Disparity Between Textbook Examples and What Young Students Find Interesting

Bowen Hui, Parsa Rajabi, Angie Pinchbeck
Department of Computer Science, Mathematics, Physics, and Statistics
University of British Columbia
Kelowna, Canada
bowen.hui@ubc.ca, {prajabi,angiepin}@student.ubc.ca

***Abstract*—To keep young students engaged in computer science, it is crucial to develop teaching material that they find interesting and relevant. Unfortunately, standard CS1 textbooks typically use examples that are uninspiring or inaccessible to young people. To better understand the disparity between textbook examples and student interests, we analyzed a collection of CS1 textbooks and compared the resulting topics to those elicited from young students via focus groups. We found 47% of textbook topics (out of 53 topics from 910 code examples) did not overlap with any topic mentioned by our participants. Conversely, among the topics elicited from the participants, we found 29% of these topics (out of 24 topics from 1936 items) missing from textbooks. To measure the overlap between these two data samples, we computed the Bhattacharyya coefficient and obtained 0.4452 indicating a strong difference between the two sets. These results lead us to advocate for changes in the teaching materials in order to make them more engaging for young students.**

***Index Terms*—Intrinsic motivation, interest-based learning, CS1 textbooks, Java, inclusion**

I. Introduction

Many initiatives have emerged to promote coding literacy in recent decades. Consequently, the university student population in computer science has diversified. Although our CS1 course is intended to be taken by majors, the class typically has students from other disciplines taking CS1 for credit and older undergraduate students considering computer science as an alternative career path. In our last academic session, approximately 20% of the CS1 class was composed of students majoring in other disciplines, such as engineering, mathematics, biology, management, and psychology.

Young students today have diverse skill sets and often pursue computer science for different reasons. To keep them engaged, it is important to ensure that CS1 content is interesting and relevant for students. Ideally, the educational resources should be designed in a way that interests the students so that we can sustain their engagement in the course and foster their intrinsic motivation for computer science.

Nowadays, it is easy to identify computing applications around us. For this reason, the importance of computer science has been recognized by many disciplines. However, the content in CS1 textbooks has largely remained unaffected by these interdisciplinary overlaps. In particular, many CS1 texts still use traditional topics to illustrate programming concepts (e.g., the area of a circle, investment calculations). Adopting these resources often adds to the belief that programming activities are "boring and demotivating" (p. 41) [1]. Moreover, examples that are hard to relate to can worsen the learning curve for students. For example, if an instructor talks about buying a house, it is unlikely that young students can relate to that experience. Thus, those students would suffer from an increased cognitive load by having to process an example in an unfamiliar domain in addition to trying to relate those ideas to understand core programming concepts.

To simplify the student's learning needs, it is important to develop material that students can relate to in a personally meaningful context. A recent study categorized 483 papers about CS1 courses published in SIGCSE over the past 50 years [2]. The authors identified 24 papers that presented model problems and examples and 12 papers that addressed gender, diversity, accessibility, and inclusion in CS1 courses. Several of these studies have also advocated for content renewal to meet inclusion objectives for women and visually impaired programmers [3], [4]. Although many interesting ideas and examples can be found in these papers, none of them emphasized the need to develop inclusive content tailored towards young students.

Our vision is to develop introductory material that speaks to young students as a way to promote intrinsic motivation. We believe that presenting computer science concepts in an accessible and relevant context can ameliorate the negative perception that programming is inherently uninteresting. Our goal here is to develop coding examples and assessments that are engaging and meaningful for post-millennials (those born in 1997 or later). As a first step, we wish to better understand what material is used in current CS1 textbooks and compare them with what young students find interesting. In this paper, we focus on the following research questions (RQs):

- RQ1: What topics are used in CS1 textbooks?
- RQ2: What topics are students interested in?
- RQ3: How much overlap is there between these two sets of topics?

