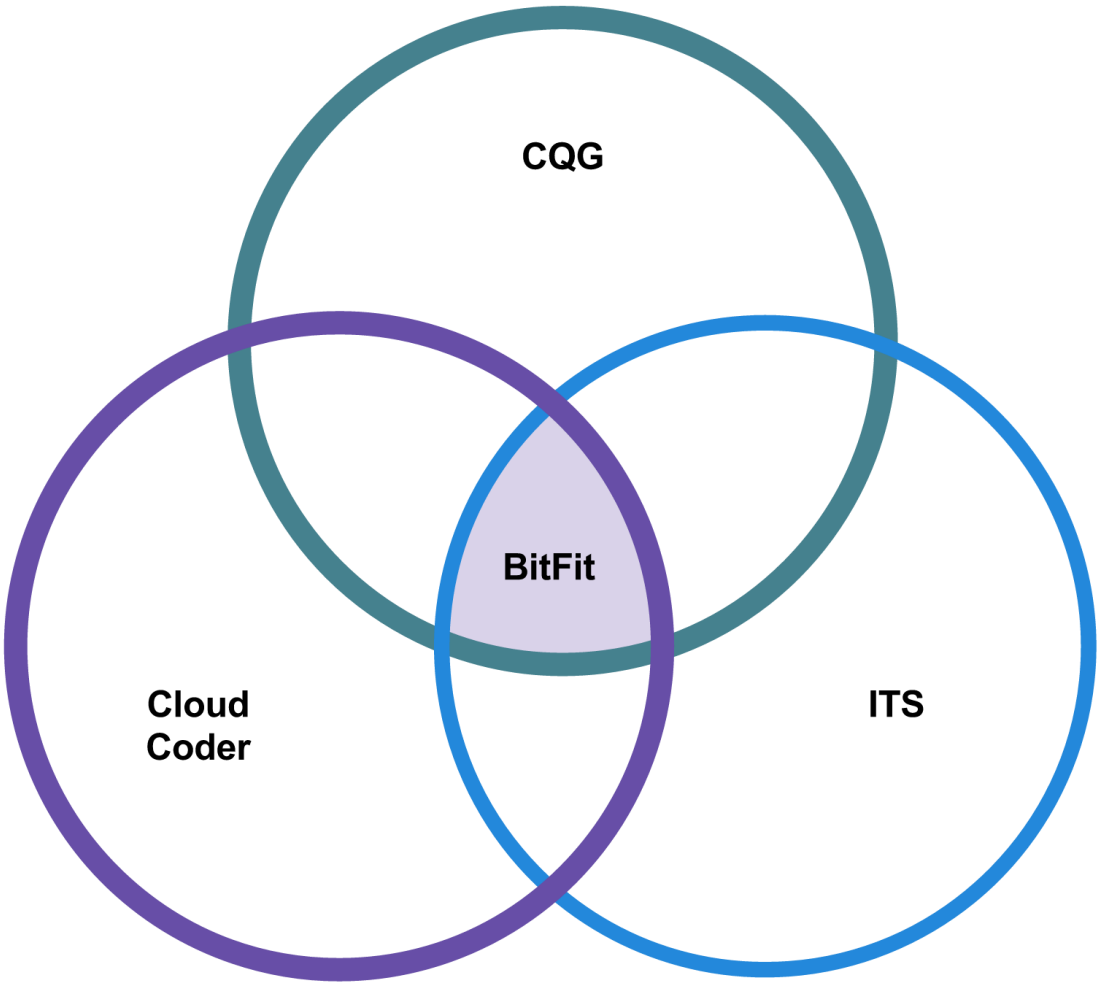

Figure 3.2: BitFit in relation to similar tools

The following paragraphs highlight details from Table 3.1, placing them in context of the two tools, and giving examples, where necessary, of the implications of the design decisions that lead to these features.

As detailed above, BitFit was developed using the Node.js and MEAN framework. In contrast, CloudCoder was built using the Java-based Google Web Toolkit. Both BitFit and CloudCoder are open source, and employ the Ace code editor in their frameworks. One important feature of CloudCoder that is currently not available in BitFit (though planned in future work, see Chapter 7) is an exercise repository. This allows instructors using CloudCoder to share exercises amongst themselves, thus making the difficult task of creating effective programming exercises a collaborative effort. To check student answers, CloudCoder runs student code against a suite of test cases for each programming exercise, while BitFit tests code-writing questions against an expected output, as provided by the instructor. While this means BitFit does not specifically provide test cases in the way CloudCoder does, BitFit does allow instructors a great degree of flexibility: they are able to build test driver programs within the body of the starter code, which, once the student completes their answer, will perform in a similar manner to CloudCoders test suites.

If updates or changes to any of the existing database tables is required, the CloudCoder application (which uses MySQL) must first be shut down completely. In contrast, updating or changing BitFit's MongoDB schemas can be done on the fly, without having to shut down the application first.

	BitFit	**CloudCoder**
Development Platform & language	MEAN stack;Node.js	Google Web Toolkit (GWT);Java
Data store	MongoDB	MySQL
Open Source	Yes	Yes
Integrated Hint System	Yes	No
Code Tracing Questions	Yes	No
Exercise repository	No	Yes
Code editor used	Ace	Ace
Code runs against test cases	No	Yes
Starter code allows for built-in testing program	Yes	Yes
Standard output/error streams displayed for debugging	Yes	Yes
Students can tell if they have made partially correct attempts	No	Yes
Types of exercises supported	Function based; Whole program	Function based;Whole program
Languages supported	Currently: Java;Planned future work: C/C++, Python	Java, C/C++, Python, Ruby
Security	Users must login to use system, accounts can only be created by admin;Size and time limits imposed on student code through Node.js	Users must login to use system, accounts can only be created by admin;Java Security Manager
Data Collection	Amount of time spent on each question attempted-Number of compiles, runs and hints requested; Correct attempts vs. total attempts; Most recent code attempts for each question	All student edits to code; All student code submissions; All test outcomes
Exercises divided by topic	Yes	No

Table 3.1: Comparison of BitFit and CloudCoder Architectures

Chapter 4

BitFit: A User's Perspective

There are two types of users of BitFit: students; and administrators (instructors). The following sections describe the details of the user experiences for these two different user groups.

4.1 Student Interface

Figure 4.1 shows the introductory screen students are taken to after logging in to the system. The list of available topic areas is shown on the left hand side of the screen. Once a student selects a topic, they are taken to the set of questions available for that topic. At any time whilst working through the exercises, students can select the *Background* tab — as the snippet for the Loops topic background shows in Figure 4.2 — should they wish to revisit or review some background information for that topic.

BitFit uses the Ace code editor [1], an open source web-based editor for programming, that supports syntax highlighting, keyboard shortcuts and automatic indentations for most programming languages. There are two types of questions available to students in BitFit: code writing and code reading. Both types of questions have an instance of the Ace editor embedded into their user interface for students to work with or view code.

4.1.1 Code Writing Questions

Figure 4.3 shows the user interface students see when working on a code writing exercise. Students are asked to complete a short task — typically writing a method that takes some number of parameters, does a computation, and outputs and/or

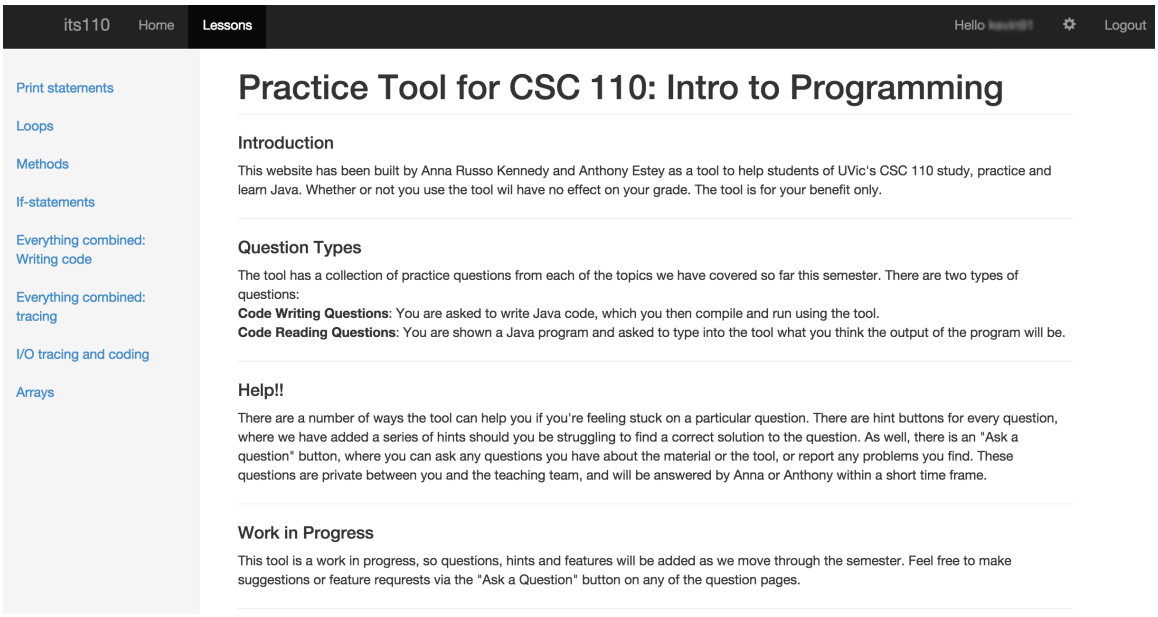

Figure 4.1: BitFit Introductory Screen

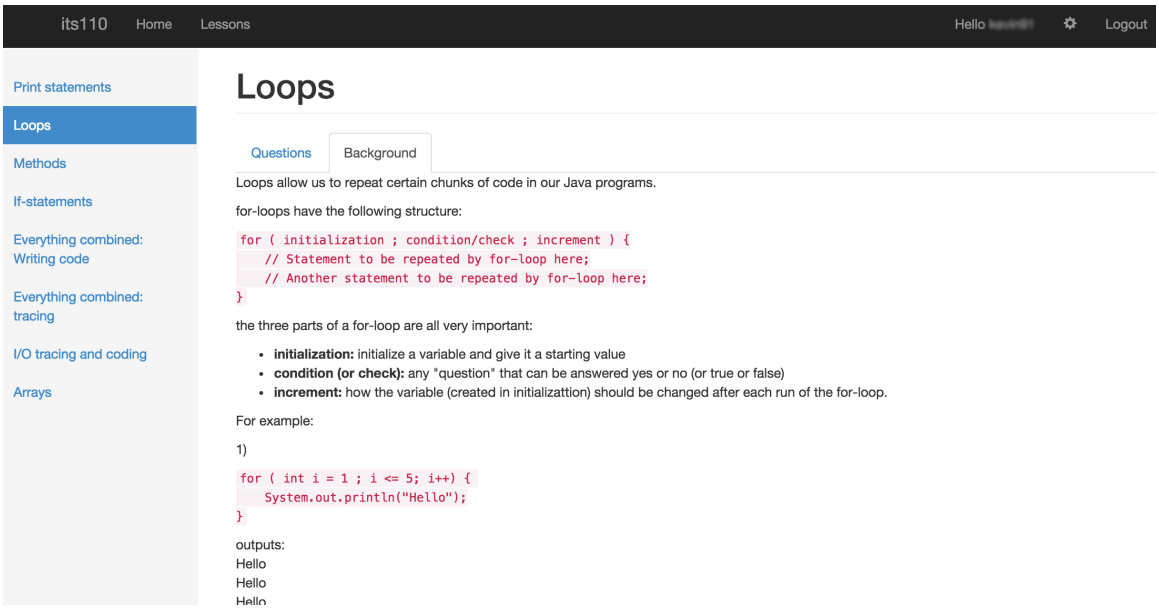

Figure 4.2: BitFit Background Tab for Loops Topic

returns a value. Students may be given starter code that they must add to (though this is the instructor's choice — the code editor may also be blank to start, requiring students to write a complete Java program) to solve the problem. When a student clicks the *Compile Code* button, their code is sent to the server which attempts to compile it. Any errors are piped to the *Compile Output* box, as shown in Figure

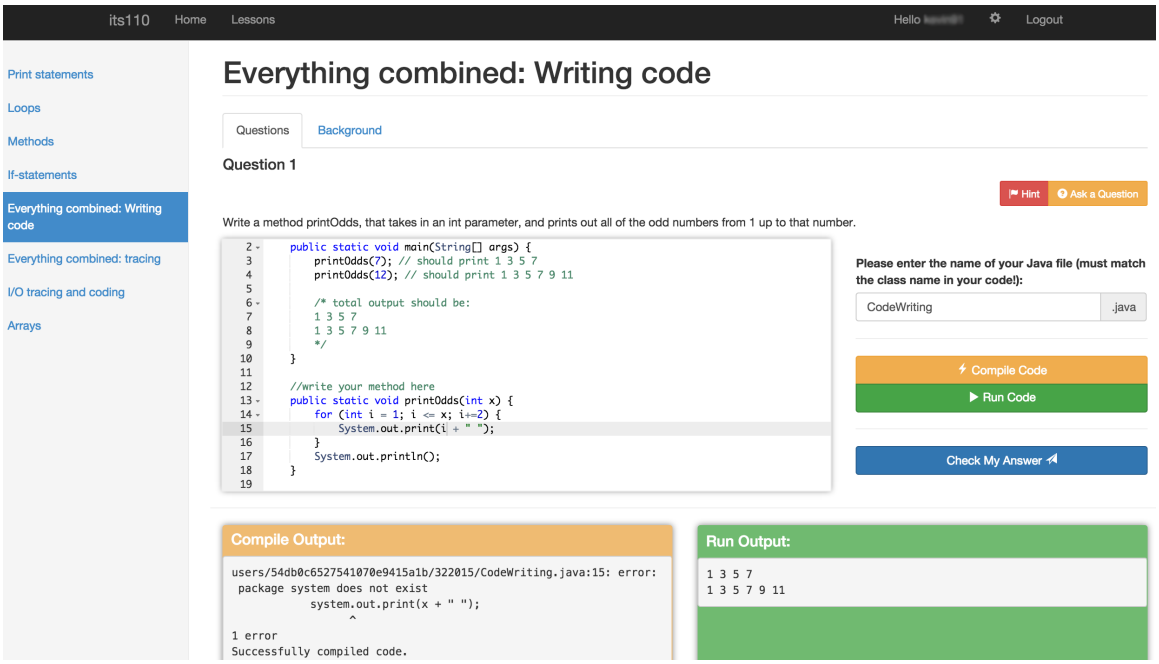

Figure 4.3: BitFit Code Writing Exercise

4.3. Clicking the *Run Code* button makes the system attempt to run the executable file on the server, and displays the standard output stream of the student's program in the *Run Output* box. Finally, students can click *Check My Answer* at any time, which will compile and run their code, then check their program's output against the expected output for that question. Students are shown a dialogue box informing them if their answer was correct, or if they need to try again.

Figure 4.3 also demonstrates how the current iteration of the BitFit system can be used to create Output Driven learning [44] style questions. Output Driven Learning is similar to Test Driven Learning, in that it forces students to first think about what the question is asking them to do, before diving into code and potentially getting lost in the syntax of the code. Future work on BitFit will see support for adding multiple test cases to each question, along with an evaluator, should the instructor wish (see Chapter 7, Future Work).

4.1.2 Code Reading Questions

Figure 4.4 shows the interface for a code reading exercise. For these types of questions, students are presented with some read-only code (i.e. they cannot edit the code, and the code is a complete Java program), and asked what the code's output will be.

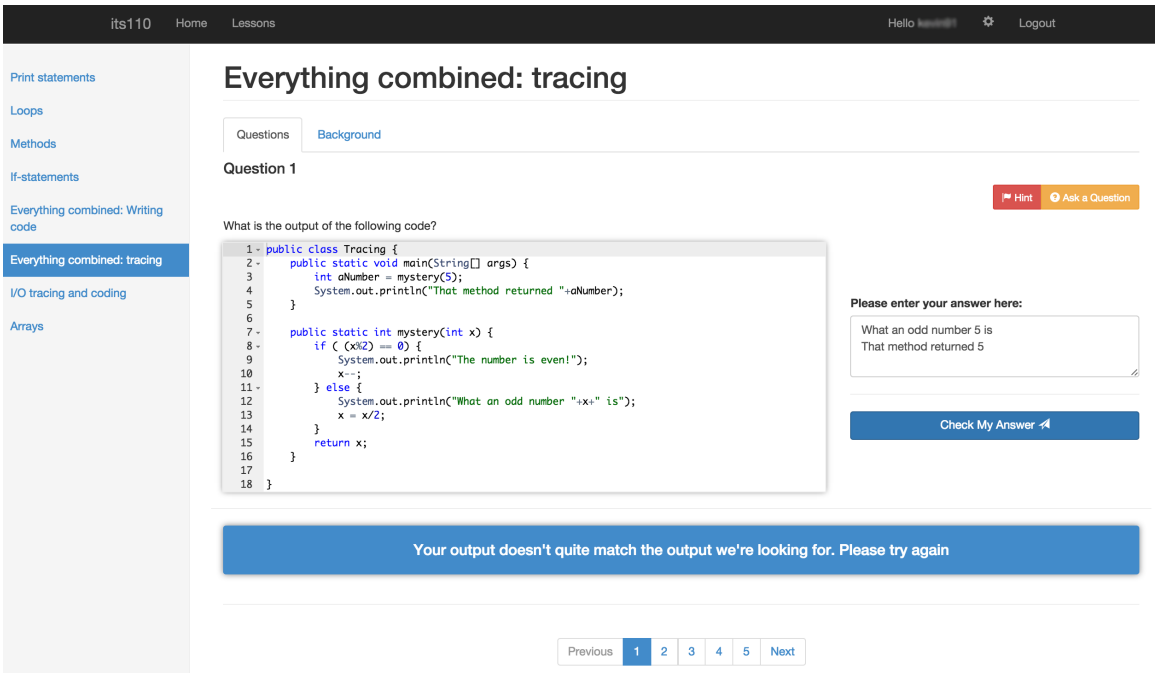

Figure 4.4: BitFit Code Reading Exercise Incorrect Attempt

Students type their answer into the box provided and click *Check My Answer*, at which point the system sends the code to the server to be compiled and run. The system compares the student's guess with the actual output of the program, displaying the appropriate message to students. Figure 4.4 shows an incorrect student attempt at answering a code reading question.

Students may attempt to answer both types of questions as many times as they wish. Should students be stuck, however, the system provides several types of additional supports.

4.1.3 Hints

Every question has a *Hint* button which, when pushed, will display a hint underneath the body of the question (see Figure 4.5). Instructors are free to add as many hints as they wish for each question. Each subsequent click of the *Hint* button displays another hint under the previous one, until all hints have been exhausted for that question.

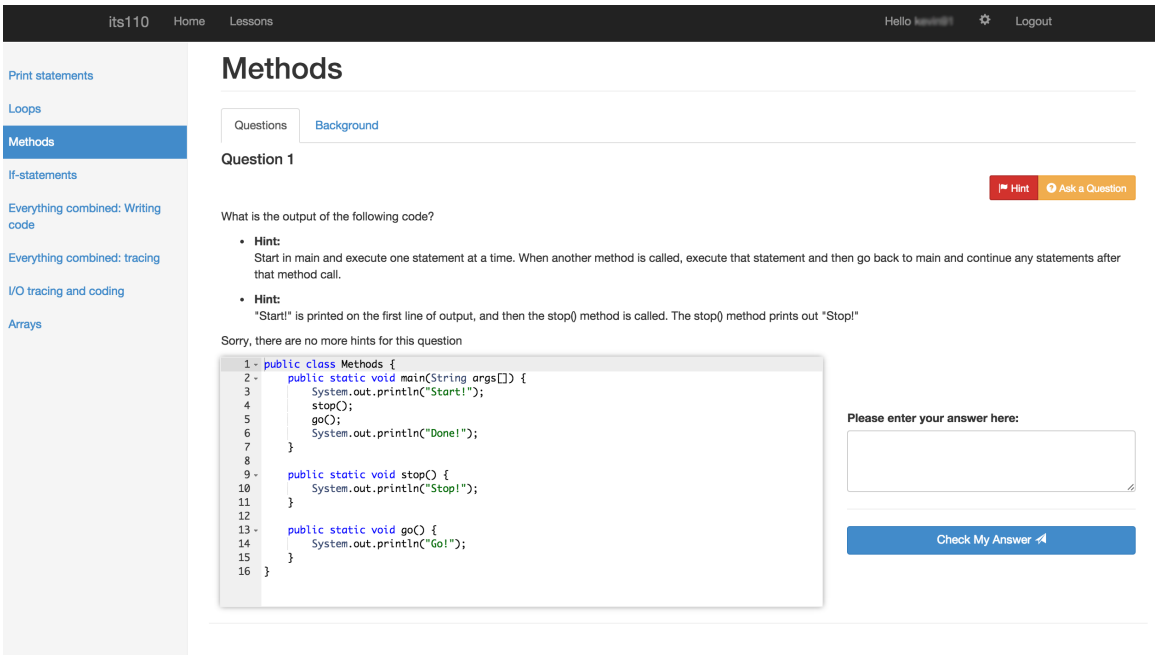

Figure 4.5: BitFit User has requested the maximum hints available for a question

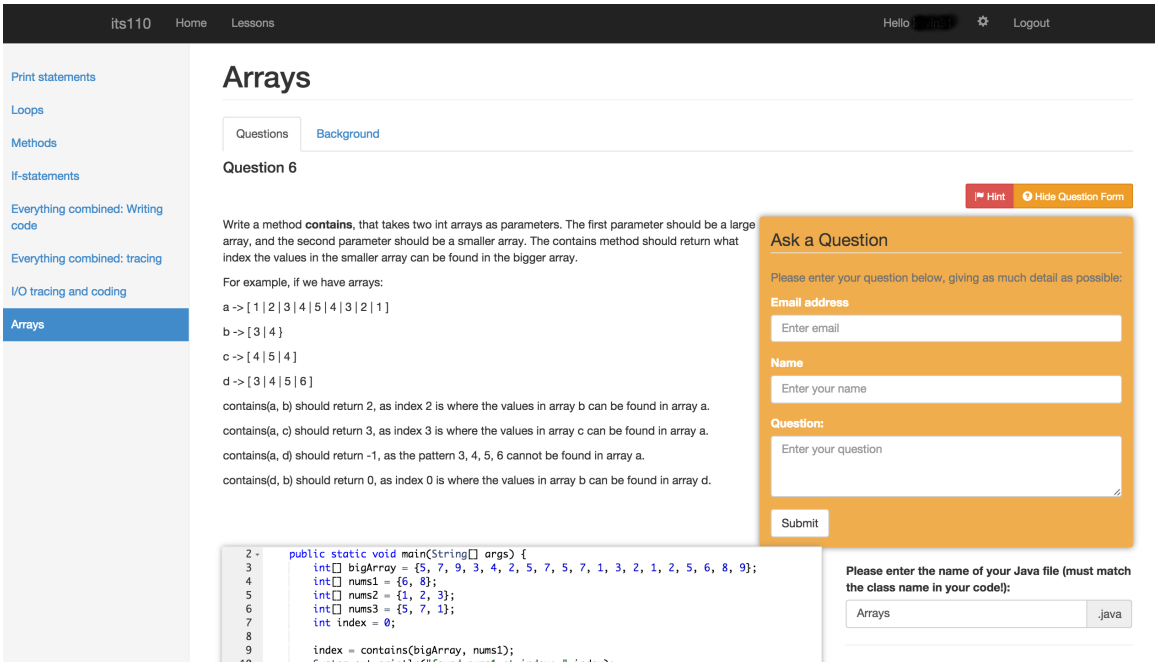

Figure 4.6: BitFit Ask a Question form

4.1.4 Ask a Question

If a student has questions beyond what is covered by hints, at any time, on any question page, they can click the *Ask a Question* button. Doing so will display a

small form for them to fill in their question as shown in Figure 4.6, which gets sent, along with the current question students are working on, to an administrative-only area to be answered by an instructor.

In future work we plan to implement a feature that will allow students to view graphs of their progress through the tool, as well as aggregated data on the rest of the class's tool usage. This will strengthen the tool's ability to help students to see how they are faring with the work — to bolster their metacognition, or knowing about what they do and don't know. See Chapter 7 Future Work for details.