Our approach is to empirically sample the content in CS1 textbooks and compare them to interesting topics elicited from young students. Section III describes the methodology used in this study and Section IV presents the results obtained from analyzing our data. As a key contribution, we provide

categories of interests gathered from our focus groups so that others may develop teaching material using those topics.

Our results showed that nearly half of the topics used in textbook programming examples were missing from the elicited student data. Based on 1936 items that young students reported as interesting to them, we derived 24 topics. We found that 7 of these topics (29%) were not used in any textbooks. Considering the number of overlapping items, we computed the Bhattacharyya coefficient [5] and obtained 0.4452, which indicates a strong difference between the two sets. The details of this analysis are provided in Section V. To illustrate how some of these young student topics can be used in teaching CS1 content, we offer examples of programming exercises. Ultimately, we wish to make use of these results in designing teaching material that is more interesting and relevant for young students and measure the impact of these resources on their motivation and academic performance.

II. Background

In this section, we review the concepts of intrinsic motivation and studies that promote interest-based learning through the development of interesting examples in CS1 education. We also identify CS education studies involving Java textbooks in order to compare their selection method to ours.

A. Intrinsic Motivation

We focus specifically on Self-Determination Theory where different dimensions of motivation explain how individuals carry out autonomous and intentional action [6], [7]. One such dimension is intrinsic motivation, where individuals take interest in activities simply because they enjoy them and not because they want to achieve the result associated with them. In this theory, when individuals are intrinsically motivated, they take autonomy in their own actions. Thus, individuals' behaviors are driven by their personal desires.

Many studies have investigated the relationship between intrinsic motivation and academic performance. While it is often believed that intrinsic motivation promotes better academic achievement, a systematic review of the literature reveals conflicting results [8]. The authors found that most of these studies involved cross-sectional designs and did not control for baseline student achievement. Through deeper investigations using three controlled longitudinal studies, these authors explored the relationship between different types of motivation and how they relate to each other over time. The main result from this work shows that intrinsic motivation is strongly and positively associated with academic performance.

In this context, Werner & Girnat surveyed 874 students to better understand what aspects of CS education motivate children in Grades 10 and 11 in Germany [9]. Rather than asking the children about their interest in different domains of computing that are largely unfamiliar to them, the authors constructed a questionnaire around three aspects of computing: theoretical, technical, and practical. The theoretical aspect had questions related to the algorithmic nature of computing, while the technical aspect had questions related to learning how technology works, and the practical aspect had questions about interest in application domains. The questionnaire had seven questions per category, with additional questions to elicit interest in studying or working in computer science. The main results from this study showed that the practical aspect of computing has the strongest association with intrinsic motivation. Furthermore, the study showed that the technical aspect is most closely linked to the behavior of pursuing a CS profession. We found the use of these three categories to be a promising approach for improving CS1 teaching material.

B. Studies Towards Interest-Based Learning

Studies dating back to the early 1980s have proposed the use of more interesting programming exercises in CS1 courses. An early paper argues for the need to choose interesting programming exercises to be used in classrooms so that students can be motivated to learn [10]. The paper presents four example exercises that introduce new technology ("new" for the paper) such as plotting a graph and using cassette tape and speech synthesizer. Since then, various ideas on interesting assignments and semester-long projects have been published (e.g., a maze exercise that teaches recursion [11], an ASCII version of Connect Four [12], a heart rate monitor to shock the heart as needed [13], a number game called Taxman that introduces concepts in artificial intelligence [14], a course-wide tournament game [15]). Due to the community's interest in this area, SIGCSE started gathering "Nifty Assignments" in 1999. Although empirical feedback from students was not provided for these assignments, their objectives of using interesting exercises to enhance student learning align with the goal of our work. Below, we highlight a few in-depth studies to gain a better perspective on interest-based learning approaches.

Researchers have argued for designing more interesting materials for target populations. In one paper, the authors described changes needed to protect the interests of women studying computer science [16]. One aspect they mentioned, which applies more generally, is to situate technology in the real world so to make computing more relevant. They emphasized the need to connect computer science to other disciplines and introduce diverse problems and teaching methods that appeal to diverse learning styles and preferences. Similarly, another study pushed for the same ideas so that non-majors can be motivated to engage in programming activities in the context of their own disciplines [17].