4.2 Instructor/Administrator Interface

When an instructor launches an instance of the BitFit application, they set up their administrative user account, which allows them, upon logging in, to enter a protected area of the tool's system. In this admin-only section, instructors can manage user accounts, and add or edit topics and questions. Future work will include a section for viewing live and historical student interaction data with the tool (which currently is only calculated offline — see the Results in Chapter 5 for details).

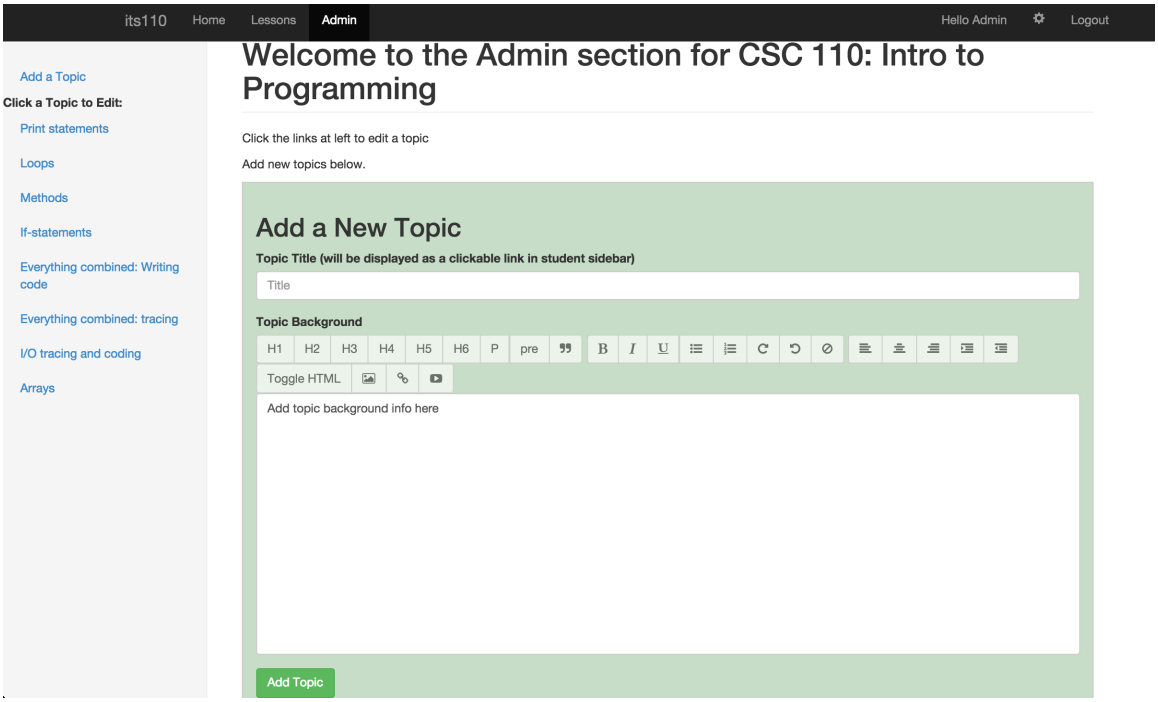

Figure 4.7: BitFit Admin — Add/select topic

Figure 4.7 shows the user interface an administrator sees upon entering the admin-

only area (accessed via the *Admin* menu item along the top navigation bar, only visible to the administrator). Here, administrators may add or edit topic titles and background information. An HTML editor is provided to allow for images, code snippets, links etc. to be included in the background information. When a new topic is created, it is added to the list of topics at the left hand side. Clicking a topic from that list takes the administrator to a page that shows the background and set of questions (if any) already entered for that topic (see Figure 4.8). The administrator can edit the topic title or background, edit or delete existing questions, and add new questions.

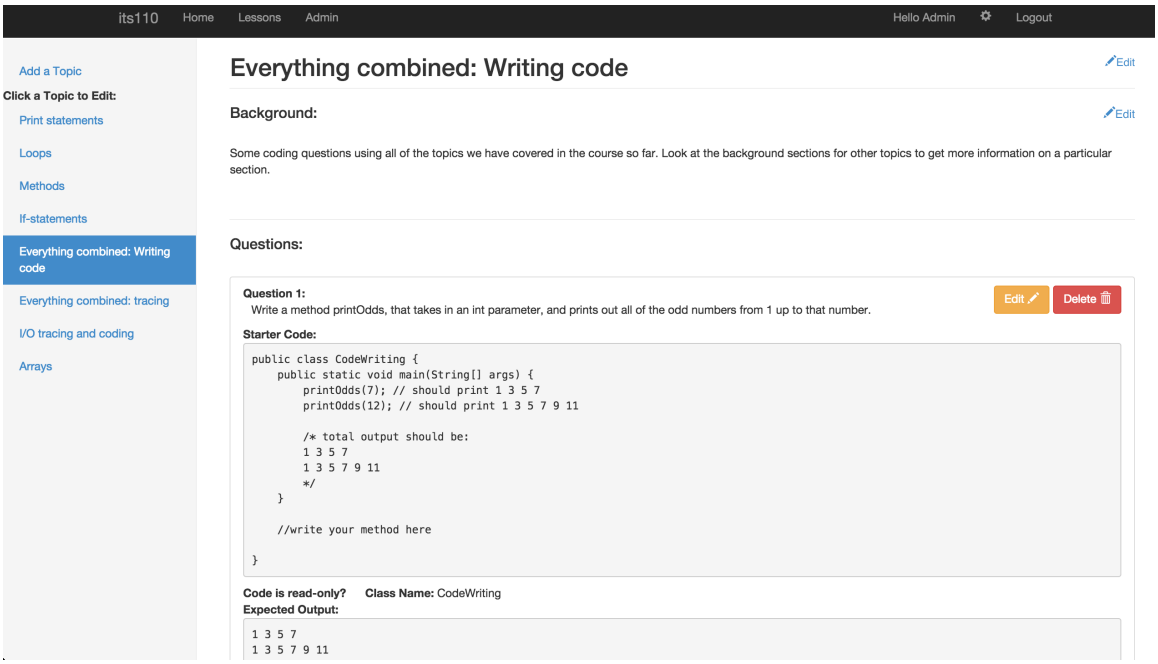

Figure 4.8: BitFit Admin — View questions for a topic

When adding new questions, the administrator is provided with an HTML editor for entering the question text, allowing for the addition of images, code snippets, links etc. The *Add Question* form (see Figures 4.9 and 4.10) has an Ace [1] code editor embedded that allows the administrator to write, compile and run any starter code for a question to ensure its correctness. Checking the *Read-only* box makes any code entered in the editor non-editable by the students when they work through that exercise, and triggers the code reading type of question for students. Leaving the *Read-only* box unchecked makes the question a code writing type, and requires the administrator to enter the output they expect the student's code to produce in the Expected Output box (shown in Figure 4.11). Finally, administrators can enter as

many hints as they wish for each question (*Add a hint to this question* button, shown in Figure 4.11). Future work will allow hints to also have HTML and code input, as currently hints can only be entered as plain text.

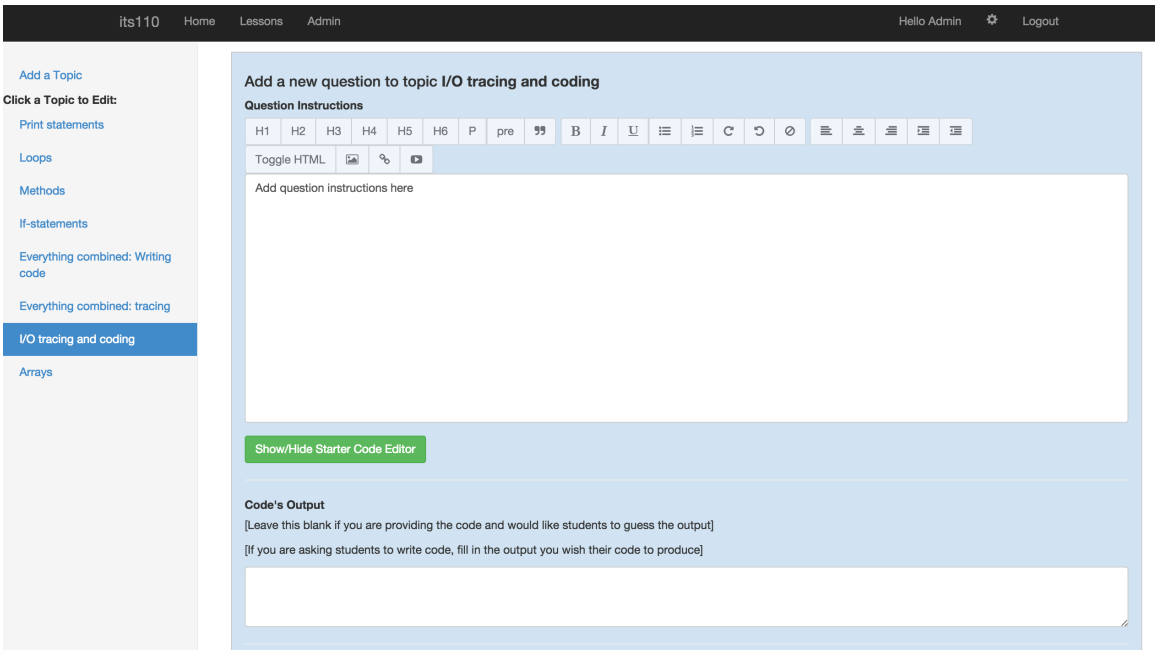

Figure 4.9: BitFit Admin — Add question form overall view

From any part of the admin-only section, the administrator can choose to click the Lessons menu item from the top navigation bar. Doing so will take the administrator to the student section of the tool, where they will be able to view the topics and exercises as students do.

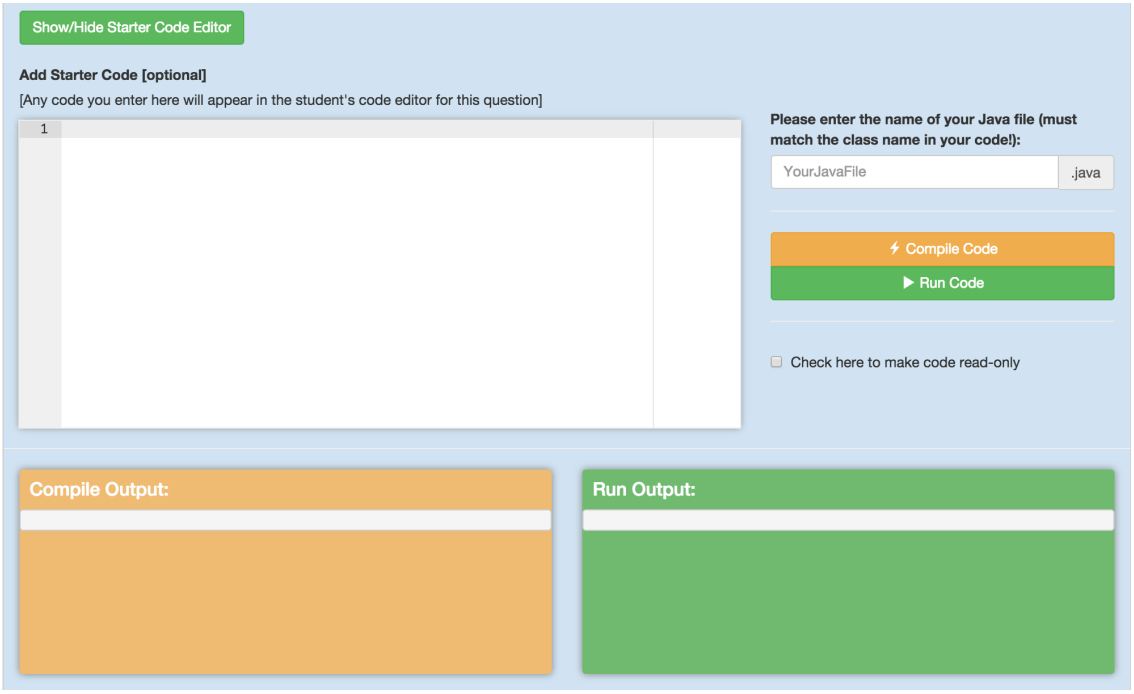

Figure 4.10: BitFit Admin — Add question form — detail view A

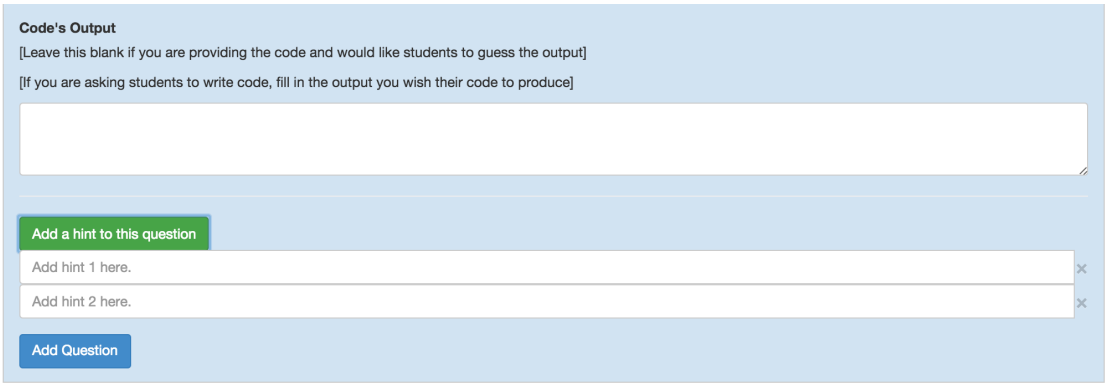

Figure 4.11: BitFit Admin — Add question form — detail view B

Chapter 5

Observational Study

This chapter describes the details of the observational study we carried out in our deployment of BitFit. In it, we present the raw results of this particular implementation of the tool. Chapter 6 gives a higher level analysis of those data, drawing on the the relevant parts of this results chapter to answer the Research Questions.

The chapter is laid out as follows: Section 5.1 introduces the observational study and the tool; Section 5.2 shows a bird's eye view of how the tool was used by looking at user behaviour; Section 5.3 also examines the data at a high level — aggregated topic by topic; Section 5.4 presents the detailed topic and question data from the tool's logs, while Sections 5.5 and 5.6 detail the usage of the Help Forum, and present the results of the student survey, respectively.

5.1 Observational Study Background

We built a web-based tool — BitFit — to provide students of CSC 110 *Fundamentals of Programming I* at the University of Victoria a space to practice the programming skills they learn during lectures and labs. The details of the observational study we carried out with BitFit are described in the following sections.

5.1.1 Course Background

CSC 110 at the University of Victoria consists of 2.5 hours of weekly lecture, taught by a UVic professor, and one hour 50 minutes per week of labs, which take place in dedicated computer labs where each student works at a computer. Labs are typically made up of groups of 20-30 students and taught by Teaching Assistants.

UVic's calendar description of CSC 110 is as follows:

Introduction to designing, implementing, and understanding computer programs using an object-oriented programming language. Topics include an introduction to computing and problem solving, selection and iteration, arrays and collections, objects and classes, top-down design and incremental development[3].

CSC 110 students are offered instruction on how to program using the Java programming language. To program in Java requires that source code be written in plain text files ending with the *.java* extension. These source files are then compiled into *.class* files by the *javac* compiler. Finally, the application is run by the Java launcher tool [2].

The winter 2015 semester (January - April 2015) of CSC 110 had two scheduled midterms: the first took place on Monday February 16, the second on Thursday March 12. Our observational study was timed to coincide with these midterms.

5.1.2 Observational Study

Our observational study ran during the five week period from February 12 to March 19 2015. We launched BitFit on February 12 with a series of practice questions on all the topics the students had covered thus far in lectures and labs, and were expected to know for the first midterm. The lead TA held a voluntary review session on Sunday February 15, during which time the TA and students worked through review questions as a group using BitFit. A second voluntary review session was held in the days leading up to Midterm 2, on Sunday March 8, where again the tool was used by the TA and students to work through review problems. Throughout the course of the observational study, students could use BitFit individually at any time on their own to work through practice problems.

Attendance at the review sessions and usage of the tool was completely optional for students. Their course grade was not directly altered in any way if they did or did not attend the sessions, or use the tool.

5.1.3 How does the tool work?

We created a user account for each student registered in the course, allowing them to log in to BitFit via their UVic email address and a password. Students could not access any of the tool's content without first logging in.

To prepare for the first midterm, students could choose to work through questions in any of the following six topics:

- Print statements
- Loops
- Methods
- If statements
- Everything combined: Writing code
- Everything combined: Tracing

Two additional topics were added to the tool for the second midterm:

- I/O tracing and coding
- Arrays

The initial set of topics and questions remained available for students to use when reviewing for the second midterm, should they wish. Each topic section contained between 5 and 9 questions.

5.1.4 Types of Questions

There are two types of questions in BitFit: code tracing and code writing.

Code tracing questions present students with the source code of a small java program. This code is not editable by students, who must read and understand what the code is doing, and then enter what they believe the output of the program will be into a box on the page. When students clicked the *Check My Answer* button, the tool compiled and ran the code in the background, then compared the actual output of the code to what the student had entered. The student was shown a message indicating whether or not their answer matched the actual output.

Code writing questions asked students to write code to create a Java program that performs a certain function. These questions provided students with a code editor, typically with some starter code to help students get started. As students worked on writing their code, they could click the *Compile* and/or *Run* buttons at any time, as many times as needed. Dialog boxes on the page showed the output of the compile attempt after pushing the Compile button (i.e., if the system was able to successfully compile their code, or, if not, the errors that the compiler encountered) and of the run attempt after pushing the Run button (i.e. the output of the program,

if everything worked correctly, or an error message if there was some error in the code arising at run time). At any time a student could click the *Check My Answer* button, at which point the system would compile and run the code in the background, and compare the student program's output with the expected output of the question. The student was shown a message indicating whether or not the output of their program matched the expected output.

Both types of questions have two additional buttons on the page: a *Hint* button and an *Ask a Question* button, either of which can be clicked at any time.

Pressing the *Hint* button results in a hint being displayed on the page to help with solving the question. Each subsequent click of the button will display another hint, until all available hints for that question have been seen, at which point a message is shown saying that no further hints are available.

Hints were only added to the first six topics; there were no hints available for any questions in the topics added for the second midterm (I/O tracing and coding and Arrays). If a student pushed the *Hint* button for any question in those topics, they received a message that no hints were available for that question. The reason behind this choice was that there are certain pedagogical tradeoffs that must be made when providing hints. When answering code writing questions, there are often a great many correct ways to answer the question. Providing code written by the instructor introduces the possibility of stifling creativity or confidence in the method the student came up with themselves, as it may differ from the instructor's hint. This study has helped us to identify some of the ideal ways to present hints that maximize the benefits to student. See the results in Chapter 5 and Chapter 7 Future Work sections for further details.

If a student clicks the *Ask a Question button*, a dialog box appears prompting for their name, email and question. Once the question is submitted, it is posted to a section of the tool only viewable by administrators, together with information about which question the student was working on at the time of asking. The administrator can then email a response to the question.

5.1.5 Data Collection

BitFit's logs collected data on how students interacted with the tool, both at a per question level, and at a per topic level.

Data Collection: Per Question

For both code writing and code reading questions, BitFit kept track of how many hints students requested per question (number of clicks on *Hint* button), the number of times they attempted the question (number of clicks on *Check My Answer*), and how many of those attempts were correct. For code writing questions, we kept track of the number of times students compiled (number of clicks on the *Compile* button) and ran (number of clicks on the *Run* button) their code.

As a student worked through each question, these data were tallied in the background of the tool. When a student moved on to the next question, the interaction data, together with a timestamp (to record the amount of time the student spent on the question), and the student's unique user ID plus the question and topic IDs were logged to the tool's server.

Data Collection: Per Topic

Based on the raw data described in the previous section, we calculated the total numbers of students who used the tool. As well, we gathered the following totals for each day of the study, for each topic (each of the following variables should be followed by "per topic, per day"):

- Number of questions attempted
- Number of students attempting questions
- Average number of questions attempted per student
- Total compiles
- Total runs
- Total hints requested
- Total attempts
- Total correct attempts

Section 5.3 and 5.4 of this chapter provides the details of these measurements, grouped by topic.

Data Collection: Threats to Validity

As with all software projects, despite initial testing before deployment, the source code for BitFit contained bugs. One such issue was a subtle error in the code for the data collection module of the tool. The bug was only discovered upon deeper

analysis of the interaction logs at the conclusion of the observational study. What was supposed to happen was that each time a student's interaction with a question was completed, the system would dispatch that question's interaction data (which included numbers of correct vs incorrect attempts, numbers of compiles, runs and hints requested, and a timestamp indicating the time the user started to work on the question), together with a timestamp to mark the end time for working on the question, to the server. The system would then reset the data collection structure to start a new timer for the next question, and zero out all the variables. However, because of the the bug (which was related to Node.js' asynchronous event loop, described in Section 3.1), what actually happened was twofold: 1.) A user's counts of numbers of runs, compiles, hints requested, total correct vs. total incorrect question attempts were never actually reset as they moved through questions. A new interaction object was created for each question and each topic, but rather than starting from zero, the counts for all these variables just kept incrementing. 2.) The starting timestamps for each previous question was overwritten by the new starting timestamp for the next question, rendering the measurements of all of the start and end timestamps meaningless.

As a consequence of these bugs, in the observational study dataset the time measurements for each question had to be thrown out. Fortunately, the nature of the bugs meant that it was possible to back-fix the first issue, correcting the bugs' ramifications by calculating the delta of the variables for each of the question interaction objects.

One final consequence that was also related to the way the tool submitted the interaction data meant that the data for the first question in each topic area were actually counted in subsequent questions. While this issue has also subsequently been fixed, it does render the data on a question by question-level not fully accurate.

All of the above issues have been fixed as of the date of writing, so future data sets collected via BitFit will not suffer from these same issues.

While the issues described in this section mean the data presented in this thesis are not 100% accurate, it does not, in fact, take away from the core argument. This thesis does not draw any direct conclusions from the actual dataset, but rather uses the dataset as a demonstration of the *kinds* of data that can be collected by tools such as BitFit, and *how* that data could be used by professors and students to improve the overall experience in introductory programming courses.

Data Collection: Google Analytics

To supplement the tool's logs, Google Analytics was used to gather additional interaction data. This data included the number of users per day, number of views per topic per day, number of user sessions per day, and average time spent on each topic. Sections 5.2 and 5.3 of this chapter provide the details of these measurements.

5.1.6 Student Survey

In the final week of the study, March 16 to March 20, 2015, students were asked to fill out a voluntary survey on their experience in using the tool (or not) to prepare for the two midterms. There were a total of 93 responses to this survey, the results of which are described in section 5.6, the Student Survey Results section.

5.2 Tool Usage by User

5.2.1 Tool Usage

Over the course of the five week study period (February 12 to March 19, 2015), a total of 728 sessions and 941 pageviews occurred (source: Google Analytics data). UVic registration showed a total of 197 students registered in CSC 110 as of March 24, 2015, though the number of students actually participating in the course was lower: the number of students with non-zero grades heading into the final exam was 192.

Number of students registered in CSC 110	Number of students who used BitFit (See table 5.2 for details)	Number of student responses to tool usage survey
197	115	93

Table 5.1: Student Engagement Numbers

Our tool logs recorded a total of 109 unique users. This number is slightly lower than the actual number of students using the tool however, due to confusion on the part of some students as to how to login in to their user accounts on the tool's website. A user account was created for each student using their UVic email and a default password, and students were encouraged to change their passwords on first usage of the tool. An email was sent to all students introducing the tool, providing the link to the tool's website, and describing how to login. In the email, an example of how to login was given, using the lead TA's user account (which was not an admin account, just a student account). This was an error on our part, as many students simply copied and pasted this login info into the tool, rather than using their own email to login. We did not discover this was happening until the end of the study period, at which point we changed the password to stop this from continuing. The implications arising as a result of this issue are twofold: first, we do not know the exact number of students who used the tool, though we can make an educated guess (See Table 5.2 — Number of Students Using Tool). Secondly, our tool's logs were affected due to a bug, described in detail in section 5.1.5, that was exacerbated by the issue of multiple students using one account at the same time.