A study that implemented three practices in a CS1 course was able to demonstrate that these practices led to the successful increase of female enrollment [18]. One of these practices is a breadth-first approach that provides students with basic programming skills to write "interesting and useful programs within the first week of the course" (p. 2) [18]. Although details of what makes these programs interesting were not provided, the underlying idea overlaps with the central motivation of our work.

Beyond improvements for specific student populations, studies have also investigated changes in the teaching material and the impact on student engagement. One study reported on the

use of a student-authored textbook [19]. The objectives were to create more collaboration and writing activities in the course and reducing textbook costs. As part of this experience report, students were tasked with the creation of exercises for each chapter. A positive outcome observed by the author is that some concepts were better explained by students than teachers because students could elaborate on the examples better and relate better to the learning needs of their peers.

Related to this work are studies that attempt to increase student motivation in CS1 courses. Strategies include situating programming in a "fun" context, such as media computation [20] or using personal robots [21], and designing open-ended exercises to positively impact student motivation [22]. More examples can be founded in [22].

Work has been done on developing a new interest-based CS0 course to improve academic performance and retention in CS1/CS2 courses [23]. The researchers employed a team-based and project-based approach in CS0 to address several non-academic factors (e.g., making the course more collaborative and multidisciplinary, making the coding process more creative). A unique aspect of this course is that students can choose one of four project tracks: mobile app, robotics, gaming, and music. The authors recognized the need to have interesting projects as a source of motivation for the students. Among other factors, compared to those who enrolled directly into CS1, the authors found that students who took CS0 have better grades in CS1 (but not CS2) and are more likely to subsequently enroll in CS2.

Lastly, Cohoon investigated the impact of giving students computer access in all their classes and using motivating examples in the course material [24]. Two classes took part in the study: a research class that used both strategies and a control class that used neither. The results showed that the research class had a higher rate of attracting students into the major. To develop motivating examples, the research class surveyed students on a 7-point Likert scale to self-report how much they liked each item in a set of 46 computing applications. In contrast to our approach, we allow participants to tell us what they are interested in without restricting them to a predetermined list of applications.

Based on 330 responses (236 males, 94 females), Cohoon reported that females found the following applications to be interesting (an average score $\geq$ 5.0): card games, connect four, daily jumble (word puzzle), exercise training zone, instant messaging, language translation, medical diagnosis, music library, personality typing, photo manipulation, sudoku, tic-tac-toe; and interesting applications for males were card games, connect four, encryption, instant messaging, password security, tic-tac-toe. In alignment with Werner & Girnat's view of intrinsic motivation, this finding shows that females are more interested in practical aspects of computing, while males are interested in both practical and technical aspects.

Cohoon's study also found that less interesting applications are often mathematically-oriented. Specifically, female participants reported the following to be less interesting (an average score $\leq$ 3.5): econometrics, elevator strategies, redistricting, statistics, tax computation; and those for males were checkbook register and redistricting. Some applications received similar low scores for both genders (e.g., tax computation).

C. Studies Involving Java Textbooks

We found several studies that analyzed textbooks for CS education. Most authors reported their method of textbook selection was largely based on availability and personal choice. One study analyzed the treatment of specific Java topics in a suite of 16 textbooks [25]. The topics under analysis were objects early versus objects late, I/O, and software engineering concepts. The author mentioned that the selection of textbooks for inclusion was "somewhat arbitrary" (p. 2) [25]. In comparison to our textbooks, we found 2 in common (and ours made use of later editions).

Another study analyzed the coverage of CS1 concepts in 22 textbooks across several programming languages [26]. The authors identified major curriculum topics based on ACM/IEEE recommendations and further indicated concepts within those topics. Based on personal experience, the Congress of Library database searches, and colleague recommendations, the authors identified 22 texts that use 7 programming languages. Among those, 4 taught Java and were all published between 1999-2001. None of these books overlap with our selection.