Unique user count (not including TA account)	Total number of user interactions	Total number of interactions on TA account	Average number of interactions per student (with / without TA account)	Likely number of users using TA account	Revised number of unique users
104	6971	646	66.39 / 60.82	11	115

Table 5.2: Number of Students Using Tool

5.2.2 Users by Day

Figure 5.1 shows user sessions of the tool, broken down by day. It shows the way the tool was used by students and the teaching team. It is a mostly bimodal graph, with peak usage occurring during the days leading up to Midterm 1 and Midterm 2.

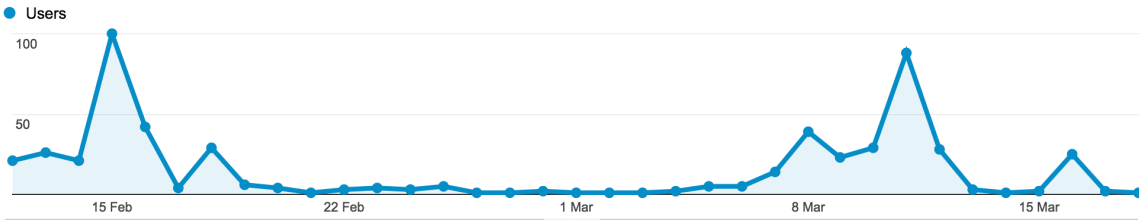

Figure 5.1: BitFit Usage by Day

A voluntary review session for Midterm 1 was held on Sunday February 15, during which the tool was used by both the lead TA and students attending the session, to work through review problems. The first midterm took place on Monday February 16. Another voluntary review session was held for Midterm 2 on Sunday, March 8, where again the tool was used by the TA and students to work through review problems. Midterm 2 took place on Thursday March 12.

5.2.3 Pageviews, Sessions and Users by Day

Figure 5.2 and Table 5.3 show the number of page views (which also can be thought of as topic views), individual sessions (i.e., a single user may have had multiple sessions on a given day), and users on a daily basis. Peak times for all three variables occur on February 15 and March 11, with a peak in pageviews occurring on February 13.

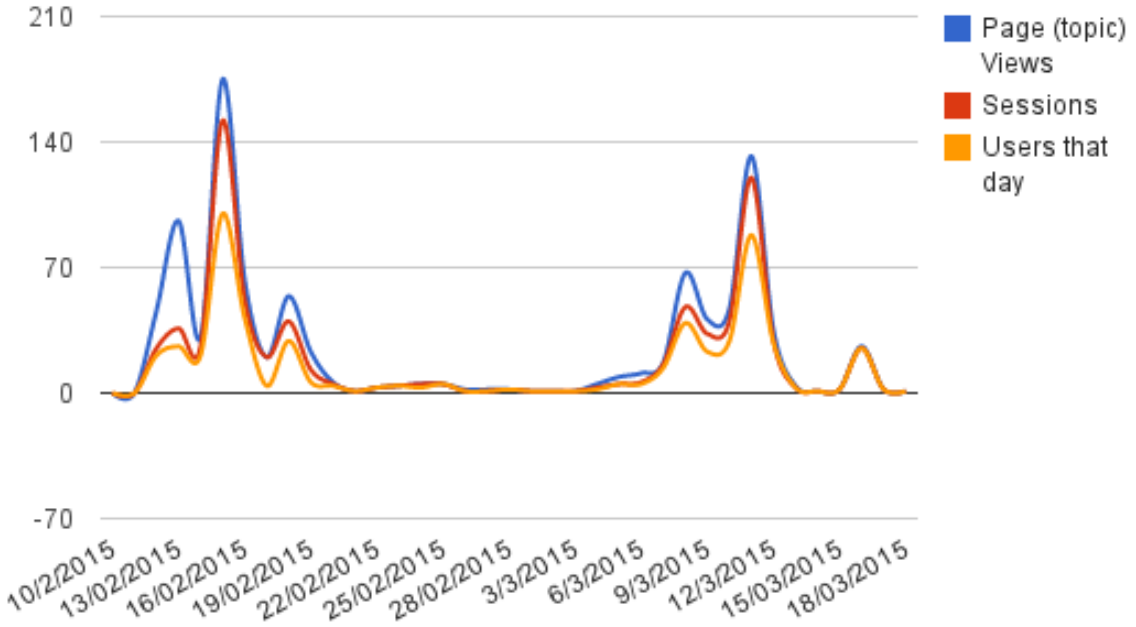

Figure 5.2: BitFit Pageviews, Sessions and Users by Day

Day	Page Views by day	Sessions by day	Users by day
12/2/2015	46	25	21
13/02/2015	96	36	26
14/02/2015	31	26	21
15/02/2015	175	152	100
16/02/2015	63	52	42
17/02/2015	20	20	4
18/02/2015	54	40	29
19/02/2015	23	13	6
20/02/2015	6	5	4
21/02/2015	1	1	1
22/02/2015	3	3	3
23/02/2015	4	4	4
24/02/2015	5	5	3
25/02/2015	5	5	5
26/02/2015	2	1	1
27/02/2015	2	1	1
28/02/2015	2	2	2
1/3/2015	1	1	1
2/3/2015	1	1	1
3/3/2015	1	1	1
4/3/2015	5	3	2
5/3/2015	9	5	5
6/3/2015	11	6	5
7/3/2015	17	16	14
8/3/2015	67	48	39
9/3/2015	41	33	23
10/3/2015	47	40	29
11/3/2015	132	120	88
12/3/2015	35	29	28
13/03/2015	4	3	3
14/03/2015	1	1	1
15/03/2015	2	2	2
16/03/2015	26	25	25
17/03/2015	2	2	2
18/03/2015	1	1	1
Totals	941	728	543

Table 5.3: Daily breakdown of pageviews, sessions and users, 02/12/2015 - 03/18/2015

5.2.4 Sample of Individual Usage Data

Tables 5.4 and 5.5 show a very small sampling of two students' usage of BitFit during a single session. Each student was working on questions from the Loops topic.

Student A				
Num. Compiles	Num. Runs	Num. Hints	Total Attempts	Correct Attempts
6	8	0	2	1
9	11	0	4	2
10	12	0	4	2
1	1	1	1	1
3	3	0	2	2
1	1	0	1	1
9	9	0	2	2

Table 5.4: Sampling of Individual User Data: Student A

Student B				
Num. Compiles	Num. Runs	Num. Hints	Total Attempts	Correct Attempts
7	2	3	4	1
3	0	5	3	1
10	3	4	5	2
12	1	3	6	0
4	2	0	7	2
3	0	3	2	0

Table 5.5: Sampling of Individual User Data: Student B

5.3 Tool Usage by Topic Summary

This section examines the dataset at a zoomed out, topic by topic level. The following sections investigate the average amounts of time users spent on each topic, and the numbers of times they visited each of the topic subsections. Next, data about the numbers of questions users attempted, plus the numbers of compiles, runs and hints requested on a per-topic basis are presented. Finally, we see what percentages of attempts to answer questions were correct, again at a topic-by-topic level.

5.3.1 Average Time Spent Per Topic

What topics did users work on, and how long did they work on those topics? Figure 5.3 and Table 5.6 show the average amount of time that users spent on each topic, calculated over the study period February 2 - March 18 2015. The topic If Statements had the greatest average time spent at 17 minutes 52 seconds. Users spent close to 10 minutes on average on Code Tracing and Arrays, while Loops and I/O Tracing and Coding had the lowest times at approximately 3 minutes each. It should be noted that the topics I/O Tracing and Coding and Arrays were only available in the second half of the observational study period, therefore users had less opportunity to spend time on those topics, as compared to the others.

5.3.2 Total Pageviews and Average Time per Topic

Figure 5.4 and Table 5.6 show the total pageviews versus the average time in minutes spent per topic. The highest numbers of pageviews were for the Everything Combined: Writing Code (49) and Loops (48) topics, while the lowest number of pageviews was for Print statements. In contrast, the greatest average amount of time spent on a topic was for If Statements (17:52 minutes), which also had one of the lowest total pageview counts (at 10 pageviews).

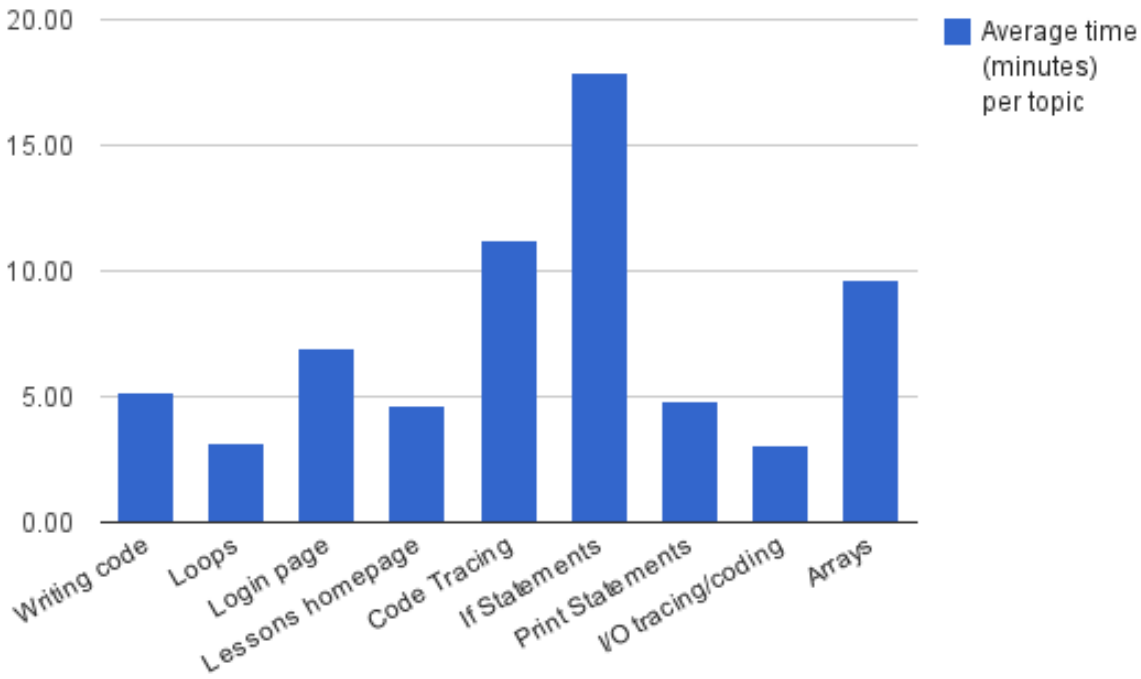

Figure 5.3: BitFit Average Time per Topic

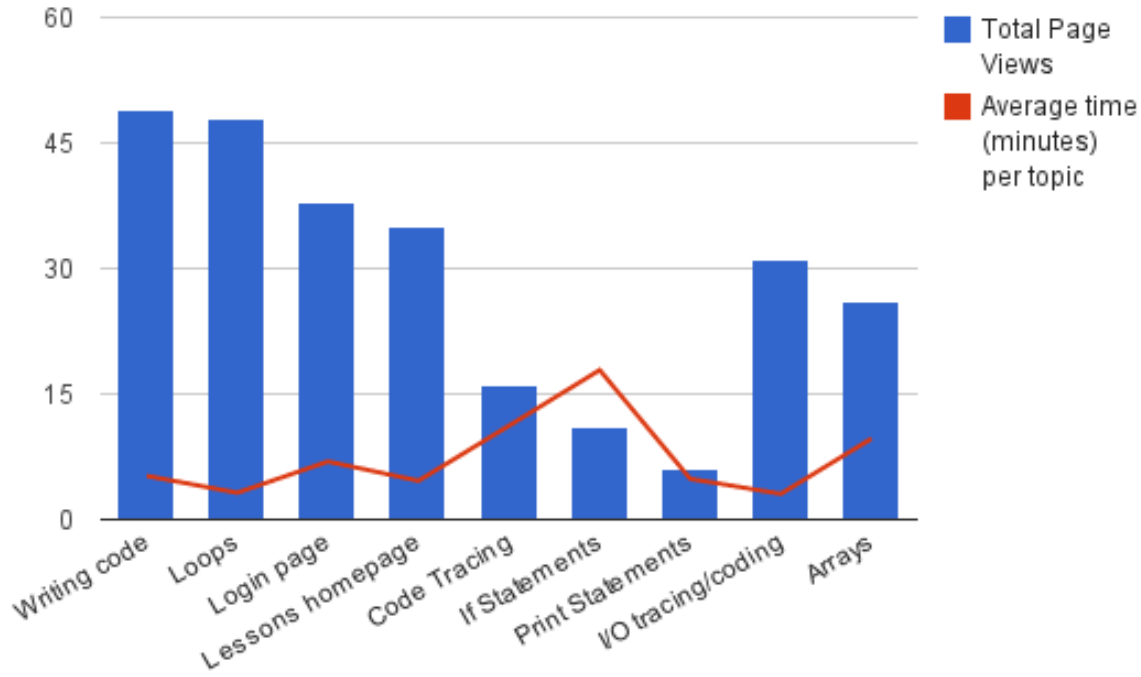

Figure 5.4: BitFit Average Time per Topic vs. Total Topic Views

Page Title	Total Page Views	Average time on page
Home	632	
Everything combined: Writing code	49	0:05:13
Loops	48	0:03:12
login	38	0:06:56
lessons home	35	0:04:38
I/O tracing and coding	31	0:03:04
Arrays	26	0:09:40
Everything combined: tracing	16	0:11:13
If Statements	11	0:17:52
Totals	941	0:07:21

Table 5.6: Total Page Views and Average Time per Topic, 02/12/2015 - 03/18/2015

5.3.3 Total Questions Attempted, Total Users, Average Questions per User

The Loops topic had the most question attempts over the period of study with 119 students attempting questions a total of 636 times, or approximately 52.3% of Loops questions attempted per user per day. The "Writing Code (Everything Combined)" topic had the greatest number of users attempting questions at 125 users making 453 question attempts, for a total of 54.3% of questions attempted per user, per day. The topics Methods and "Tracing (Everything Combined)" had the highest percentages of questions attempted per user per day, at 65.3% and 64.3% percents respectively.

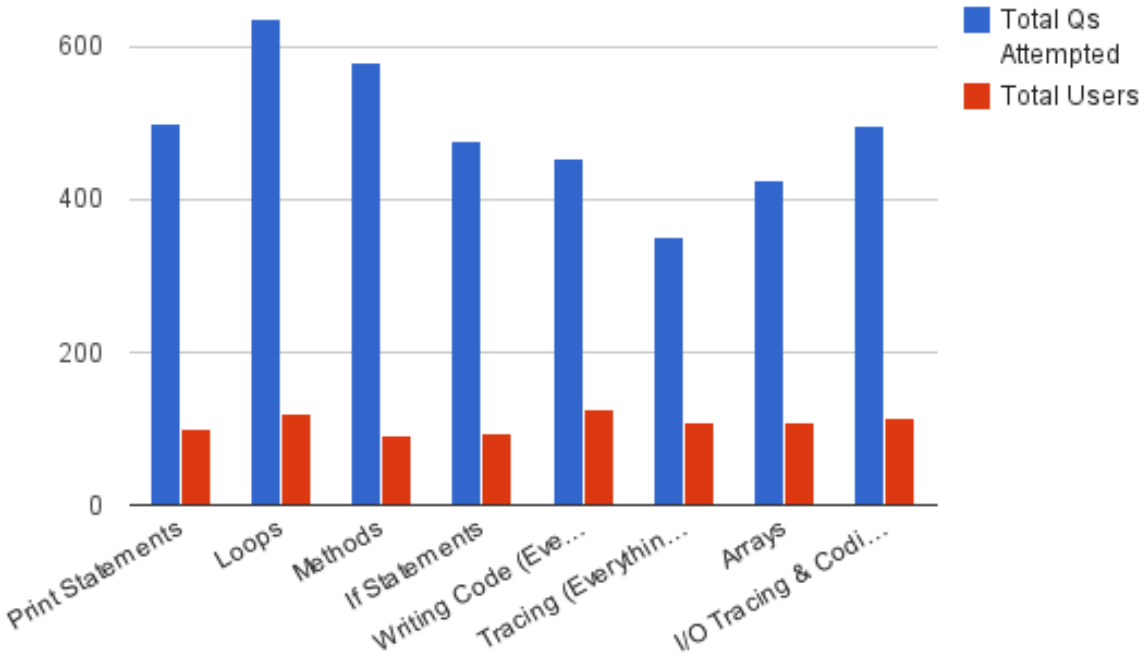

Figure 5.5: BitFit Total Questions Attempted and Total Users

5.3.4 Total Compiles, Total Runs and Total Hints Requested

Figure 5.7 and Table 5.8 show the total numbers of compiles, runs and hints requested per topic over the course of the study period (February 12 to March 19, 2015).

Due to the bug in the data collection/logging code (described in Section 5.1.5), there are errors in the topics Print Statements, Methods, If Statements and Tracing (Everything Combined): since these four topics contained only read-only questions, meaning the students could not edit the code in the question, and therefore could

Topic	Total Qs Attempted	Total Users	Average Qs Attempted per User per Day	Total Qs in Tool	Percent of Questions Attempted per User per Day
Print Statements	500	100	4.19	7	59.87%
Loops	636	119	4.18	8	52.30%
Methods	578	92	5.88	9	65.33%
If Statements	475	94	4.40	7	62.89%
Writing Code (Everything Combined)	453	125	3.26	6	54.25%
Tracing (Everything Combined)	352	107	3.21	5	64.25%
Arrays	424	108	3.24	7	46.29%
I/O Tracing & Coding: Totals	495	114	3.83	7	54.69%

Table 5.7: Total Questions Attempted and Total Users

not compile or run the code, the counts for Total Compiles and Total Runs in these topics should all be zero. Therefore the only valid numbers of Compiles and Runs in the following graph and tables are for the topics Loops, Writing Code (Everything Combined), Arrays and I/O Tracing & Coding.

The Loops topic had the greatest number of compiles and runs, at 3717 and 2584, respectively, while the Writing Code topic's numbers were somewhat lower at 2640 compiles and 1709 runs. Compiles and runs for the I/O Tracing & Coding topic were 1118 and 504, respectively, while for the Array topic the respective numbers were the lowest at 938 and 437.

The number of hints requested ranged from 111 for Arrays to 482 for loops. What is interesting about the numbers for hints requested is that for the two topics added for midterm 2 — Arrays and I/O Tracing & Coding — we did not add any hints at all, though the *Hint* button was still present on the UI. The button recorded a hint request every time somebody clicked on it, despite the fact that students received no

hint, but saw a message indicating there were no hints for the question they were on.

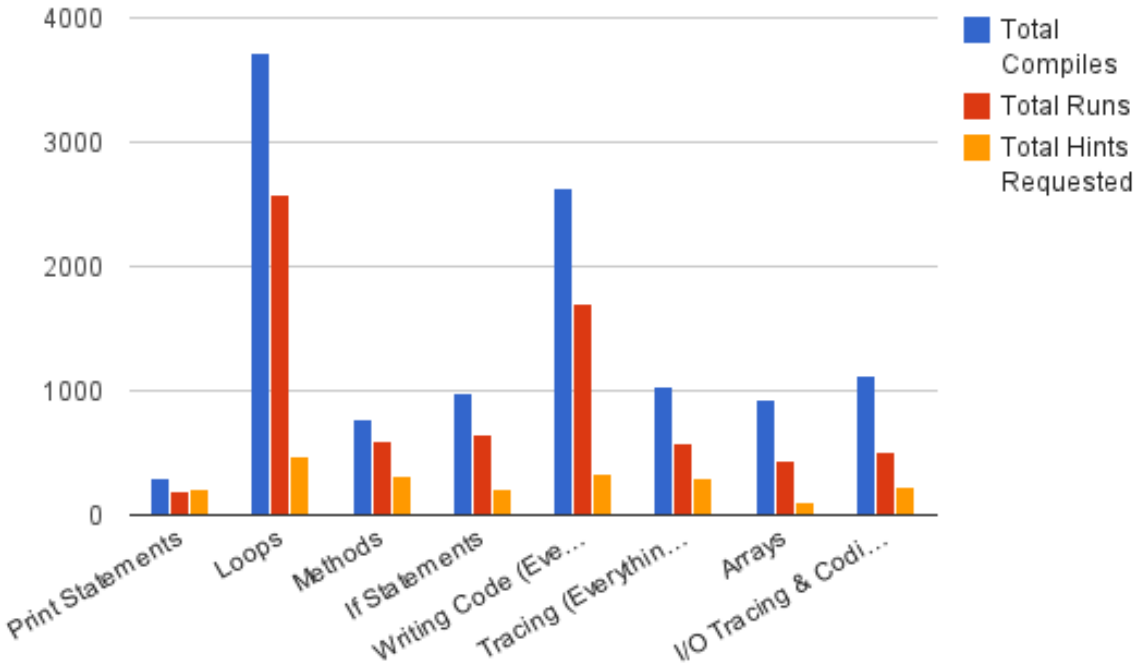

Figure 5.6: BitFit Total Compiles, Total Runs and Total Hints

Topic	Total Compiles	Total Runs	Total Hints Requested
Print Statements	297	187	209
Loops	3717	2584	482
Methods	776	599	310
If Statements	981	647	204
Writing Code (Everything Combined)	2640	1709	331
Tracing (Everything Combined)	1043	572	294
Arrays	938	437	111
I/O Tracing & Coding: Totals	1118	504	230

Table 5.8: Total Compiles, Total Runs and Total Hints Requested

5.3.5 Total Attempts vs. Total Correct Attempts

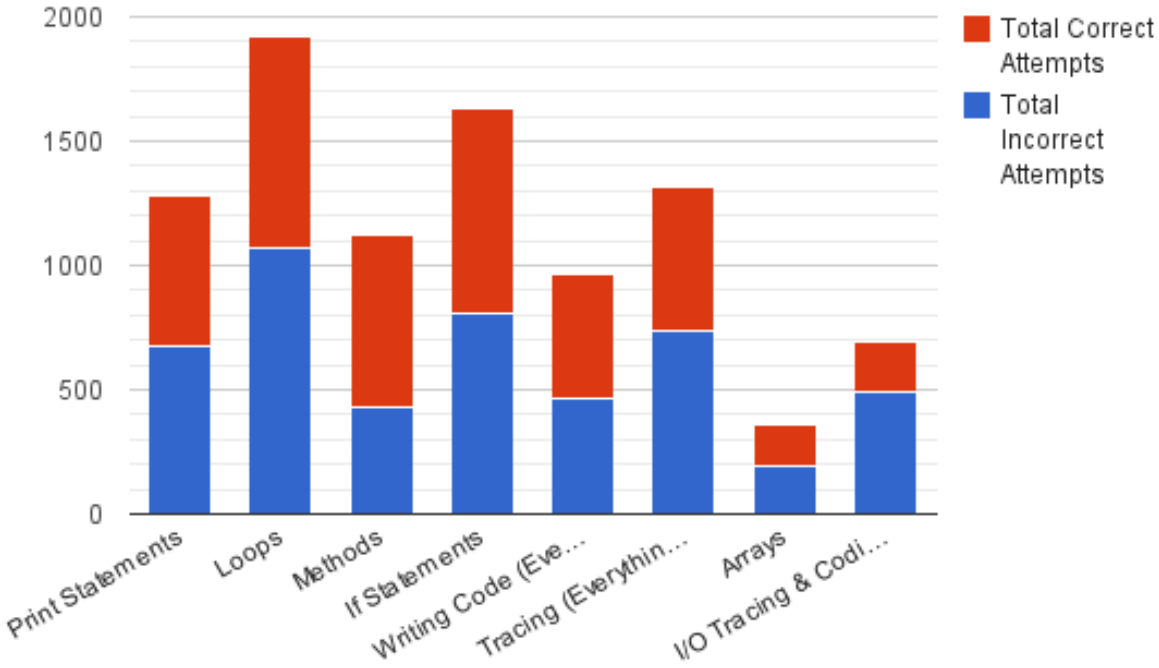

Figure 5.7: BitFit Total Incorrect Attempts vs. Total Correct Attempts

Figure 5.7 and Table 5.9 show the total number of times students attempted a question in a topic area, and the total number of correct or incorrect attempts per topic. The highest percentage of correct attempts was in the Methods topic, at 61.5%, while the highest percentage of incorrect attempts was in the I/O Tracing & Coding topic, with 71.45% incorrect question attempts. The greatest number of overall attempts was in the loops topic, with 1074 attempts.