A third study compared the coverage of programming concepts in C++ and Java textbooks [27]. The purpose was to determine which programming language had better textbooks suited for the CS1 curriculum. Based on their discretion, the authors selected 10 online textbooks for the two languages and compared their transformed word rate. In comparison to their Java textbooks, we have 3 overlaps with later editions.

We also found a recent study that evaluated the use of diagrams in Java textbooks [28]. In contrast to the studies described earlier, these authors selected 15 textbooks based on a combination of polled results from members of the CS education community and best seller programming textbook lists from Amazon and Barnes & Noble. In comparison to our textbooks, we found 6 in common (one that was an older edition, two were new editions).

III. Methodology

A. Textbook Analysis

To facilitate our search, we used an online syllabus tool to find textbooks used in CS1 courses [29]. In this tool, the researchers manually curated a set of CS1 syllabi by looking up university websites and conducting Google searches on 916 institutions taken from the QS World University Rankings 2016-2017. Their results included 234 syllabi from 207 institutions, 53% of which were from North America.

To manage the scope of our analysis, we limited our search to focus only on universities in North America that teach Java. The online database from [29] was accessed on December 02, 2019, and it returned 2 results from Canada and 42 from the USA (two of the latter were actually Canadian). Among these, some syllabi did not contain textbook information, some of them referenced a broken web link, and some of them referred

to custom textbooks that were not available online. As a result, we identified 12 Java textbooks used in CS1 courses [30]–[41].

For each text, we selected chapters covering topics in a typical CS1 curriculum. These topics include basic programming, selection, Math/Random/Character/String classes, loops, methods, single-dimensional arrays, multidimensional arrays, objects, and classes (prior to complicated object-oriented programming concepts such as inheritance and polymorphism).

Next, we identified the coding examples included in the selected chapters and categorized them based on the context of the intended application in those examples. Specifically, given a random chapter from each textbook (approximately 10% of the material), one coder went through each programming example and determined a domain topic label for it. Consider a basic program that calculates a person's body mass index. This program would be assigned the topic *health*. In this case, we decided to not assign the program to the *math* topic because the main purpose is not about the calculation. For simplicity, all examples belonged to only one topic. We did not use a predefined list of topics in this categorization task. Instead, we allowed topics to emerge.

Independently, a second coder categorized the same examples from the same chapters. Although the two coders did not coordinate the list of topics in advance, most topics identified were the same. In some cases, the labels used were different but they captured the same group of examples (e.g., using *banking* instead of *bank accounts*, or using *finance* instead of *payroll*). Comparing the resulting topics identified by the two coders, we found 85% agreement and resolved the remaining categories through discussions until consensus was reached. Thereafter, one of the two coders categorized the examples in the remaining chapters. The category labels from this analysis will henceforth be referred to as the textbook topics.

B. Student Focus Groups

To collect data about student interests, we recruited participants through volunteer sampling by targeting young participants from different academic backgrounds. We chose to conduct focus groups in order to explore their interests more deeply and ask clarification questions where needed. At the beginning of the session, we explained the purpose of the study is to re-design teaching material that would interest young students in computer science university courses. Participants were compensated with pizza at the end of the session. During the session, we posed a series of questions that probe their interests and asked each participant to individually write their responses on sticky notes. Sample questions include:

- What are your hobbies?
- What kind of activities do you do on the weekend?
- Which activities do your friends or relatives engage in that you want to learn more about?
- Do you collect any items?
- Where do you volunteer or where would you like to volunteer?
- What topics do you read about in your spare time?