Topic	Total Incorrect Attempts	Total Correct Attempts	Percent Correct	Total Attempts
Print Statements	679	601	46.95%	1280
Loops	1074	845	44.03%	1919
Methods	432	690	61.50%	1122
If Statements	807	823	50.49%	1630
Writing Code (Everything Combined)	466	497	51.61%	963
Tracing (Everything Combined)	735	579	44.06%	1314
Arrays	195	163	45.53%	358
I/O Tracing & Coding: Totals	493	197	28.55%	690

Table 5.9: BitFit Total Incorrect Attempts vs. Total Correct Attempts

5.4 Tool Usage by Topic and Questions

This section describes the data collected for each of the eight topics. The full set of data, both graphs and tables, is included in the Appendix, Sections A.1 and A.2. There, data for each of the collected variables is presented on a daily basis, highlighting peaks in student usage of the tool. The variables in each topic are as follows:

- Number of questions attempted
- Number of students attempting questions
- Average number of questions attempted per student
- Total compiles
- Total runs
- Total hints requested
- Total attempts
- Total correct attempts

Only the dates where non-zero data exists for each topic have been included in the topic graphs and tables.

Within each topic, the data are then broken down further to detail tool interaction on a per question basis. The following variables are measured for each question in a topic:

- Total compiles
- Total runs
- Total hints requested
- Total attempts
- Total correct attempts

5.5 Help Forum Usage

Each question on the tool is presented to the user with both a *Hint* button and an *Ask a Question* button. The students were told about the *Ask a Question* button when the tool was first introduced to them, and we explained that any questions they ask are between the TA's and the students only. By using this button, a total of 12 questions were asked by 9 different users over the course of the study period. We answered these questions by replying via email to the address the student provided, in as short a time as possible (usually no more than a 1-2 hour delay, if that). Subsequently,

a number of these students continued the discussion via email after the initial email contact was made.

Questions on the help forum typically centred around usage of the tool, rather than questions specific to the material.

5.6 Student Survey Results

During the week of March 16 to March 20, 2015, students were asked to fill out a voluntary survey on their experience in using the tool (or not) to prepare for the two midterms. There were a total of 93 responses to this survey, the results of which are discussed below.

The following sections contain the questions and student responses.

5.6.1 Did you use the programming practice tool to help prepare for either of the midterms?

Eighty-nine students responded to this question, with the following breakdown of answers:

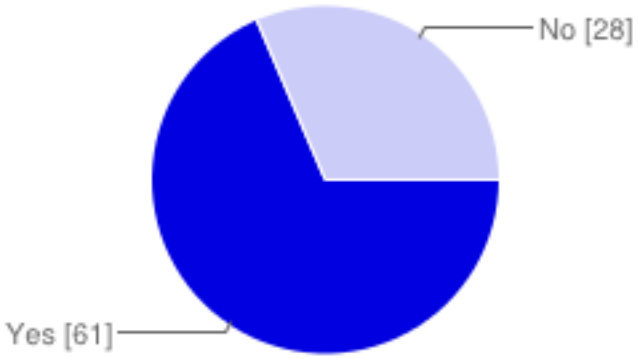

Figure 5.8: Students Who Used or Did Not Use BitFit

Yes: 61 (65.6%) No: 28 (30.1%)

5.6.2 Questions on Why Tool Not Used

Thirty-two students responded to the question "Why did you not use the programming practice tool?".

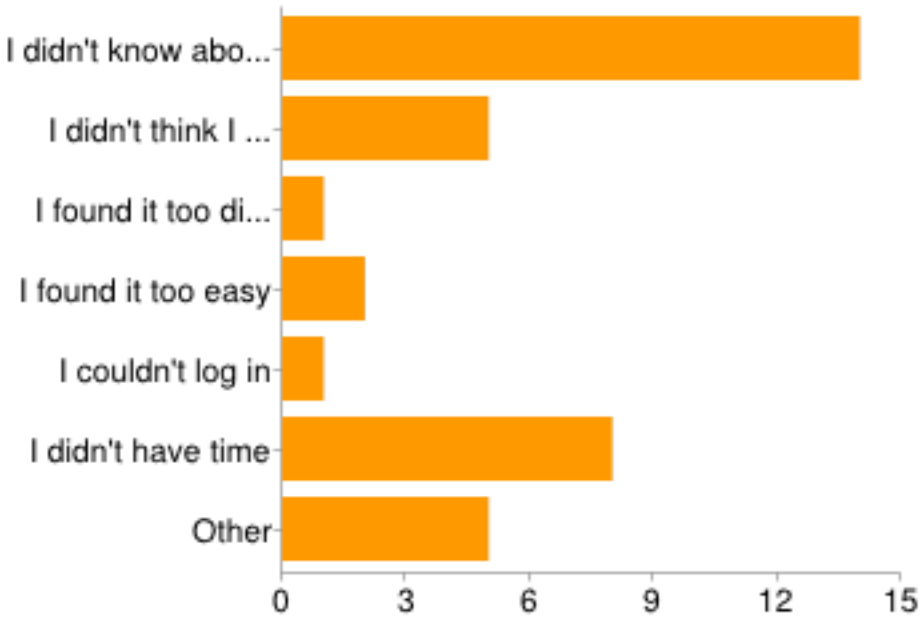

Figure 5.9: Why did you not use the programming practice tool? (see Table 5.10 for possible answers)

There were 18 responses to the question "Would you have used the tool or one similar, under different circumstances? Why or why not? Please give as much detail as you are able." Only 2 of these respondents indicated they would not use the tool or one similar in future, while the rest of the responses were either positive (the majority) or neutral in their willingness to try out the tool or one similar.

There were 25 responses to the question "Which (if any) of the following question topics did you struggle with?". These non-users of the tool indicated they struggled most with Arrays (16.1%) and I/O Tracing and coding (10.8%). Only one respondent (1.1%) indicated struggling with Print Statements.

5.6.3 Questions on usage of Programming Practice Tool

Fifty-six students responded to the question "Do you think the programming practice tool helped your grade on Midterm 1? Why or why not?". Forty-nine of the responses indicated that the tool helped them in some way. All of the six negative responses arose from the student indicating they did not use the tool to help prepare for midterm 1.

Fifty-three students responded to the question "Do you think the programming

I didn't know about it	14	15.10%
I didn't think I needed extra help before the midterms	5	5.40%
I found it too difficult/too advanced	1	1.10%
I found it too easy	2	2.20%
I couldn't log in	1	1.10%
I didn't have time	8	8.60%
Other	5	5.40%

Table 5.10: Reasons Why Students Did Not Use BitFit

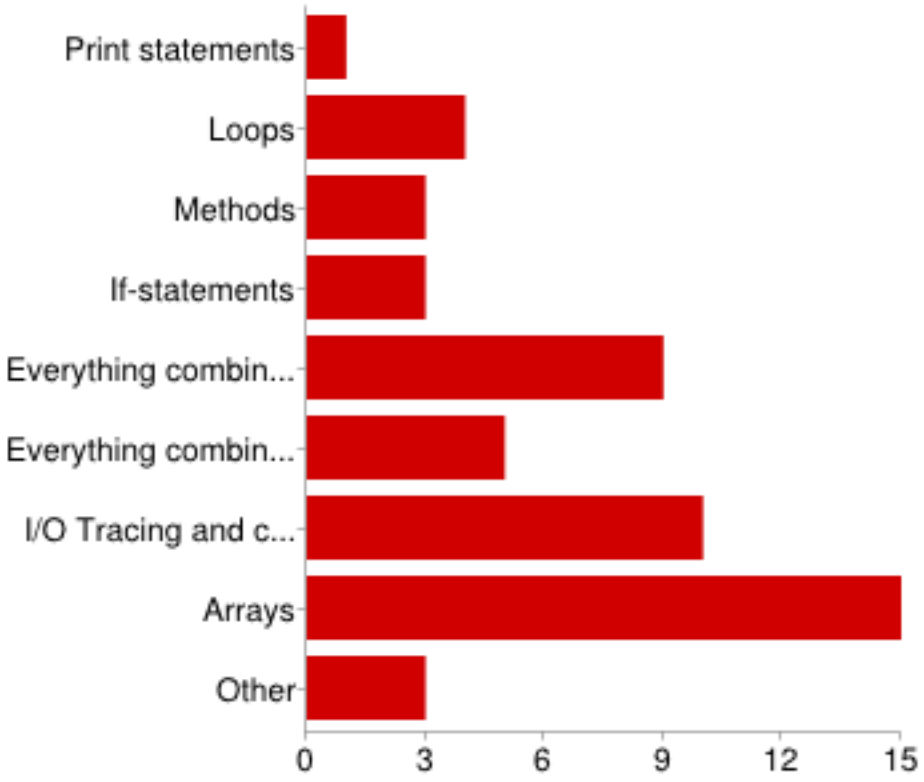

Figure 5.10: Which (if any) of the following question topics did you struggle with?

practice tool helped your grade on Midterm 2? Why or why not?". Forty-three indicated that they felt the tool helped them in some way prepare for Midterm 2, while 10 indicated that the tool did not help them.

There were 54 student responses to the question "Did the programming practice tool have any affect on how you felt before writing either midterm? Why or why not?". Of these short answer responses, seven indicated the tool had no effect while

Print statements	1	1.10%
Loops	4	4.30%
Methods	3	3.20%
If-statements	3	3.20%
Everything combined: Writing code	9	9.70%
Everything combined: Tracing	5	5.40%
I/O Tracing and coding	10	10.80%
Arrays	15	16.10%
Other	3	3.20%

Table 5.11: Topics Non-Users of BitFit Struggled With

45 indicated that the tool had some form of positive effect. Thirty-three students said that using the tool improved their confidence going into one or both of the midterms, while six students indicated the tool had some form of negative effect. Of these six responses indicating a negative effect such as worry or anxiety, all of the students also indicated the tool having some form of positive effect in addition to the negative one.

Fifty-one students responded to the question "Did the programming practice tool help you get an idea of where you stand with respect to the course content? Why or why not?". Forty-six responses indicated yes, seven indicated no, and one indicated I don't know. These responses do not add up to the total as there were some students who said the tool both was and wasn't helpful.

Fifty-two students responded to the question "Which (if any) of the following question topics did you struggle with?". Nearly 35% of students who used the tool indicated they struggled with the I/O Tracing and coding topic, while 30.1% struggled with arrays. The topics Methods, If-Statements and Print statements had the least number of people who struggled with them, at 1.1%, 1.1% and 2.2%, respectively.

Forty students responded to the question "Did the programming practice tool help with your understanding of the topics you struggled with?". Twenty-six students indicated the tool did help them, seven indicated that it did not, eight indicated that the tool helped "only a little bit" or "somewhat", while two indicated not having any struggles with the material. Of the seven who indicated the tool did not help with their struggles, most provided insights into why this was so, which usually was to do with the fact that the tool did not provide the actual answer to a question if they were really stuck on it (see Chapter 7 Future Work for discussion of this).

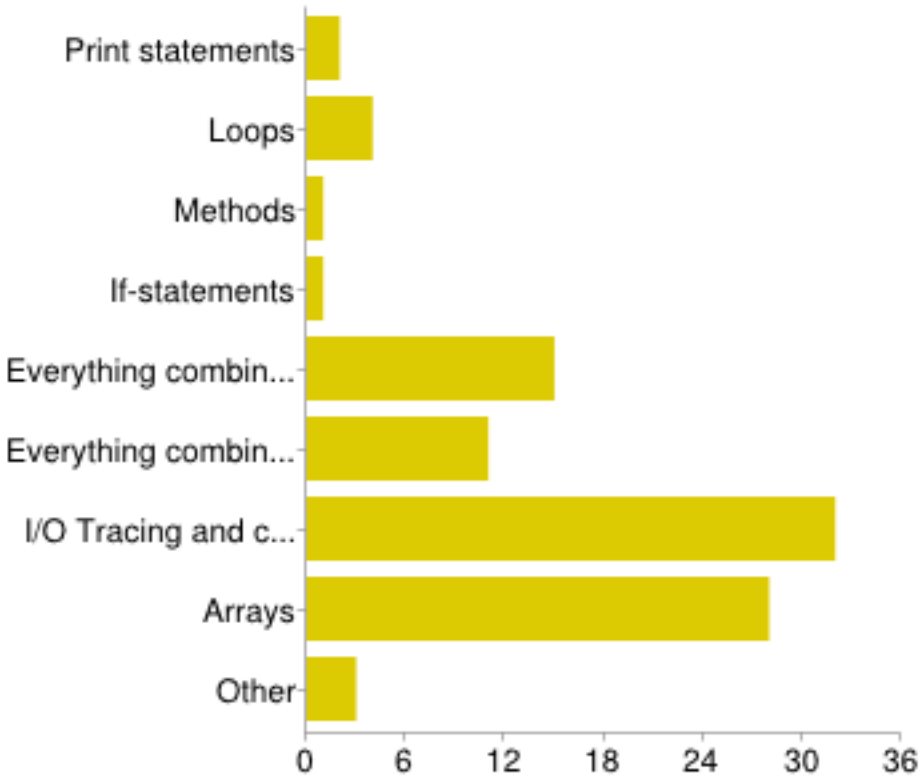

Figure 5.11: Which (if any) of the following question topics did you struggle with?

5.6.4 Questions for Users and Non-Users of the Tool

There were 92 responses to the question "If there was a 'Practice Final Exam' section added to the tool, how likely would you be to work through the questions prior to the final exam?", broken down as follows.

Eighty-eight students responded to the question "During this course, when you struggled with some concept or assignment, where did you go for help?".

Print statements	2	2.20%
Loops	4	4.30%
Methods	1	1.10%
If-statements	1	1.10%
Everything combined: Writing code	15	16.10%
Everything combined: Tracing	11	11.80%
I/O Tracing and coding	32	34.40%
Arrays	28	30.10%
Other	3	3.20%

Table 5.12: Topics Users of BitFit Struggled With

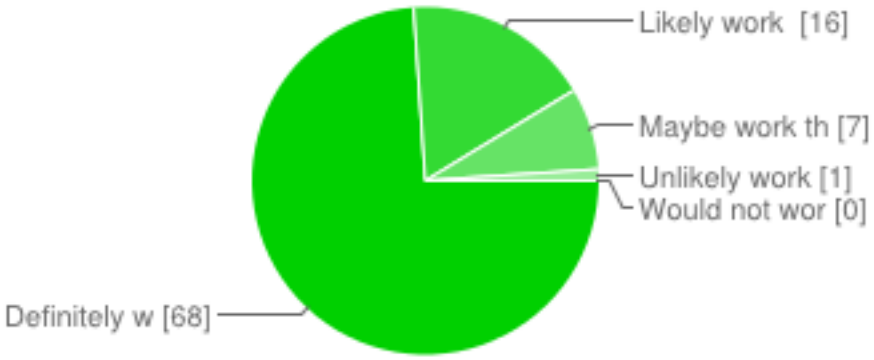

Figure 5.12: If there was a 'Practice Final Exam' section added to the tool, how likely would you be to work through the questions prior to the final exam?

Definitely work through questions	68	73.10%
Likely work through questions	16	17.20%
Maybe work through questions	7	7.50%
Unlikely work through questions	1	1.10%
Would not work through questions	0	0%

Table 5.13: Likelihood of doing practice final exam questions

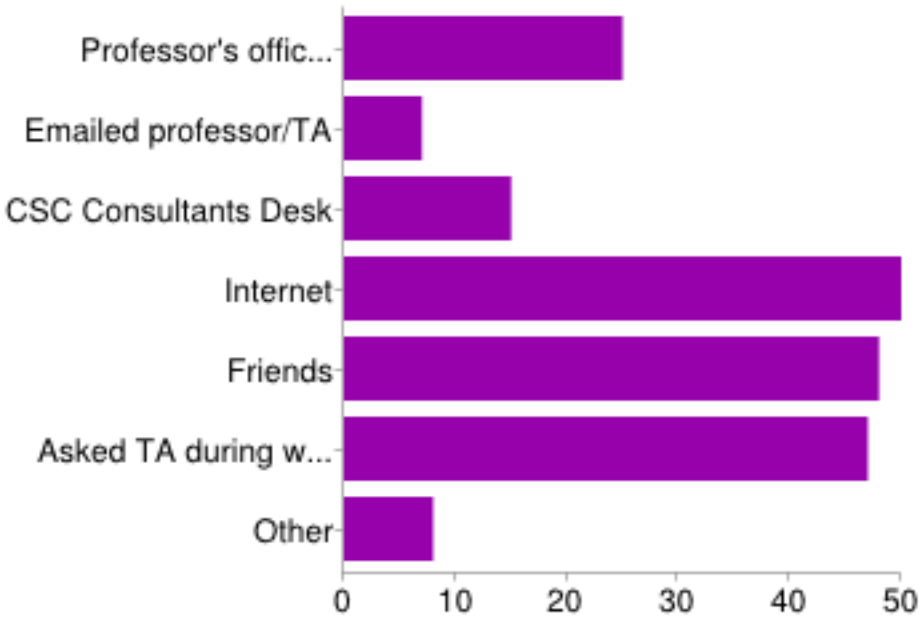

Figure 5.13: During this course, when you struggled with some concept or assignment, where did you go for help?

Professor's office hours	25	26.90%
Emailed professor/TA	7	7.50%
CSC Consultants Desk	15	16.10%
Internet	50	53.80%
Friends	48	51.60%
Asked TA during weekly lab	47	50.50%
Other	8	8.60%

Table 5.14: Where students went for help during course struggles

Ninety students responded to a question asking about their likelihood of asking questions during LECTURES.

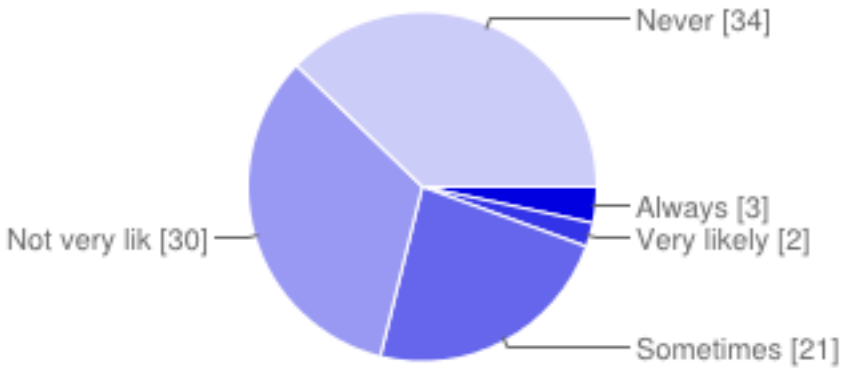

Figure 5.14: Asking questions in LECTURE

Always	3	3.20%
Very likely	2	2.20%
Sometimes	21	22.60%
Not very likely	30	32.30%
Never	34	36.60%

Table 5.15: Likelihood of asking questions in LECTURE

Ninety students responded to a question asking about their likelihood of asking questions during LABS.

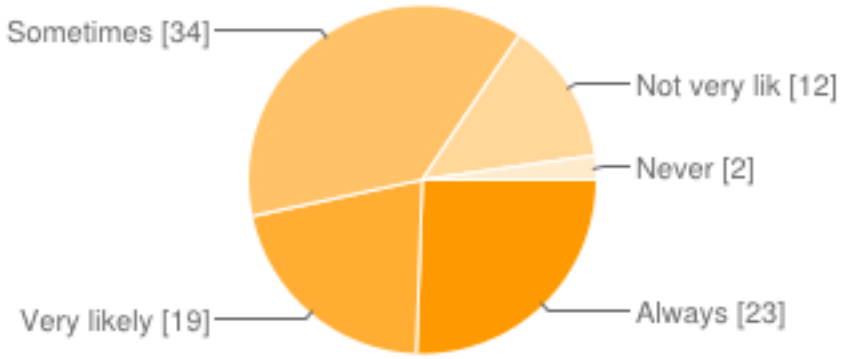

Figure 5.15: Asking questions in LABS

Always	23	24.7%
Very likely	19	20.4%
Sometimes	34	36.60%
Not very likely	12	12.9%
Never	2	2.2%

Table 5.16: Likelihood of asking questions in LABS

Chapter 6

Analysis: Quantitative and Qualitative Results

In our observational study, we used BitFit to gather raw data on seven variables at an individual user level. As each user worked through the questions in the tool, we collected the following statistics for each question, in each topic area:

1. User ID
2. Total question attempts
3. Total correct question attempts
4. Number of hints requested
5. Number of compiles (code writing questions only)
6. Number of runs (code writing questions only)
7. Time spent per question

The subsections in this chapter each address one of the three research questions, describing the methodology used to answer the question, the pertinent results from Chapter 5, the results chapter, and how these results answer the particular research question.

6.1 Collecting Data to Help Instructors

Instructors want to know whether students are understanding the material being taught. Is the pace of the course too fast? Too slow? With which topics do students have more difficulty than others? With which types of exam questions do students struggle most?

Answering these questions can be tough. Questions asked by students, plus marks from assignments, lab exercises, midterms and final exams are typically used to guess at the answers. Often, professors don't get a real sense of how students are doing until after the first midterm — often about a third of the way through the course. According to our student survey, when students have questions during lecture, only 5.5% of respondents (5/90) are likely to ask those questions of the professor. It is encouraging that a greater percentage of students reported seeking help during the professor's office hours (26.9%), or via email (7.5%); however, this still leaves nearly three quarters of the class seeking help elsewhere (indeed, 50% and 48% reported consulting the Internet and friends, respectively, while 47% and 15% reported asking the TA during weekly lab, or visiting the CSC consultants' desk, respectively). How, then, can a professor begin to see a clearer picture of how students are faring in their courses at an earlier point in time?

A tool like BitFit can be used to fill in gaps in a professor's knowledge of overall student struggles or comprehension, right from the first week of class. The following sections address research questions 1 and 2, first presenting the methodology used to answer the questions, then highlighting the relevant data from the previous chapter, and finally identifying how these results answer the research questions.

6.1.1 Research Question 1: What data could be gathered by a tool like BitFit to help instructors offer a better learning experience to students relative to existing offerings?

Methodology to Answer Research Question

We approached this question first by identifying the topics and questions with which students struggle, which they have mastered, and which questions, if any, are too easy or too hard. To do this we aggregated the data from individual user interactions with BitFit, and grouped the results by question and by topic. We then analyzed, for each topic, the average time spent by students on that topic, and the average number of questions students tried to answer per topic. Next, we aggregated the following variables, both for each question in a topic, as well as for each topic:

- Total number of compiles
- Total number of runs

- Total number of hints requested
- Total number of question attempts (*Check my answer* button clicked)
- Total number of correct attempts

Relevant Results

Section 5.3 in the results chapter shows summaries of BitFit usage by topic. Figure 5.3 calculates the average time users spent on each topic, and shows that over the course of the 5 week study period, users spent, for example, an average of 17 minutes on If Statements, and 10 minutes each on Code Tracing and Arrays. Currently, this measurement has many potential flaws, as it merely shows the average amount of time users spent on each of the topic pages, as calculated by Google Analytics. Future work on BitFit will correct the time calculation bug (see Chapter 7 for details) and allow for a much finer grained calculation of how long students spend on a question and a topic.

Timing measurements on their own, however, are not enough, as they are susceptible to many issues that could corrupt calculations. For instance, if a student begins work on a question (so the timer starts running), but then moves on to something else for a period of time before returning to finish the question, the timer will be unnecessarily high. This issue could be mitigated perhaps by capping time measurements, or by marking periods of inactivity on the tool logs. To bolster the usefulness of timing measurements, we must also consider other variables.

To more thoroughly answer the question of which topics students struggle with or have mastered in a course, we can consider the timing measurements alongside the total number of compiles, runs, hints requested and incorrect total question attempts by topic. Drawing on the data from the previous chapter, we consider the measurements for the Loops and the Methods topics to demonstrate how the tool's data could be used to do so.

The Loops topic saw 119 users attempting to answer 636 Loops questions. With 8 loops questions available, this means the average user attempted about 52% of the 8 questions per day of activity in the study, or roughly 4 questions. Of the topics with code writing questions, Loops had the highest number of compiles (3717) and runs (2584), and it had the greatest number of hints requested (482) across all topics. In total, users checked their answers to Loops questions 1919 times, of which 845 attempts (44%) were correct.

For the Loops topic, in the days leading up to Midterm 1, students requested the most hints for question 2 (88). That question also had the lowest number of correct attempts at 36.4%. Students did not fare much better on questions 7 and 8 with only about 40% correct attempts for both, and with 67 and 86 hints requested (respectively). In contrast, the highest percentage of correct answers was for question 3 (53.5%), which also had the fewest hints requested (39).

The Methods topic saw 92 users attempting to answer 578 Methods questions. With 9 Methods questions available, this works out to each user attempting about 65% of the 9 questions per day, or roughly 6 questions. Students requested a total of 310 hints on Methods questions, and checked their answers to questions 1122 times, of which 690 attempts (61.5%) were correct.

For the Methods topic, in the days leading up to Midterm 1, students requested the most hints for question 4 (109). Questions 4 and 8 had the lowest numbers of correct attempts at 51% and 50.6% respectively. The highest percent of correct attempts was for question 6, at 77.4%, which also had the lowest number of hints requested — only 7.

How Do Results Answer the Research Question?

These results must, of course, be interpreted within the context of the situation in which they were measured and cannot be said to answer the research question with absolute certainty. They do, however, suggest some compelling points.

High numbers of compiles and runs alone are not enough to infer that students struggled on a topic: it could be that a particular question was more difficult than others, while students breezed through the rest of that topic's problems. As well, what constitutes "high" in terms of numbers of compiles, runs and hints? Students may be compiling after each line of code written, but never encountering compile errors. A more useful measurement than merely the total number of compiles would be the ratio of compile attempts with errors to error-free compiles (see Future Work section in Chapter 7 for further discussion of this).

Looking deeper into the data for these two topics, on a question by question basis we see that the highest percent correct for any Loops question is 53.5%, which is only slightly greater than the lowest percent correct of any Methods question, at about 50.6% (see Figure 6.1).

If we consider numbers of hints requested per question in these two topics, we

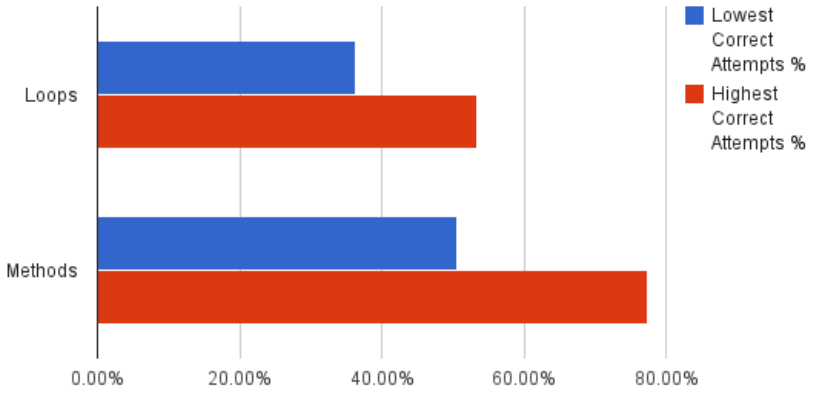

Figure 6.1: Questions with the Lowest and Highest Correct Attempts in Loops and Methods Topics

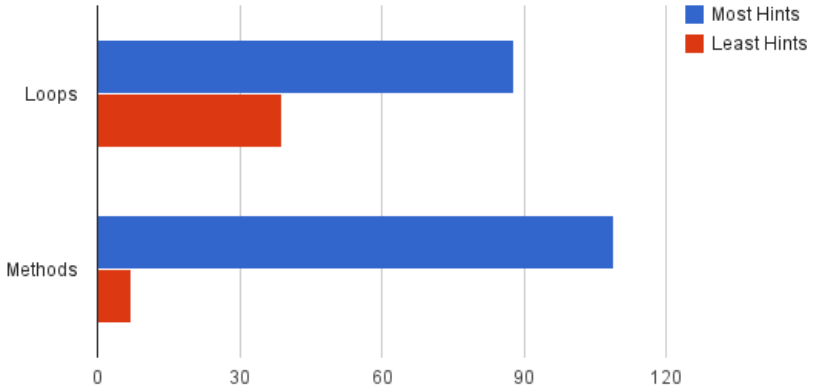

Figure 6.2: Questions with the Lowest and Highest Numbers of Hints Requested in Loops and Methods Topics

see that while two Methods questions had a high number of hints requested, the remaining 6 questions only had between 3 and 35 hints requested. This is in contrast to the Loops topic, for which all questions attempted had between 39 and 88 hints requested — a greater number of questions with high numbers of hints requested (see Figure 6.2).

The data for the Loops topic may suggest, with students attempting only half of the questions, requesting many hints on all questions, and achieving only 44% correct

answers on their attempts, that some number of students are struggling with the topic. In contrast, the data for the Methods topic seems to suggest that students had less difficulties with this topic, given that students attempted on average 65% of Methods questions, requested fewer hints than other topics, in general, and achieved 61.5% correct answers on their attempts .

In conclusion, as the above paragraphs demonstrate, the data gathered from the BitFit logs may suggest that students struggled less with the Methods topic than they did with Loops.

To generalize the above conclusions and answer Research Question 1:

1. Students attempting low numbers of questions in a topic, low percentages of correct attempts, plus high numbers of hints requested on all or most of a topic's questions, together with high numbers of compiles and runs could imply that students are struggling with that topic.

2. Students attempting greater numbers of questions in a topic, high percentages of correct attempts, plus few hints requested on all or most of a topic's questions, right from the early days of students attempting that topic's questions, could imply that students have mastered that topic, or that the questions for that topic are too easy.

3. Outliers in a topic's question set could imply that the question is either too easy (in the case of low hints requested and high correct percentages, when all other questions in the set have the opposite), or too hard (if it alone has high numbers of hints requested, compiles, runs and incorrect attempt percentages).

Threats to validity for the above points include the fact that the group who partook in our data collection did so voluntarily, and therefore may not have been a highly representative sample of the whole.

6.1.2 Research Question 2: What data could help instructors more precisely identify at-risk students, why they are at risk, and allow for more effective interventions relative to existing approaches?

Methodology to Answer Research Question

To answer this question we identify the range of data that a tool like BitFit can offer to instructors who might use it to help identify and intervene with students at risk of falling behind or failing the course.

The data set presented and analyzed in this thesis does not include individual user data — the individual data has all been aggregated together into question and topic groups. While the deep analysis of individual student data is outside the scope of this thesis, in order to highlight what might be possible in future work, we chose to include a small sampling of individual data in Tables 5.4 and 5.5 in Section 5.2, Tool Usage By User. This logging data details two students' interactions with the tool during one session of BitFit. The students were attempting questions in the Loops topic. While such a small sampling means we cannot draw any firm conclusions, the data might indicate that Student A did quite well on this topic, with one question requiring a little more work, but ultimately, the student was able to find the correct answer for all the questions attempted, without needing to request any hints. On the other hand, Student B appears to have struggled with these questions. Student B had high numbers of compiles, with relatively few runs in comparison, possibly indicating many errors in their code. The student requested many hints on each question, and on two questions was not able to successfully arrive at the correct answer.

This sampling of individual level data collected by BitFit demonstrates that this type and level of detail of data may provide more opportunity for instructors to be in tune with students who are struggling the most in the course and at risk of failing.

To further illustrate the possibility of using the data from BitFit in such a way, we consider three possible at-risk student cases that would warrant instructor intervention, and how the tool's data would assist in identifying these cases. We first briefly consider the case where a student doesn't use the tool at all, before moving on to three more complex case studies:

1. A student who uses the tool initially, achieving very low correct percentage attempts, and then stops using the tool altogether.

2. A student who doesn't start working through the tool's questions early enough.

3. A student who very quickly worked through all questions in the tool, accessing all hints for all questions.

In the next section we consider which data from the tool aids in identifying students who might fall into these cases of potential risk.

Relevant Results

Identifying cases where students don't use the tool at all is simply a matter of looking at the users for whom data is entered in the tool's logs. In our study, where use of the tool was completely voluntary and outside of the regular lecture and lab work, we recorded data for 115 users, of a possible 197 students registered in the course. There were, therefore, 82 students who did not use the tool.

For the at-risk student described in case 1, the tool's data would show interaction points for the student early on in the semester, possibly with high numbers of compiles and runs, or also with low numbers, if the student is not even attempting the code writing questions. Likely the hints requested would be high for many questions, and, most importantly in identifying this case, there would be low percentages of correct attempts, if any are, in fact, correct. After this initial burst of activity, interaction points would tail off to zero. As the course progresses and new topics are added to the tool, another strong indicator of this case would be interaction data that shows the student rapidly jumping through many questions in new topics added, with no attempts actually made to answer those questions.

At-risk student case 2 would have no interaction points in the tool's logs during the first days or weeks of the course, and then an intense burst of interaction perhaps the day or night before a midterm. There would be high numbers of hints requested on many questions, and low percentages of correct question attempts.

Finally, for case 3, the tool's data would show a burst of interaction points over a short time period, with the maximum number of hints requested for every question in every topic. Percentages of correct attempts would be high, and time spent on the tool would be low.

How Do Results Answer the Research Question?

For case 1 the initial burst of activity, taken together with the subsequent lack of attempts to answer questions, could illustrate a student who started out trying to answer the questions. As she progresses, she requests many hints when she finds herself unable to answer questions correctly. Finally, the student gives up completely as she sees new questions she feels unable to event attempt to answer correctly.

In case 2, we can imagine a student who finds herself completely unprepared for the midterm, after not making any attempts at practice from the early days of the course. While this student may well have a chance of success, the mere fact of having not left herself enough time to prepare for the midterm means she will not do well on it.

For case 3, while percentages of correct attempts would be high, they are also suspect. It is possible the student knows the material very well and merely wanted to see what hints were available, but also very possible that the student was merely trying to "game the system" to get the most assistance without actually trying or ensuring they understand the material on their own.

Instructors could set up thresholds for when students may be at risk of falling behind in the course. Any of the scenarios outlined in the preceding section could be reason to flag a student at risk, and could be seen via the data from BitFit's logs.

In future, data from a tool like BitFit could be used to support an experiment that would answer this research question more deeply. One possible approach for identifying at-risk students is for instructors to use the tool from the first week of class, and require students to work through the question sets, perhaps during some portion of the weekly lab. If there were a small amount of marks contingent upon student use of the tool (but not upon the correctness of student answers), more students would be compelled to use the tool.

6.2 Collecting Data to Help Students

Research Question 3 turns the tables to focus on students, and how a tool like BitFit can be used to improve their experience — both academically, as well as experientially, relative to existing course offerings. The following sections describe our process for answering Research Question 3. As with the previous Research Questions, we first present the methodology used to answer the question, then highlight the relevant data

from the Results chapter, and finally identify how these results answer the question.

6.2.1 Research Question 3: Can the use of a tool like BitFit offer students an improved learning experience relative to existing offerings?

With this section of the research, we seek to answer the following questions: Can a tool like BitFit help a student know where they stand in relation to the course content? Can a tool like BitFit help students understand where their strengths and weaknesses lie? Can we measure and accurately convey at what point students have mastered a particular topic?

Methodology to Answer Research Question

We use a mixed methods approach to answer this question, combining quantitative data from the tool with qualitative data from the student survey.

Quantitatively, the tool provided students with instantaneous feedback for their attempts to answer questions (described in detail in Sections 5.2 through 5.4). The setup of the tool allowed for questions to be grouped by topic, and we structured the topics to be roughly equivalent to each unit of learning in the course. If a student had successfully answered all questions in a topic set, they could feel confident they had a good grasp of that particular topic. As well, two of the topics for the first midterm — "Everything Combined: Writing Code" and "Everything Combined: Tracing" — consisted of questions that required students to exercise their knowledge of all the other topics combined (Print Statements, Methods, If Statements and Loops).

The types of questions we included in the tool could specifically be answered in a short time — a few minutes at most, if the student understood the material. This was an intentional choice, to make the questions similar to the types of questions students could expect to see on their midterms. Students could therefore feel confident that they were effectively using their time on the tool to study material that would help them succeed on the midterm.

Currently, the only way we present feedback to a student on their individual progress is via a dialogue box that appears each time they check their answer to the current question, that tells them whether or not their attempt is correct. However, given that the tool's logs collect data on the whole of the student's progress, it will

be relatively simple in future work to add the functionality for a student to view information about their progress on all questions in the tool.

In the next section we examine the qualitative data results from the student survey, to understand how students felt the tool impacted their understanding of course material, plus their feelings about and readiness for their midterms, after using the tool.

Relevant Results

The following paragraphs highlight responses from the student survey that surface several themes to aid in answering this research question. Responses have been selected as a summary representation of the body of long answer student responses.

The first question we examine is: **Did the programming practice tool help you get an idea of where you stand with respect to the course content? Why or why not?**

Theme 1: Instantaneous feedback and readiness for midterms.

> *- Yes, provided immediate feedback on what concepts I had trouble with*
>
> *- Yes, because it actually checked if I understood, and could do what was asked of me*
>
> *- Yes it did, because at the beginning I knew nothing and it really showed and now I know a little bit. I don't get the same error messages I use to!*
>
> *- Yes. because when i could do the questions without problems i knew i was ready for the test*
>
> *- Yes, as it is comprised of similar questions to midterm, I could get an idea of where I am at with my progress.*
>
> *- Yes — the difficulty level of the programming practice tool was much closer to the midterm difficulty level than was the textbook practice.*
>
> *- Yup. If you can do the questions you can probably do the ones on the midterm as well*

Theme 2: Division of questions by topic and combined section — consistency with course design.

- Yes because it progressed with the course so it was easy to check your own progress and understanding

- Yes, I feel it did. It synthesized stuff that I hadn't seen before which was good practice.

- It did because of the sectioned layout; I think having a section for each chapter, and then combining them all together with tracing and writing code is very effective to help sort out where I was uncomfortable with course material.

Theme 3: Enabling students to self-identify strengths and weaknesses, and work on weaknesses.

- Helped me understand what i needed to work on

- Yes, I can find my weakness by using this tool

- Yes, it helped with understanding what I needed to practice more of and what I was comfortable with

- Yes, it helped me realize that I still had more to learn in some cases, and helped me practice that.

- Yes because if I could look at a problem and knew how to do it, then I mostly understood the concepts.

Theme 4: Practice on programming "in the small".

- It was helpful in catching some small problem areas and made me think a little more. It was also good practice in solving novel small problems rapidly rather than the large scale types of problems presented in assignments.

- Yes. it was good to tie it all together in questions that weren't as big as the assignment

Theme 5: Areas of weakness students found with the tool.

- Somewhat... a "solutions manual" is needed in order to compare why our codes were falling short.

- In the first half of the course this statement holds true, but as the tone of the course shifted from displaying that you knew how to implement

certain features into your code, and transitioned into problem solving with the material, this tool was less capable of giving me a clear idea of how good my grasp on the material was.

- Yes somewhat, if the practice tool was more structure to how the textbook is laid out that would be good.

- Yes, but not so much. The only feedback given after completing question was whether or not the question was correct. Specific feedback would have helped a bit more.

- Kind of. When I pressed the hint button sometimes it would not explain how to solve the question and so I was always left guessing and questioning myself whether I was putting down the right answer.

We next examine the responses to the question: **Did the programming practice tool have any affect on how you felt before writing either midterm? Why or why not?**

As described in the results section 5.6, there were 54 responses to this question, of which 45 indicated a positive effect resulting from use of the tool. The following is a sampling of these positive responses:

- Improved my confidence in my programming ability

- It boosted my confidence, as I felt I had adequate practice and understanding.

- I found that the programming practice tool decreased my anxiety heading into both midterms, as I felt that it was a source of higher-level questions than those found in the textbook.

- Yes, slightly more confident on both midterms knowing I had practiced the material beforehand.

- 1st midterm — for sure helped calm anxiety. Had a lasting effect going into midterm 2

- It made me more confident in the exam because i had an idea what questions might be on it

- It definitely lowered my anxiety about the exams, and was a huge confidence booster to see myself answering correctly.

Six students indicated that the tool did not help how they felt going into the exams, while several others felt ambivalent about the tool's effect. The following is a sampling of these responses:

> *- Midterm 1: increased confidence. Midterm 2: I was a little more concerned*
>
> *- It contained some difficult questions that made me worried, however, it helped me to learn a lot.*
>
> *- No, because i did not use it very much*
>
> *- If the examples were hard and I couldn't figure some of them out, it would cause me to worry for the midterm. But at least it was a good indicator of what would be on the midterm so I felt a little more at ease. - Before first: confidence, because I felt prepared. Before second: panic, because the online problems were not working properly and I didn't know if I was getting them correct. I also expected to use the tool's background section and hints to help me prepare, and was a lot more nervous knowing that these had been helpful before and were then missing.*

This final response bears further examination. The problem the student refers to was the result of an issue with the tool that arose when multiple users were working on a question that used a shared file as input. This is an issue that has been fixed for future work, but is of concern, and worth noting, given that this student's emotions were clearly affected in a negative way by the situation.

Several students raised the very valid point that while practicing writing code on the computer can be helpful, it doesn't replicate the exam setting, where students must write code with pencil and paper:

> *- The practice tool did raise my confidence but I still was very anxious and stressed for the midterm. Since I was very used to practicing on the computer, I was worried that I will not do so well when writing on paper.*

How Do Results Answer the Research Question?

In general, students indicated that they liked using the tool, and found it improved their experience in the course. As highlighted in the previous section, the students said this was due to the targeted extra practice, instantaneous feedback, and by the

division of questions both by topic to mimic the course flow, and by topics that combine many pieces of knowledge from different topics. They also appreciated that the tool provided practice questions that required similar effort to what would be expected of them in the midterms — smaller and shorter than assignments. Yet it is important to note that use of the tool does not replace practice with pen and paper, which should also make up a student's preparations for midterms. The majority of students who responded felt that using the tool helped their confidence going into their midterms, though there were some students who felt otherwise.

Students also provided useful suggestions for future work: namely to allow for "bottoming-out" type hints that provide the correct answer to a question after a student has attempted multiple times, unsuccessfully, to answer the question, and exhausted all hints available to them.

The response samples in the previous section demonstrate that, in general, students feel a tool like BitFit can help them know where they stand in relation to the course content, understand where their strengths and weaknesses lie, and improve how they feel as they prepare for their exams, thus improving their overall experience in the course.

Chapter 7

Future Work and Conclusions

7.1 Future Work

The work described in this thesis gives rise to the following questions which we plan to address in future work.

First, we address ways to further assist instructors improve the quality of instruction:

- Can the data be mined for further insights?
- Does the number of times students compile and run code writing questions indicate more or less understanding?
- Does the number of compiles and runs of a coding question indicate student struggles and/or ability in answering that question?
- Does measuring the amount of time spent on a topic area give any indication of how comfortable or not students are with that topic?
- Is there other data we should be collecting (for instance compile time programming error types, and numbers, runtime error types and numbers)?
- Can a tool like BitFit be effective both as a group learning tool and for individual learning?
- Can a tool like BitFit be used in a flipped classroom approach to teaching? In a MOOC?
- Is the tradeoff of more work for the professor worth it for the potential gains?
- Could the inclusion of a tool like BitFit into course offerings help convey instructor empathy toward students and instil greater student confidence?

Next, we present some of the features we plan to add to BitFit with the goal of

further improving both the instruction quality and the student experience:

- Interactive graphs for students to view their own progress through the tool, as well as aggregated data of their classmates' progress.
- Interactive graphs for instructors to view student progress in real time.
- Support for adding test cases to questions, and evaluator programs.
- Hints can be added as HTML, not just plain text.
- Ability for students to save their work.
- Ability for instructors to save questions in progress as they are being added to the tool.
- Ability for instructors to import and export question sets.
- "Bottoming-out" hints - after students have exhausted all hints for a question, and made some minimum number of viable attempts to solve the problem without success, BitFit can provide the correct answer for them.
- Adding a box to read-only questions to allow students to specify the input to the program. Then, based on that input, students will be asked to type the program's output into another dialog box. This will allow for greater flexibility in terms of the types of code reading questions instructors can ask, and encourage a greater level of active code reading on the part of the students.
- In addition to logging data for students and instructors to view later, an adaptive system could also potentially use the same data to create a customized learning experience for every user (this is a key feature of Intelligent Tutoring Systems — the ability of the system to adapt the content of the lesson and the difficulty level to the individual student.

7.2 Concluding Remarks

The growing distance in the relationship between students and instructors seems to correlate with the continual growth of retention challenges in CS, particularly for underrepresented groups. This thesis proposes an all encompassing approach that works within the current reality of growing class sizes and less human interaction to address this distancing. We demonstrate that the introduction of tools such as BitFit into introductory programming courses, with the conscious intention of using telemetry from the tool to directly feed the progression of the course, can improve the current situation in three distinct ways: First, instructors can use the data to

improve the quality of their instruction by allowing for a data and student-driven approach to their teaching, identifying areas of student struggles and ensuring all student questions are addressed. Second, the data — used together with traditional assessment methods — can help instructors to identify at-risk students early enough to make positive interventions. Finally, student experiences in their courses can be improved through use of tools such as BitFit: not only can their performance be improved, but their confidence, motivation, metacognition, and awareness of the fact that their instructors care about their success can too. We believe that this approach can begin to address the challenges of retention by building a greater sense of empathy between the instructor and students, thus lessening the distance between them.