Overall, we asked 27 open-ended questions about their general interests, hobbies, spare-time activities, activities with family and friends, work and volunteering experience, personal and do-it-yourself projects, club and team membership, favorite entertainment, collector items, places and cultures, global issues, and positive classroom experiences. After everyone was finished, focus group leaders helped participants identify categories and asked participants to post their own sticky notes under the categories or create new ones. The focus group format also lets participants see each other's responses and add to their own. During this time, the focus group leaders reviewed the content on the sticky notes and asked any clarification questions they may have. For example, when one participant wrote "water", the focus group leader was able to ask what the underlying interest was and discovered that the note referred to learning about water usage in communities.

In total, we conducted 6 one-hour focus group sessions with 26 participants (18 males/8 females). Among them, 69% are post-millennials and 31% are millennials, with 85% of them being in third-year undergraduate studies or higher. While the majority of the participants are computer science majors (41%), some are majoring in chemistry, creative writing, political science, earth and environmental sciences, engineering, psychology, and mathematics.

After the focus group sessions, we anonymized all the elicited items and collectively identified an initial set of categories that included games, television shows, sports, music, food, events, movies, social media, hobbies, and places of travel. Three coders began categorizing the data into the initial categories. During this process, we split up categories with many items into smaller subcategories if the larger category had many semantically related items in it. For example, we noticed that a good number of the items in the *music* category were actually about making music. Thus, we separated those items into a new category called *making music* and kept the rest under the general *music* category. On the other hand, there were many items in the *sports* category, but we did not split them up because we did not find any semantically meaningful subcategories that could be used. Lastly, we wanted to align the focus group topics to the textbook topics where possible. Based on this, we created new categories for *collection*, *blogs/websites*, *work/school*, *animals*, *friends*, and *language*.

Due to the large number of items we elicited, the three coders split up the data into three mutually exclusive sets so they could work in parallel. Afterward, they reviewed all the categories made by everyone and recorded what they agreed with. The result of this process was 92% agreement. The categories for the remaining items were resolved through discussion until consensus was reached. The category labels from this exercise will henceforth be referred to as the topics that emerged from the student focus group sessions.

IV. Results

A. Topics from CS1 Java Textbooks

Figure 1 summarizes the results of our textbook analysis. Our analysis discovered 910 programming examples from CS1

chapters across 12 textbooks. With each example belonging to one topic, we found 910 textbook topics which we grouped into 53 distinct topics. This histogram addresses RQ1 by presenting the common topics used in CS1 textbooks. Due to space limitations, topics with fewer than 10 counts (across all texts) are omitted from this histogram.

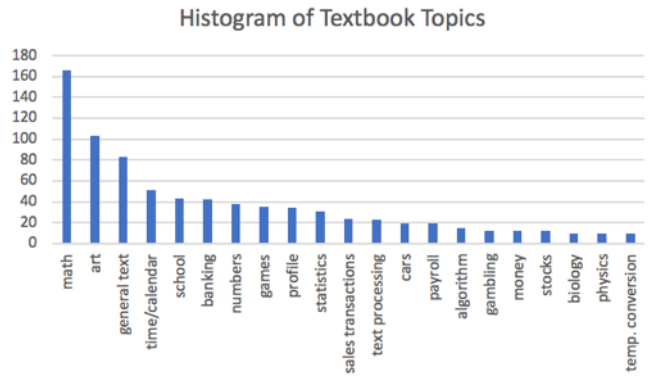

Fig. 1. A partial histogram of textbook topics found from our collection of 12 CS1 textbooks. Topics with fewer than 10 occurrences are omitted.

From Figure 1, we see an overwhelming number of examples belonging to the *math* topic. These examples include programs that focus on geometry, metric conversions, and calculations that have no other application context. The second most common topic is *art*, which includes examples such as ASCII art and graphical user interfaces (GUI) that covered basic GUI development, interface layout, event handling, and animation. Many of these examples came from special sections at the end of the chapter devoted to GUI development. Also, one textbook took an event approach so all of its examples are based on GUI development. However, many of them had to do with drawing shapes or developing a dialog popup that calculated investment earnings. In those cases, the examples were categorized as *math* and *banking* respectively.