Appendix A

Additional Information

A.1 Topic Summary Data

Date	Total Qs Attempted	Total Users	Average Qs Attempted per User per Day	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
2/12/2015	28	6	4.67	12	11	0	32	23
2/13/2015	51	8	6.38	8	8	11	91	50
2/14/2015	92	14	6.57	118	102	18	121	88
2/15/2015	258	37	6.97	600	444	151	674	408
2/16/2015	61	11	5.55	7	8	109	103	44
⋮								
2/24/2015	6	1	6	0	0	4	0	0
2/25/2015	0	0	0	0	0	0	0	0
2/26/2015	9	1	9	0	0	0	9	8
⋮								
3/7/2015	8	1	8	2	2	0	9	8
3/8/2015	17	3	5.67	0	0	0	10	8
3/9/2015	8	1	8	4	3	1	14	9
3/10/2015	5	1	5	0	0	0	0	0
3/11/2015	28	4	7	4	2	0	25	23
3/12/2015	5	2	2.5	21	19	16	34	21

Table A.1: Methods Topic Summary Data

Date	Total Qs Attempted	Total Users	Average Qs Attempted per User per Day	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
2/12/2015	44	10	4.4	45	39	38	103	61
2/13/2015	68	11	6.18	5	1	11	118	64
2/14/2015	34	7	4.86	0	0	7	65	33
2/15/2015	214	39	5.49	208	116	123	777	325
2/16/2015	34	7	4.86	35	25	12	52	34
2/17/2015	0	0	0	0	0	0	0	0
2/18/2015	0	0	0	0	0	0	0	0
2/19/2015	9	2	4.5	0	0	2	22	6
⋮								
2/24/2015	5	1	5	0	0	0	0	0
2/25/2015	4	1	4	0	0	0	0	0
2/26/2015	2	1	2	0	0	0	2	2
⋮								
3/4/2015	6	1	6	0	0	0	11	6
3/5/2015	0	0	0	0	0	0	0	0
3/6/2015	2	1	2	0	0	0	4	2
3/7/2015	14	4	3.5	4	6	0	19	13
3/8/2015	7	2	3.5	0	0	0	9	6
3/9/2015	1	1	1	0	0	0	0	0
3/10/2015	7	2	3.5	0	0	0	8	6
3/11/2015	36	7	5.14	0	0	15	68	31
3/12/2015	7	2	3.5	0	0	0	10	6
⋮								
3/19/2015	6	1	6	0	0	1	12	6

Table A.2: Print Statements Topic Summary Data

Date	Total Qs Attempted	Total Users	Average Qs Attempted per User per Day	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
2/12/2015	52	10	5.2	170	141	10	115	58
2/13/2015	66	14	4.71	213	143	23	114	56
2/14/2015	81	12	6.75	349	227	23	143	78
2/15/2015	253	43	5.88	2507	1717	357	1230	487
2/16/2015	67	13	5.15	116	84	37	105	56
2/17/2015	0	0	0	0	0	0	0	0
2/18/2015	3	1	3	0	0	0	0	0
2/19/2015	1	1	1	0	0	0	0	0
2/20/2015	4	1	4	17	10	2	27	11
2/21/2015	0	0	0	0	0	0	0	0
2/22/2015	0	0	0	0	0	0	0	0
2/23/2015	2	1	2	54	40	3	27	11
2/24/2015	8	1	8	92	73	5	39	18
2/25/2015	0	0	0	0	0	0	0	0
2/26/2015	0	0	0	0	0	0	0	0
2/27/2015	1	1	1	0	0	0	0	0
⋮								
3/5/2015	7	1	7	0	0	0	0	0
3/6/2015	2	1	2	2	2	0	1	1
3/7/2015	28	4	7	29	23	1	20	14
3/8/2015	14	3	4.67	81	52	4	30	16
3/9/2015	4	1	4	6	6	0	3	2
3/10/2015	1	1	1	0	0	0	0	0
3/11/2015	37	8	4.63	78	63	17	62	34
3/12/2015	5	2	2.5	3	3	0	3	3

Table A.3: Loops Topic Summary Data

Date	Total Qs Attempted	Total Users	Average Qs Attempted per User per Day	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
2/12/2015	26	4	6.5	4	1	0	38	21
2/13/2015	27	6	4.5	0	0	4	29	17
2/14/2015	65	12	5.42	45	63	12	201	87
2/15/2015	225	40	5.63	850	523	156	1113	562
2/16/2015	65	14	4.64	82	60	31	168	82
2/17/2015	0	0	0	0	0	0	0	0
2/18/2015	3	2	1.5	0	0	0	0	0
⋮								
3/7/2015	7	2	3.5	0	0	0	8	7
3/8/2015	7	2	3.5	0	0	0	9	7
3/9/2015	6	1	6	0	0	1	13	6
3/10/2015	7	2	3.5	0	0	0	9	5
3/11/2015	29	7	4.14	0	0	0	30	24
3/12/2015	8	2	4	0	0	0	12	5

Table A.4: If Statements Topic Summary Data

Date	Total Qs Attempted	Total Users	Average Qs Attempted per User per Day	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
2/12/2015	12	4	3	43	31	0	11	8
2/13/2015	40	10	4	109	91	9	81	22
2/14/2015	50	12	4.17	210	126	17	24	13
2/15/2015	215	55	3.91	1579	985	222	667	339
2/16/2015	54	17	3.18	327	226	46	127	81
2/17/2015	0	0	0	0	0	0	0	0
2/18/2015	6	2	3	0	0	0	0	0
⋮								
2/25/2015	5	1	5	7	3	0	0	0
2/26/2015	0	0	0	0	0	0	0	0
2/27/2015	2	1	2	0	0	0	0	0
⋮								
3/7/2015	5	1	5	12	9	0	8	6
3/8/2015	9	3	3	0	0	0	1	1
3/9/2015	4	2	2	44	18	0	9	2
3/10/2015	2	1	2	8	7	0	3	1
3/11/2015	43	13	3.31	293	208	32	17	13
3/12/2015	6	3	2	8	5	5	15	11

Table A.5: “Everything Combined: Writing Code” Topic Summary Data

Date	Total Qs Attempted	Total Users	Average Qs Attempted per User per Day	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
2/12/2015	7	2	3.5	7	5	2	15	6
2/13/2015	38	10	3.8	89	79	17	124	74
2/14/2015	26	7	3.71	15	15	5	63	35
2/15/2015	153	44	3.48	839	428	207	929	377
2/16/2015	54	15	3.6	26	16	32	78	38
⋮								
3/2/2015	3	1	3	0	0	6	13	1
⋮								
3/8/2015	8	2	4	0	0	0	12	6
3/9/2015	8	2	4	0	0	2	16	7
3/10/2015	5	2	2.5	0	0	0	5	2
3/11/2015	46	19	2.42	67	29	23	59	33
3/12/2015	4	3	1.33	0	0	0	0	0

Table A.6: "Everything Combined: Tracing" Topic Summary Data

Date	Total Qs Attempted	Total Users	Average Qs Attempted per User per Day	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
3/7/2015	17	4	4.25	60	15	1	32	10
3/8/2015	99	24	4.13	164	84	39	128	32
3/9/2015	13	3	4.33	15	7	4	14	6
3/10/2015	65	15	4.33	153	71	15	66	32
3/11/2015	225	49	4.59	637	283	140	312	71
3/12/2015	75	18	4.17	89	44	31	138	46
3/13/2015	1	1	1	0	0	0	0	0

Table A.7: I/O Tracing & Coding Topic Summary Data

Date	Total Qs Attempted	Total Users	Average Qs Attempted	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
3/7/2015	11	4	2.75	13	1	0	3	2
3/8/2015	102	22	4.64	109	42	10	55	24
3/9/2015	17	5	3.4	24	8	0	15	10
3/10/2015	34	9	3.78	79	44	11	21	18
3/11/2015	186	42	4.43	538	237	67	190	83
3/12/2015	70	24	2.92	175	105	23	74	26

Table A.8: Arrays Topic Summary Data

A.2 Usage Data by Topic and Questions

A.2.1 Print Statements

User interaction for this topic peaked at two separate times: first, on February 15, the day of the first midterm review session and the day before Midterm 1, and second on March 11, the day before Midterm 2. There were also increases in usage on March 7.

On February 15, a total of 214 question attempts by 39 users were made, an average of 5.49 questions attempted per user. On that day, students compiled their code 208 times, ran it 116 times, and made a total of 123 requests for hints. This day saw 777 attempts to answer questions on Print Statements, 325 of which (41.8%) were correct.

On March 11, a total of 36 question attempts by 7 users were made, an average of 5.14 questions attempted per user. On that day, students compiled their code 0 times, ran it 0 times, and made 15 requests for hints. There were 68 attempts to answer Print Statement questions, 31 of which (45.6%) were correct.

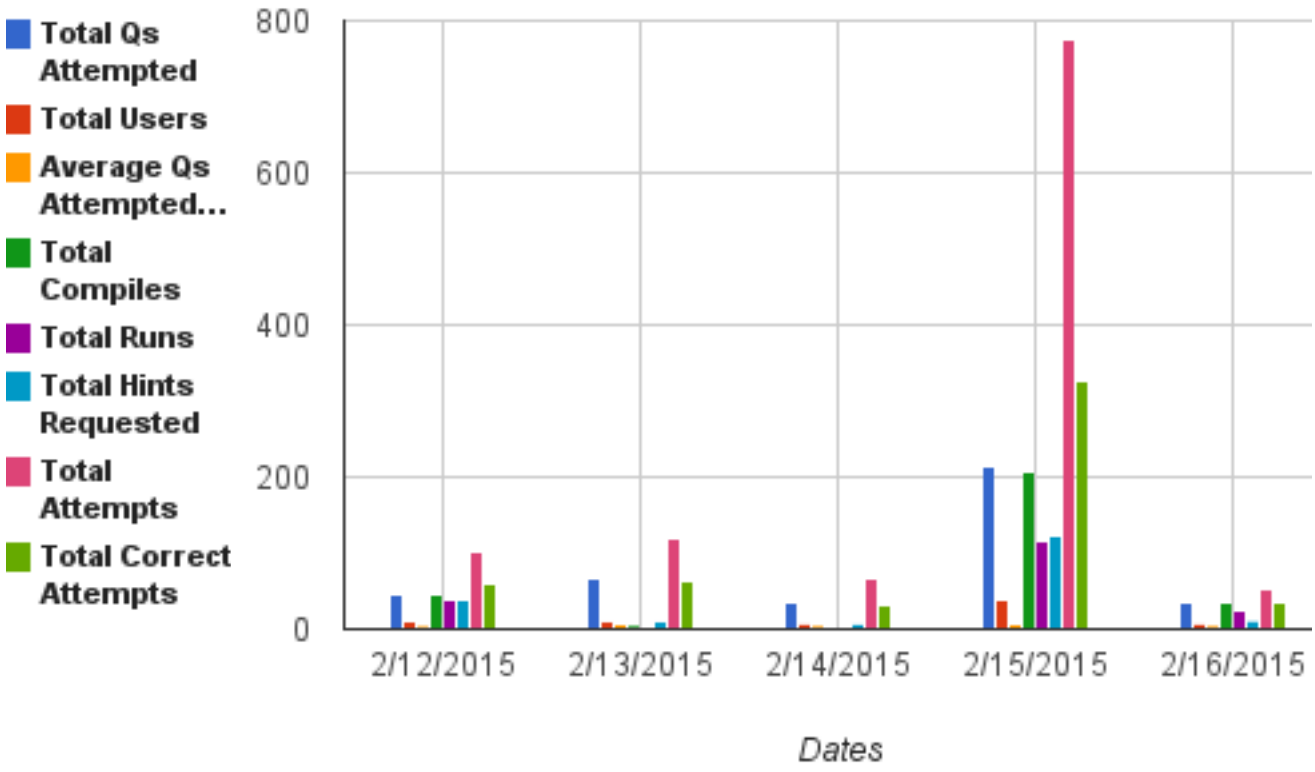

Figure A.1: Print Statements Summary Data - Part A (see Table A.2 for corresponding data values)

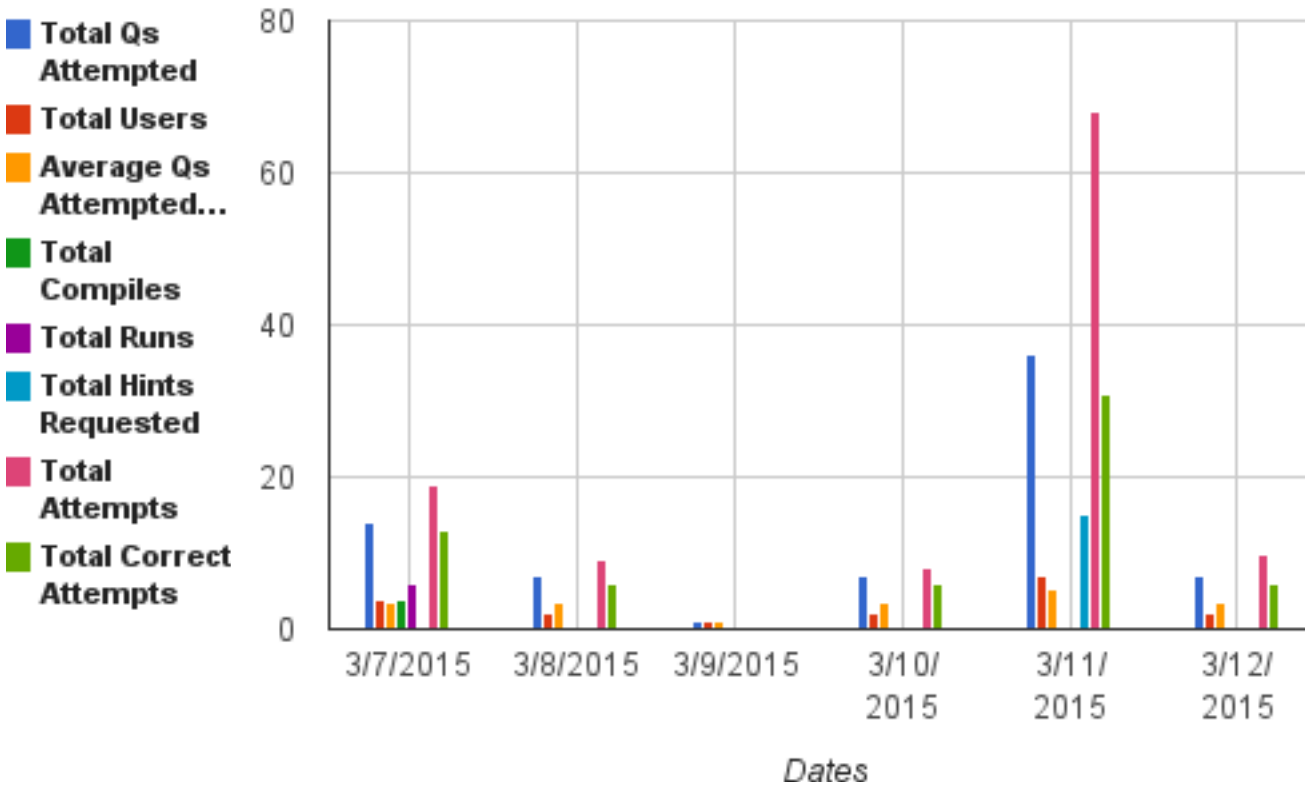

Figure A.2: Print Statements Summary Data - Part B (see Table A.2 for corresponding data values)

Print Statements Question Summary Data

Question number 2 had the greatest number of compiles and runs at 128 and 75, respectively. The most hints were requested for question 6 (53). Number 2 also had the highest number of attempts at 249, of which 98 (39%) were correct. The greatest number of correct attempts was for question 5, with 107 of 188 attempts correct (57%).

February Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	2	2	0
2	128	75	45	249	98
3	60	38	20	179	93
4	22	15	26	126	68
5	67	47	26	188	107
6	4	1	53	244	90
7	12	5	21	151	69
March Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	0	0	0
2	4	6	1	39	16
3	0	0	5	20	14
4	0	0	5	23	11
5	0	0	1	25	12
6	0	0	4	22	14
7	0	0	0	12	9

Table A.9: Print Statements Question Summary Data

A.2.2 Loops

User interaction for this topic peaked at two separate times: first, on February 15, the day of the first midterm review session and the day before Midterm 1, and second on March 8 and 11, the day before Midterm 2. There were also increases in usage on March 7.

On February 15, a total of 253 question attempts by 43 users were made, an average of 5.88 questions attempted per user. On this day, students compiled their code 2507 times, ran it 1717 times, and made a total of 357 requests for hints. This day saw 1230 attempts to answer questions on Loops, 487 of which (40%) were correct.

On March 8, a total of 14 question attempts by 3 users were made, an average of 4.67 questions attempted per user. On this day, students compiled their code 81 times, ran it 52 times, and made 4 requests for hints. There were 30 attempts to answer Loops questions, 16 of which (53%) were correct. On March 11, 37 questions were attempted by 8 users, compiling and running their code 78 and 63 times, requesting 17 hints, and making 62 attempts to answer questions, 34 of which (55%) were correct.

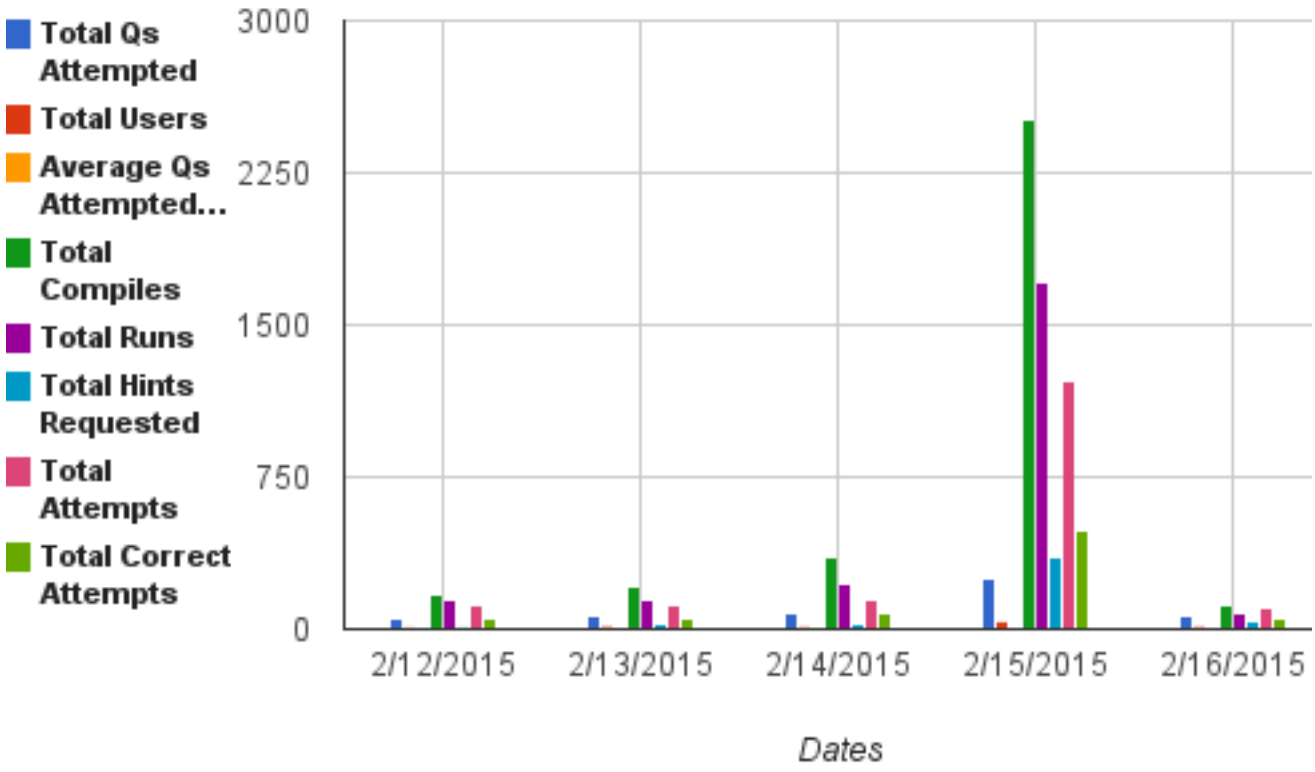

Figure A.3: Loops Summary Data - Part A (see Table A.3 for corresponding data values)

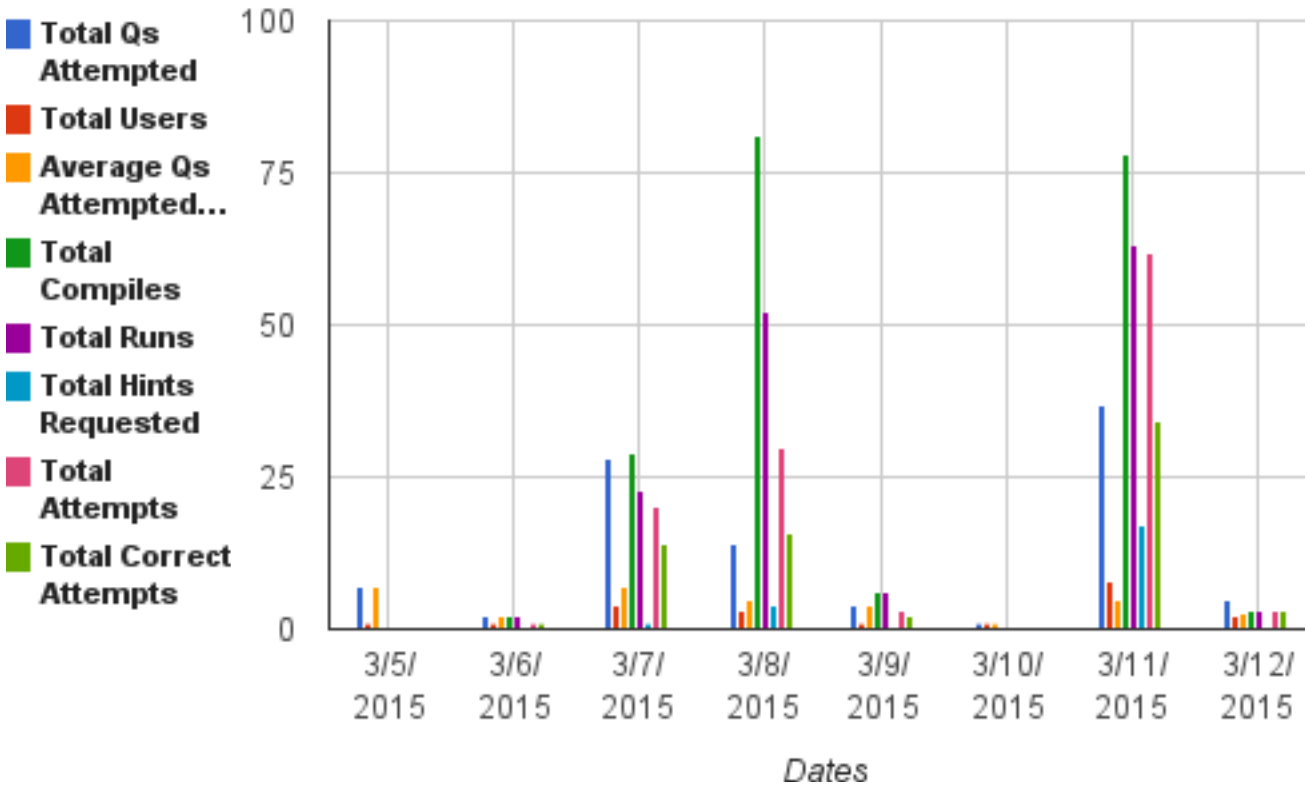

Figure A.4: Loops Summary Data - Part B (see Table A.3 for corresponding data values)

Loops Question Summary Data

Question number 7 had the greatest number of compiles at 632 (and 426 runs), while question 8 had the greatest number of runs at 462 (with 591 compiles). The most hints were requested for question 8 (86). Question number 2 had the highest number of attempts at 423, of which 154 (36.4%) were correct - the lowest number of correct attempts per question. The greatest number of correct attempts was for question 3, with 84 of 157 attempts correct (53.5%).

February Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	23	22	7	6	4
2	432	236	88	423	154
3	350	261	39	157	84
4	461	313	55	212	96
5	525	352	49	172	89
6	504	363	69	209	97
7	632	426	67	322	130
8	591	462	86	299	121
March Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	0	0	0
2	51	32	8	48	19
3	14	13	0	10	10
4	34	21	0	8	8
5	33	22	4	11	6
6	34	29	1	14	7
7	12	12	0	7	4
8	21	20	9	21	16

Table A.10: Loops Question Summary Data

A.2.3 Methods

User interaction for this topic peaked at two separate times: first, on February 15, the day of the first midterm review session and the day before Midterm 1, and second on March 11 and 12, the day before and the day of Midterm 2. There were also increases in usage on February 7-9 and March 14.

On February 15, a total of 258 question attempts by 37 users were made, an average of 6.97 questions attempted per user. On this day, students compiled their code 600 times, ran it 444 times, and made a total of 151 requests for hints. This day saw 674 attempts to answer questions on Methods, 408 of which (60%) were correct.

On March 11, a total of 28 question attempts by 4 users were made, an average of 7 questions attempted per user. On this day, students compiled their code 4 times, ran it 2 times, and made 0 requests for hints. There were 25 attempts to answer Methods questions, 23 of which (92%) were correct. On March 12, 5 questions were attempted by two users, compiling and running their code 21 and 19 times, requesting 16 hints, and making 34 attempts to answer questions, 21 of which (61.8%) were correct.

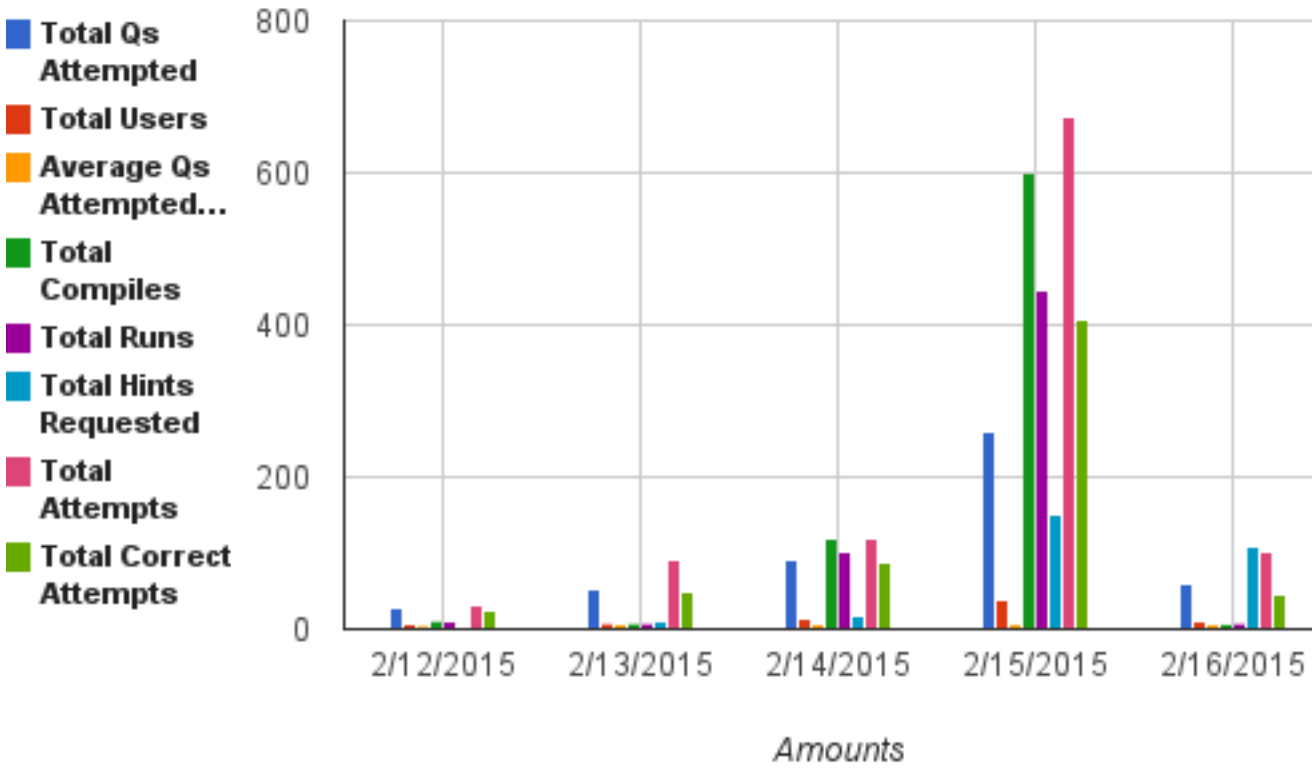

Figure A.5: Methods Summary Data - Part A (see Table A.1 for corresponding data values)

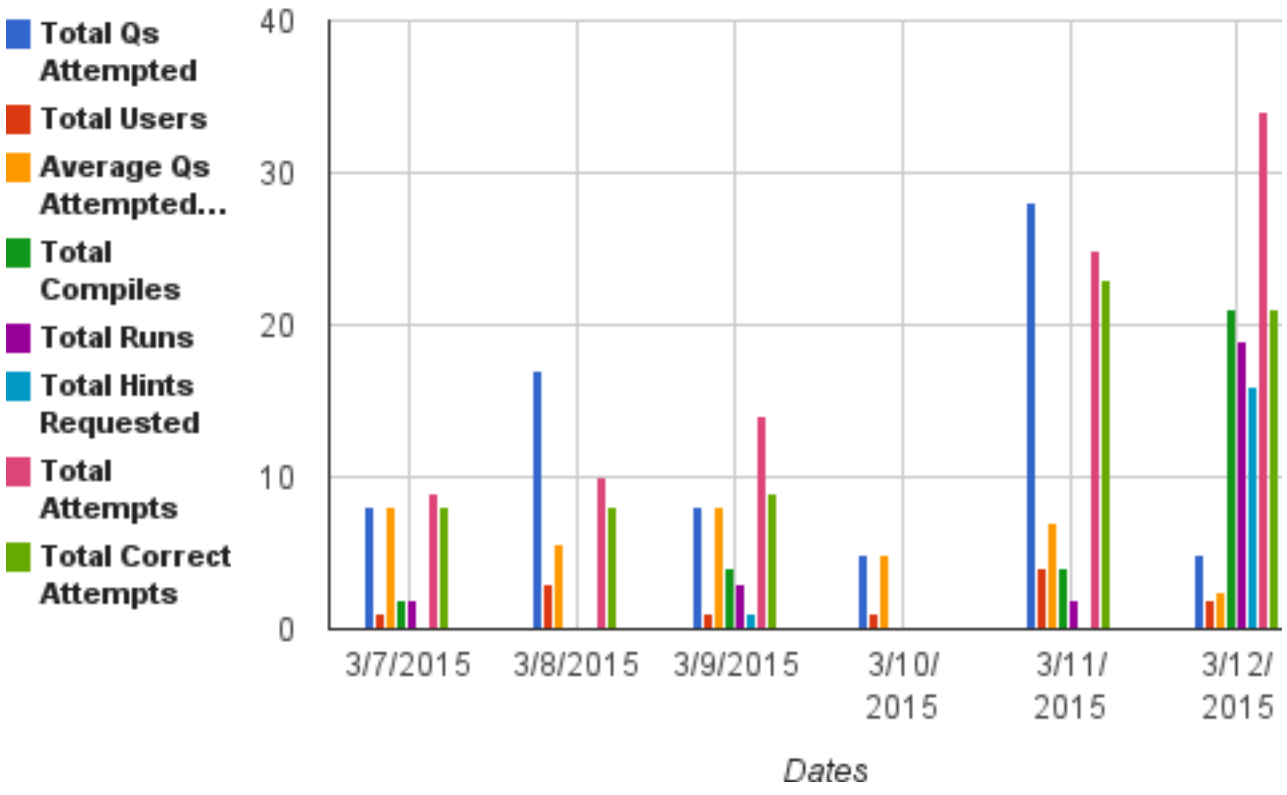

Figure A.6: Methods Summary Data - Part B (see Table A.1 for corresponding data values)

Methods Question Summary Data

Question number 2 had the greatest number of compiles and runs at 288 and 242 runs respectively. The most hints were requested for question 4 (109). Question number 8 had the highest number of attempts at 247, of which 125 (50.6%) were correct. The greatest number of correct attempts was for question 6, with 72 of 93 attempts correct (77.4%). It is worth noting that question 6 saw a spike in attempts in March, with 21 compiles and runs, respectively, 16 hints requested, and 39 attempts to answer (of which 25, or 64.1% were correct).

February Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	0	0	0
2	288	242	92	123	79
3	3	4	3	108	72
4	11	11	109	149	76
5	89	66	20	94	63
6	5	1	7	93	72
7	3	4	12	81	57
8	229	165	35	247	125
9	117	80	15	135	77
March Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	0	0	0
2	10	7	0	12	10
3	0	0	1	10	7
4	0	0	0	6	6
5	0	0	0	9	6
6	21	19	16	39	25
7	0	0	0	5	4
8	0	0	0	6	6
9	0	0	0	5	5

Table A.11: Methods Question Summary Data

A.2.4 If Statements

User interaction for this topic peaked at two separate times: first, on February 15, the day of the first midterm review session and the day before Midterm 1, and second on March 11, the day before Midterm 2 was held, with increases in usage on March 7-10 and March 12.

On February 15, a total of 225 question attempts by 40 users were made, an average of 5.63 questions attempted per user. On this day, students compiled their code 850 times, ran it 523 times, and made a total of 156 requests for hints. This day saw 1113 attempts to answer If Statements questions, 562 of which (50%) were correct.

On March 11, a total of 29 question attempts by 7 users were made, an average of 4.14 questions attempted per user. On this day, students compiled their code 0 times, ran it 0 times, and made no requests for hints. There were 30 attempts to answer If Statements questions, 24 of which (80%) were correct.

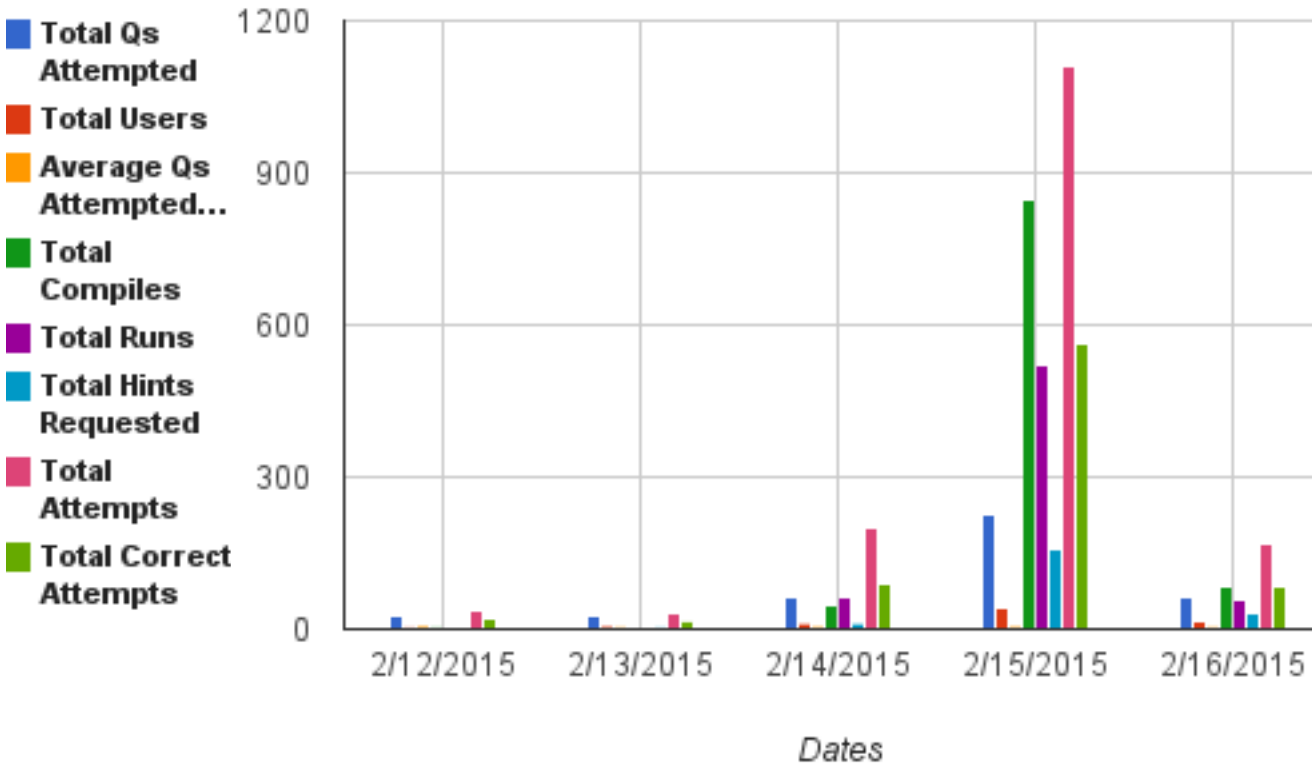

Figure A.7: If Statements Summary Data - Part A (see Table A.4 for corresponding data values)

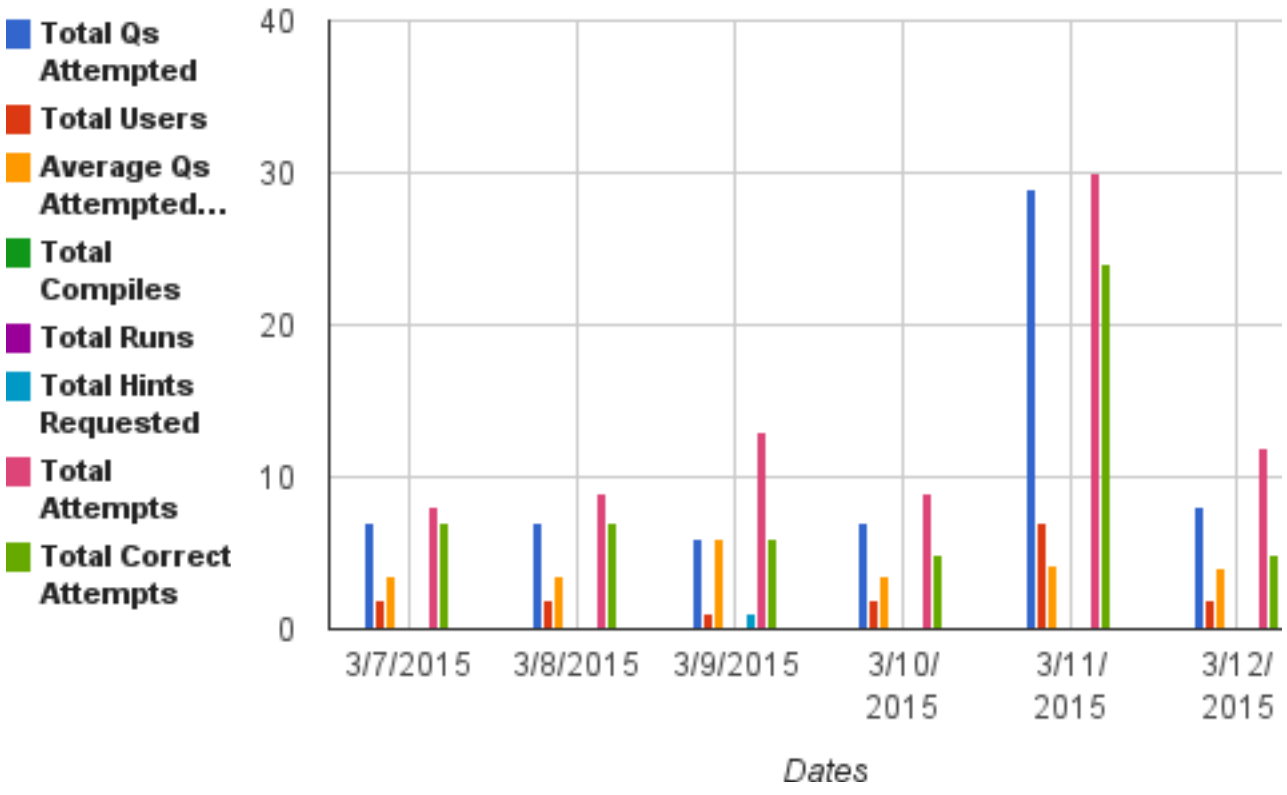

Figure A.8: If Statements Summary Data - Part B (see Table A.4 for corresponding data values)

If Statements Question Summary Data

Question number 2 had the greatest number of compiles at 295 (and 171 runs), while question 6 had the greatest number of runs (200, with 260 compiles). The most hints were requested for Question 2 (61). Question number 6 had the highest number of attempts at 453, of which 120 (26.5%) were correct. The greatest number of correct attempts was for Question 5, with 104 of 133 attempts correct (78.2%).

February Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	0	2	0
2	295	171	61	414	222
3	115	79	34	159	103
4	84	60	9	134	79
5	48	37	13	133	104
6	260	200	58	453	120
7	179	100	28	254	141
March Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	0	0	0
2	0	0	0	27	16
3	0	0	0	9	9
4	0	0	0	9	8
5	0	0	0	10	9
6	0	0	1	18	4
7	0	0	0	8	8

Table A.12: If Statements Question Summary Data

A.2.5 "Everything Combined: Writing Code"

User interaction for this topic peaked at two separate times: first, on February 15, the day of the first midterm review session and the day before Midterm 1, and second on March 11, the day before Midterm 2 was held, with slight increases in usage on March 9.

On February 15, a total of 215 question attempts by 55 users were made, an average of 3.91 questions attempted per user. On this day, students compiled their code 1579 times, ran it 985 times, and made a total of 222 requests for hints. This day saw 667 attempts to answer Writing Code questions, 339 of which (50.8%) were correct.

On March 11, a total of 43 question attempts by 13 users were made, an average of 3.31 questions attempted per user. On this day, students compiled their code 293 times, ran it 208 times, and made a total of 32 requests for hints. There were 17 attempts to answer Code Writing questions, 13 of which (76.5%) were correct.

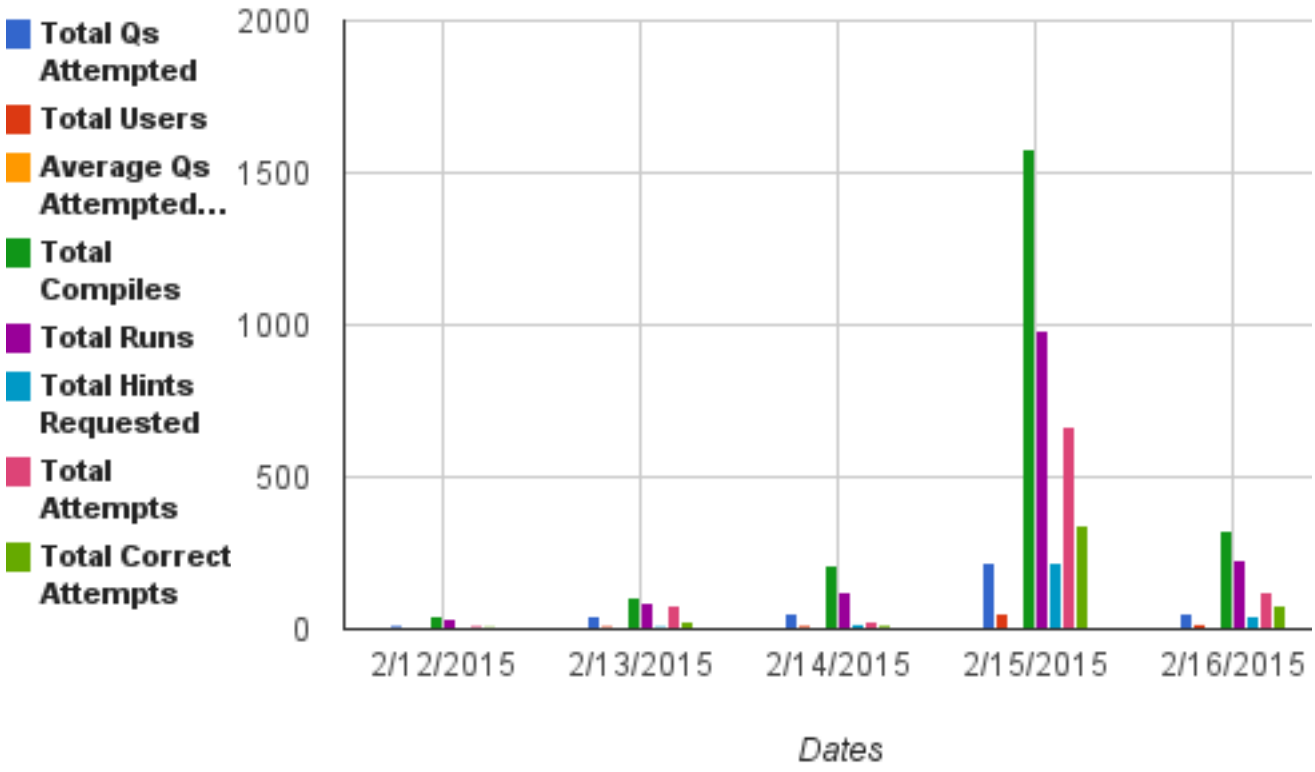

Figure A.9: "Everything Combined: Writing Code" Summary Data - Part A (see Table A.5 for corresponding data values)

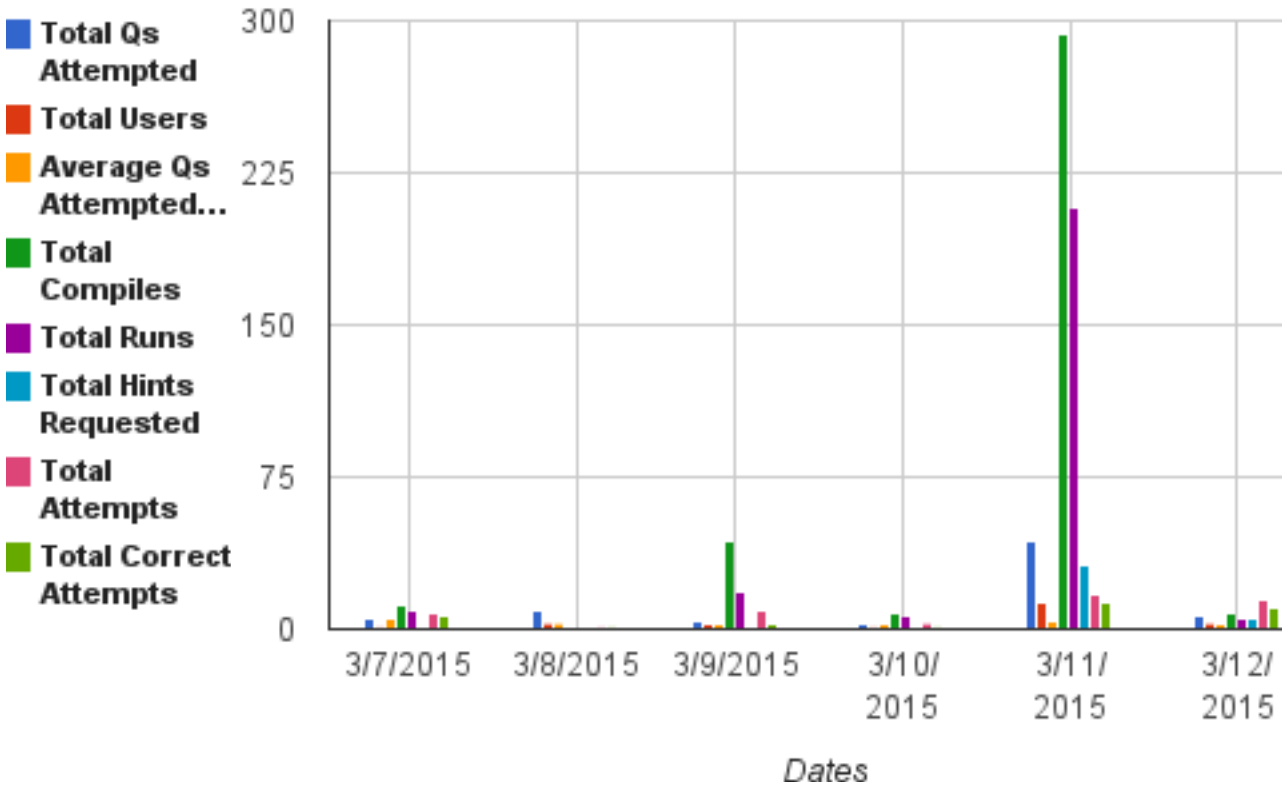

Figure A.10: "Everything Combined: Writing Code" Summary Data - Part B (see Table A.5 for corresponding data values)

"Everything Combined: Writing Code" Question Summary Data

Question number 2 had the greatest number of compiles and runs at 806 and 560, respectively. The most hints were requested for question 2 (136). Question number 2 also had the highest number of attempts at 415, of which 207 (49.9%) were correct. The greatest number of correct attempts was for question 6, with 66 of 77 attempts correct (85.7%). It is also worth noting that this question had the fewest attempts outside of question 1 in the February period. Question 2 also had a peak in usage in March, with 192 and 132 compiles and runs, respectively, and 10 out of 22 total attempts correct (45.4%).

February Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	27	18	1	2	1
2	806	560	136	415	207
3	521	385	46	193	90
4	347	223	39	129	47
5	303	117	49	94	52
6	271	159	23	77	66
March Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	0	0	0
2	192	132	13	22	10
3	88	71	10	7	4
4	45	27	3	14	12
5	23	7	3	5	3
6	17	10	8	5	5

Table A.13: "Everything Combined: Writing Code" Question Summary Data

A.2.6 "Everything Combined: Tracing"

User interaction for this topic peaked at two separate times: first, on February 15, the day of the first midterm review session, and the day before Midterm 1, and second on March 11, the day before Midterm 2 was held, with slight increases in usage on March 8 and 9.

On February 15, a total of 153 question attempts by 44 users were made, an average of 3.48 questions attempted per user. On this day, students compiled their code 839 times and ran it 428 times, and made a total of 207 requests for hints. This day saw 929 attempts to answer Tracing questions, 377 of which (41%) were correct.

On March 11, a total of 46 question attempts by 19 users were made, an average of 2.42 questions attempted per user. On this day, students compiled their code 67 times and ran it 29 times, and made a total of 23 requests for hints. There were 59 attempts to answer Tracing questions, 33 of which (55.9%) were correct.