We labeled several topics to denote examples that illustrate a programming construct without further application context. The topic *general text* refers to examples where the code would display very simple text messages. *Text processing* consists of examples with changes in a word or extrapolation of a letter within a word, typically using String or Character operations. *Numbers* involved simple changes to numeric variables. *File utility* referred to examples that demonstrated file input and output. *Algorithm* referred to examples that illustrated search or sort methods.

Common CS1 examples include time conversions and calendar calculations which we found in most texts and are collectively labeled as *time/calendar*. Surprisingly, there were also many examples of *banking* (e.g., investment calculations or credit card information). There were also some examples of *stocks*. We suspect these financial topics are not familiar to most young students.

We take a closer look at the number of examples from the topics *math*, *general text*, *text processing*, *numbers*, *file utility*, *algorithm*, and *time/calendar*. We found that these topics make up of 42% of all the textbook topics. This finding indicates that nearly half of the textbook examples focus purely on the mechanics of programming.

There were interesting topics such as *games*, *gambling*, and *money*, as well as familiar but dry topics such as *profile* (creating a profile of a person but with no other context), *sales transactions*, *payroll*, and *temperature conversion*. There were also a few academic topics like *statistics* (calculating the standard deviation and representing a coin or die but without further context), *biology*, and *physics*.

Topics with fewer than 10 occurrences that are not shown in Figure 1 include *food*, *health*, *house*, *music*, *toy*, *animals*, *colors*, *people*, *phone*, *literature*, *sports*, *language*, *astronomy*, *tv/movies*, *geography*, *history*, *voting*, *collection*, *encryption*, *fundraising*, *images*, *shirt*, *survey*, *travel*, *astrology*, *botany*, *game theory*, *Internet*, *law*, *weather*, and *wedding*. Although these topics were diverse and interesting, there were very few examples of them.

These textbook topics cover a mix of applications that students in Cohoon's study rated [24]. In comparison to the topics those students rated as highly interesting, the textbooks covered *art* and *games* but did not have much on *language*, *music*, *sports*, *social media*, or other specialized subjects. In comparison to the topics that those students rated as uninteresting, the textbook topics covered *math*, taxes (counted as *payroll*), *statistics*, and *sales transactions*.

B. Topics from Student Focus Groups

A total of 1936 items were elicited from our focus group sessions. As mentioned earlier, three coders were involved in processing the data from the focus groups. As a result, all the items were categorized into 24 topics. For example, when participants mentioned soccer, karate, running, camping, etc., all of these items were grouped into *sports*. Furthermore, we ensured everyone was in complete agreement and resolved disagreements before moving on. We selected one of the two coders from the text analysis activity as one of the three coders of this categorization task. Where possible, we tried to use the labels from the textbook analysis in this task as well. However, if certain topics had a large number of items, then that topic had a further split (e.g., *music* later became *music* and *making music*). The resulting categories for our focus group data are shown in Figure 2 as a histogram to address RQ2 by presenting the topics that young students find interesting. We leave the gender analysis of this data for future work.

We see from this data that the popular topics with 100 or more items are: *sports*, *places* (for travel), *events/issues*, *music*, *tv/movies*, *social media*, and *art creation*. Among these, two of them were very diverse. One is *events/issues* which covered a wide variety of social issues and concerns, including politics, climate change, indigenous issues, ethics in technology, religion, homelessness, mental health, war, protests, employment, safety, social welfare, famous figureheads, and recent events. The second diverse category is *other activities*, which covered a variety of pastimes and hobbies, such as volunteering, studying academic subjects, resting, cleaning,