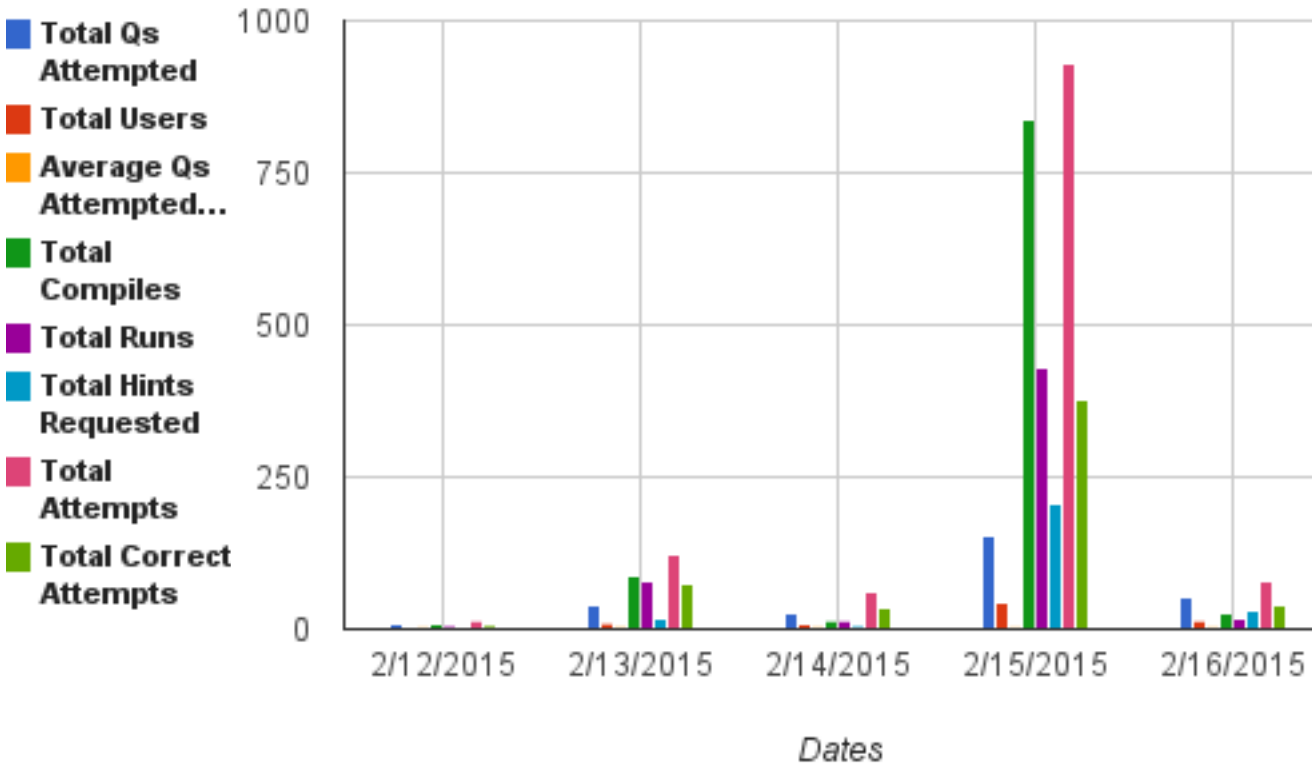

Figure A.11: "Everything Combined: Tracing" Summary Data - Part A (see Table A.6 for corresponding data values)

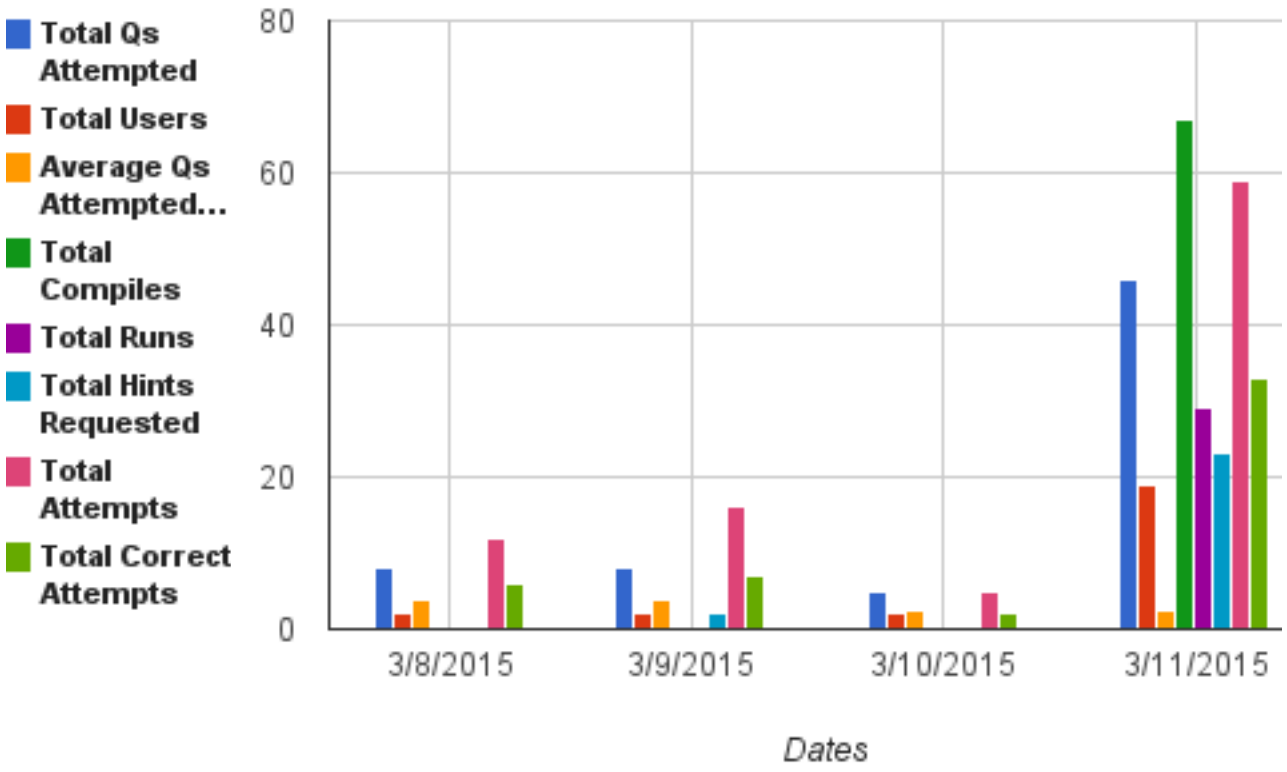

Figure A.12: "Everything Combined: Tracing" Summary Data - Part B (see Table A.6 for corresponding data values)

"Everything Combined: Tracing" Question Summary Data

Question number 2 had the greatest number of compiles and runs at 410 and 279 respectively. The most hints were requested for question 2 (127). Question number 2 also had the highest number of attempts at 435, of which 184 (42.2%) were correct. The greatest number of correct attempts during the February peak period was for Question 3, with 118 of 216 attempts correct (54.6%).

February Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	3	15	3
2	410	279	127	435	184
3	198	94	30	216	118
4	201	87	39	266	121
5	167	83	64	277	104
March Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	0	0	0	0	0
2	30	11	14	35	16
3	2	0	7	18	13
4	2	2	1	27	14
5	33	16	9	25	6

Table A.14: "Everything Combined: Tracing" Question Summary Data

A.2.7 I/O Tracing & Coding

Since this topic was added only for the second midterm in March, there is no datum for February.

User interaction for I/O Tracing & Coding peaked on March 11, the day before Midterm 2 was held, with increases in usage on March 8, the day of the second review session, March 10, and March 12, the day of the midterm. On March 11, a total of 225 question attempts by 49 users were made, an average of 4.59 questions attempted per user. On this day, students compiled their code 637 times and ran it 283 times, and made a total of 140 requests for hints, despite the fact that none were available. There were 312 attempts to answer I/O Tracing & Coding questions, 71 of which (22.8%) were correct.

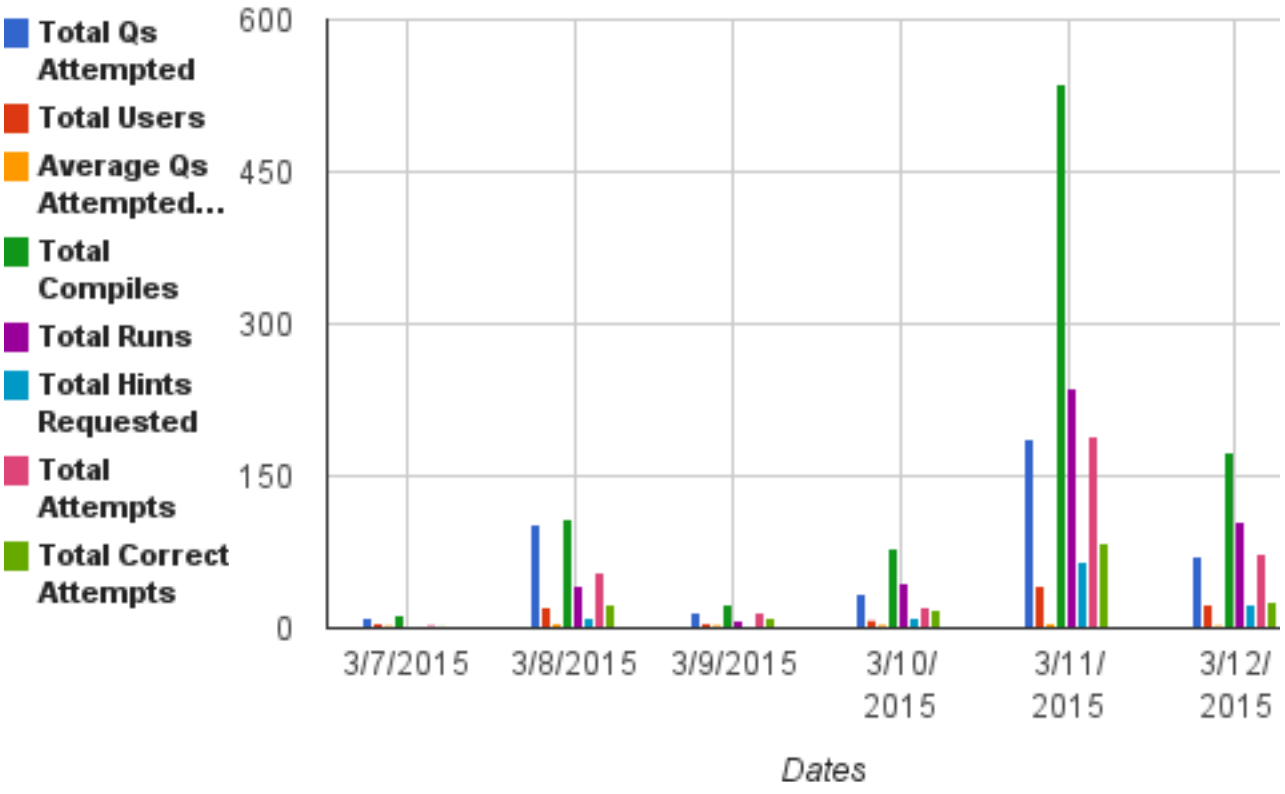

Figure A.13: I/O Tracing & Coding Summary Data (see Table A.7 for corresponding data values)

I/O Tracing & Coding Question Summary Data

Question number 2 had the greatest number of compiles and runs at 641 and 277 respectively. The most hints were requested for question 2 (83), despite the fact that none were available for questions in this topic. Question number 2 also had the highest number of attempts at 178, of which 60 (33.7%) were correct. The greatest number of correct attempts was for question 5, with 44 of 80 attempts correct (55%).

March Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	19	12	7	27	5
2	641	277	83	178	60
3	50	15	65	152	30
4	162	78	39	80	23
5	25	14	5	80	44
6	27	15	9	115	22
7	194	93	22	58	13

Table A.15: I/O Tracing & Coding Question Summary Data

A.2.8 Arrays

Since this topic was added only for the second midterm in March, there is no data for February.

User interaction for Arrays peaked on March 11, the day before Midterm 2 was held, with increases in usage on March 8, the day of the second review session, and March 12, the day of the midterm. On March 11, a total of 186 question attempts by 42 users were made, an average of 4.43 Arrays questions attempted per user. On this day, students compiled their code 538 times and ran it 237 times, and made a total of 67 requests for hints, despite the fact that none were available. There were 190 attempts to answer Arrays questions, 83 of which (43.7%) were correct.

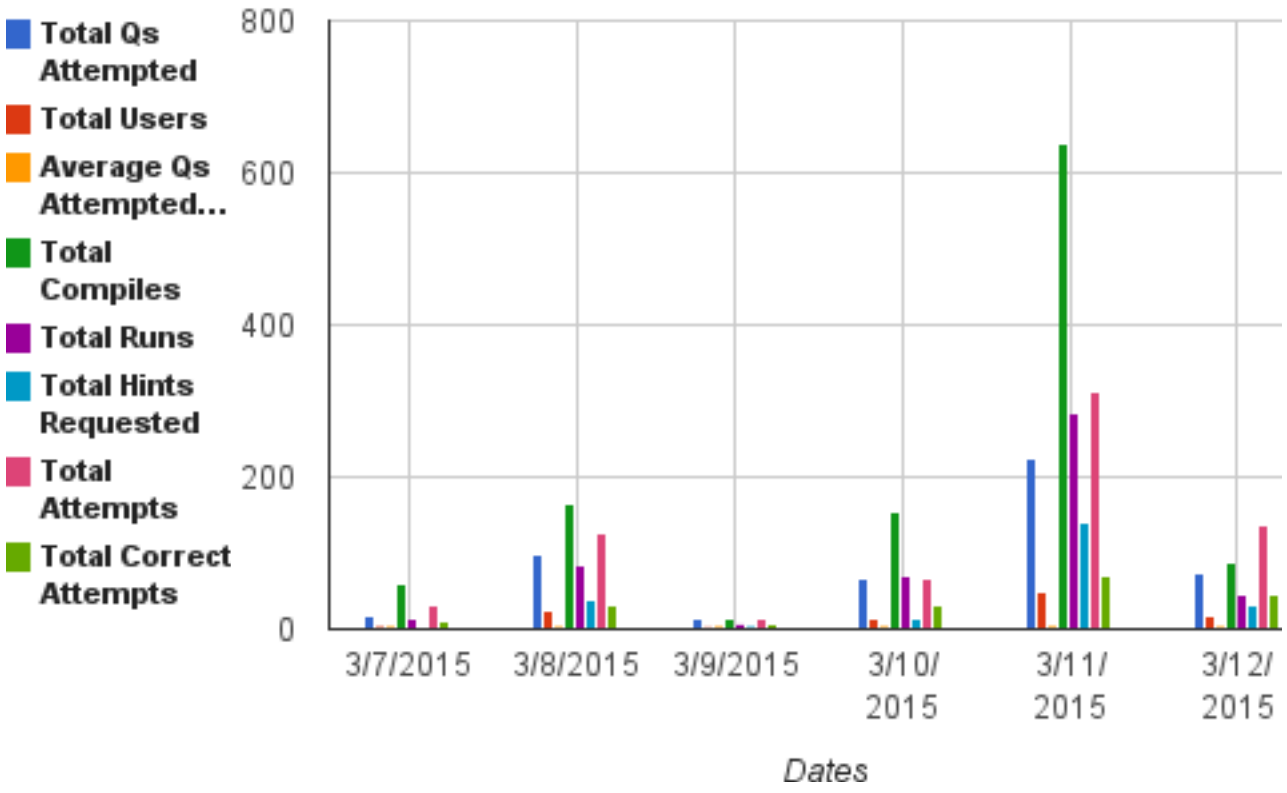

Figure A.14: Arrays Summary Data (see Table A.8 for corresponding data values)

Arrays Question Summary Data

Question number 3 had the greatest number of compiles and runs at 337 and 156 respectively. The most hints were requested for question 2 (45), despite the fact that no hints were actually available for questions in this topic. Question number 2 also had the highest number of attempts at 155, of which 70 (45.2%) were correct. The greatest number of correct attempts was for question 4, with 31 of 47 attempts correct (66%).

March Data					
Question #	Total Compiles	Total Runs	Total Hints Requested	Total Attempts	Total Correct Attempts
1	15	0	2	5	1
2	293	132	45	155	70
3	337	156	25	52	20
4	29	4	15	47	31
5	117	56	8	19	8
6	24	19	4	48	20
7	123	70	12	32	13

Table A.16: Arrays Question Summary Data

Bibliography

[1] Ace - the high performance code editor for the web. `http://ace.c9.io/`, 2015. Accessed: 2015-04-10.

[2] The Java Tutorials - Oracle Documentation. `https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html`, 2015. Accessed: 2015-04-13.

[3] UVic Course Calendar. `http://web.uvic.ca/calendar2015-01/CDs/CSC/110.html`, 2015. Accessed: 2015-04-10.

[4] Christine Alvarado, Zachary Dodds, and Ran Libeskind-Hadas. Increasing women's participation in computing at Harvey Mudd College. *ACM Inroads*, 3(4):55, 2012.

[5] Alexander W. Astin. Student involvement : A developmental theory for higher education. *Journal of College Student Development*, 40(July):518–529, 1999.

[6] Sylvia Beyer, Kristina Rynes, Julie Perrault, Kelly Hay, and Susan Haller. Gender differences in computer science students. *ACM SIGCSE Bulletin*, 35:49, 2003.

[7] Benjamin S. Bloom. *Taxonomy of educational objectives: the classification of educational goals. Handbook 1: Cognitive domain.* 1956.

[8] Kristy Elizabeth Boyer, Robert Phillips, Michael D. Wallis, Mladen a. Vouk, and James C. Lester. The impact of instructor initiative on student learning. *ACM SIGCSE Bulletin*, 41:14, 2009.

[9] Kevin Buffardi, Mcbryde Hall, Stephen H Edwards, and Mcbryde Hall. Adaptive and Social Mechanisms for Automated Improvement of eLearning Materials. pages 165–166, 2014.

[10] J. M. Cohoon. Toward improving female retention in the computer science major. *Communications of the ACM*, 44(5):108–114, 2001.

[11] StackOverflow.com Contributors. Figure: Node.js Event Loop. `http://stackoverflow.com/questions/21596172/what-function-gets-put-into-eventloop-in-nodejs-and-js`, 2015. Accessed: 2015-04-13.

[12] Jennifer Dempsey, Richard T Snodgrass, and Isabel Kishi. The Emerging Role of Self-Perception in Student Intentions Categories and Subject Descriptors. pages 108–113.

[13] Bernie Dodge, John Whitmer, and James P Frazee. Improving Undergraduate Student Achievement in Large Blended Courses Through Data-Driven Interventions. pages 412–413, 2015.

[14] Sh Edwards and Jason Snyder. Comparing effective and ineffective behaviors of student programmers. *Fifth international workshop on Computing education research workshop - ICER '09*, pages 3–14, 2009.

[15] Nickolas J.G. Falkner and Katrina E. Falkner. A fast measure for identifying at-risk students in computer science. *Proceedings of the ninth annual international conference on International computing education research - ICER '12*, page 55, 2012.

[16] Scott Frees. A Place for Node.js in the Computer Science Curriculum. *Journal of Computing Sciences in Colleges*, 30(3):84–91, 2015.

[17] Paul Gross and Kris Powers. Evaluating assessments of novice programming environments. *Proceedings of the 2005 international workshop on Computing education research - ICER '05*, pages 99–110, 2005.

[18] Daniel Malcolm Hoffman, Ming Lu, and Tim Pelton. A web-based generation and delivery system for active code reading. *Proceedings of the 42nd ACM technical symposium on Computer Science Education - SIGCSE '11*, page 483, 2011.

[19] Ryan Holmes. Why Canada Is Failing at Tech. `http://business.financialpost.com/entrepreneur/fp-startups/why-canada-is-failing-at-tech?__lsa=7df4-07e0`, 2013. Accessed: 2015-03-13.

[20] K. Huffman and M Vernoy. *Psychology in Action*. Wiley, 2003.

[21] Gregory Hume, Joel Michael, Allen Rovick, and Martha Evens. Hinting as a Tactic in One-on-One Tutoring. *Journal of the Learning Sciences*, 5(1):23–47, 1996.

[22] Matthew C Jadud. A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education*, 15(April 2015):25–40, 2005.

[23] David S. Janzen, John Clements, and Michael Hilton. An evaluation of interactive test-driven labs with WebIDE in CS0. *Proceedings - International Conference on Software Engineering*, pages 1090–1098, 2013.

[24] Kenneth R. Koedinger, Elizabeth A. McLaughlin, and John C. Stamper. Data-driven Learner Modeling to Understand and Improve Online Learning. *Ubiquity*, 2014(05):1–13, 2014.

[25] Theodora Koulouri, Stanislao Lauria, and Robert D. Macredie. Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *Trans. Comput. Educ.*, 14(4):26:1–26:28, 2014.

[26] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, 37:14, 2005.

[27] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y.B.D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multinational, multi-institutional study of assessment of programming skills of first-year CS students. 2001.

[28] J. McGrath Cohoon and Lih-Yuan Chen. Migrating Out of Computer Science. *Computing Research News*, 16(2), 2003.

[29] Christian Murphy, Gail Kaiser, Kristin Loveland, and Sahar Hasan. Retina: helping students and instructors based on observed programming activities. *ACM SIGCSE Bulletin*, pages 178–182, 2009.

[30] Cindy Norris, Frank Barry, and Jb Fenwick Jr. ClockIt: collecting quantitative data on how beginning software developers really work. *Acm Sigcse ...*, pages 37–41, 2008.

[31] Andrei Papancea, Jaime Spacco, and David Hovemeyer. An open platform for managing short programming exercises. *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13*, page 47, 2013.

[32] Seymour Papert and Idit Harel. *Constructionism*. Ablex Publishing Corporation, 1991.

[33] Nick Parlante. CodingBat. `http://codingbat.com`, 2015. Technical Report. Accessed: 2015-03-13.

[34] D. Penniman. Cra outline of cs overview. `http://www.cra.org/Activities/itdeans/penniman.pdf`, 2003. Technical Report, Computing Research Association. Accessed: 2015-04-05.

[35] Arti Ramesh, Dan Goldwasser, Bert Huang, Hal Daume, and Lise Getoor. Uncovering hidden engagement patterns for predicting learner performance in MOOCs. *Proceedings of the first ACM conference on Learning @ scale conference - L@S '14*, pages 157–158, 2014.

[36] Mitchel Resnick. Technologies for lifelong kindergarten. *Educational Technology Research and Development*, 46(4):43–55, 1998.

[37] Brendan Riordan and John Traxler. Supporting Computing Students at Rick Using Blended Technologies. In *4th Annual LTSN-ICS Conference, NUI Galway*, pages 1–6, 2003.

[38] Guido Röß ling, Ari Korhonen, Rainer Oechsle, J. Ángel Velázquez Iturbide, Mike Joy, A. Moreno, Atanas Radenski, Lauri Malmi, Andreas Kerren, Thomas L. Naps, and Rockford Ross. Enhancing learning management systems to better support computer science education. 2008.

[39] E Seymour and N Hewitt. *Talking About Leaving: Why Undergraduates Leave the Sciences*. Westview Press, Boulder, CO, 1997.

[40] Vincent Tinto and Vincent Tinto. Research and Practice of Student Retention: What Next? *Journal of College Student Retention*, 8(1):1–19, 2007.

[41] Ian Utting, Neil Brown, Michael Kölling, Davin McCall, and Philip Stevens. Web-scale data gathering with BlueJ. pages 1–4, 2012.

[42] Kurt Vanlehn. The Behavior of Tutoring Systems. *Int. J. Artif. Intell. Ed.*, 16:227–265, 2006.

[43] Kurt VanLehn. The interaction plateau: Answer-based tutoring ¡ step-based tutoring = natural tutoring. In *Intelligent tutoring systems*. Springer, 2008.

[44] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. Extreme apprenticeship method in teaching programming for beginners. *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11*, page 93, 2011.

[45] Naomi White. Tertiary education in the Noughties: the student perspective. *Higher Education Research and Development*, 25(April 2015):231–246, 2006.

[46] Stuart Zweben. Computing Degree and Enrollment Trends From the 2011-2012 CRA Taulbee Survey. Technical report, Computing Research Association, 2013.