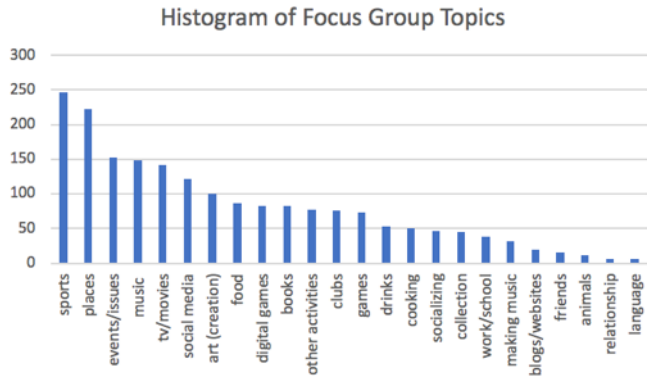

Fig. 2. The histogram of topics elicited from the focus groups.

gardening, and shopping. These categories reveal that our participants are active, interested in traveling, socially aware, and creative. There was also significant interest expressed for the following topics, which had between 50-100 items: *food*, *digital games*, *books*, *other activities*, *clubs*, (non-digital) *games*, *drinks*, and *cooking*.

In comparison to the aforementioned study where students self-reported how much they like a predetermined list of computer applications [24], we see these student topics cover all the items that were rated highly and did not include any item that was rated poorly (aside from one instance of math). Student interests as reported by [24] and this study are relatively consistent. Our study includes many more topics because we allowed the students to name their own interests rather than providing them with a predetermined list of items.

V. Discussion

Our main research question (RQ3) is to investigate the amount of overlap between the topics used in examples for teaching CS1 and the topics that young students find interesting. Given 53 textbook topics and 24 student topics, we first attempted to align them to investigate topic coverage.

At first glance, we were surprised to find that many topics were identical or nearly so: *sports*, *places* (matched to *geography* and *travel*), *music*, *tv/movies*, *art* (matched to *art* and *images*), *food*, *books* (same as *literature*), *clubs* (matched to *fundraising*), *games*, *collection* (matched to *collection*, *toy*, and *shirt*), *work/school* (treated as a superset of *school*), *blogs/websites* (same as *Internet*), *friends*, *animals*, and *language*. The student topic *events/issues* overlapped with the textbook topics of *health*, *voting*, *law*, and the student topic *other activities* overlapped with *math*, *gambling*, *biology*, *physics*, *botany*, and *history*.

As a result, we are left with the following student topics with no coverage from any textbook: *social media*, *digital games*, *drinks*, *cooking*, *socializing*, *making music*, and *relationships*. That gives us 7 out of 24, or 29% of the topics not covered by textbooks. Among the 1936 items elicited from focus groups, these 7 topics were responsible for 295 of them (about 15% of the student data).

Similarly, the following textbook topics were not related to any of the topics elicited by our participants: *general text*, *time/calendar*, *banking*, *numbers*, *profile*, *statistics*, *sales transaction*, *text processing*, *cars*, *payroll*, *algorithm*, *money*, *stocks*, *temperature conversion*, *file utility*, *house*, *colors*, *people*, *astronomy*, *encryption*, *survey*, *astrology*, *game theory*, *weather*, and *wedding*. That gives us 25 out of 53 topics, or 47% of the topics not identified by young students. Among the 910 textbook examples considered, these 26 topics were responsible for 457 of them (about 50% of the textbook data). This result suggests that students do not have an interest in half of the examples used in textbooks.

Although this analysis gives us a sense of topic coverage, a deeper analysis that investigates the percent overlap of these two distributions is warranted. Consider the topic of math as an example. As a textbook topic, math was the most popular topic and it appeared in 166 code examples. On the other hand, math was only mentioned once from the focus group data. This means that math covers 18% of the textbook data while it only represents 0.05% of the student data. Although the math topic appeared in both cases, a deeper analysis is needed to assess the overlap of the two data sets.

For this reason, we converted the two data sets into discrete distributions with common topic labels. To compute the overlap of the discrete distributions, we used the Bhattacharyya coefficient to measure the overlap between two statistical samples [5]. By definition, the value of this coefficient ranges in [0,1], with 0 being no overlap and 1 being complete overlap. Our calculation gives us 0.4452, which we interpret as a strong difference between the two sets. In other words, the topics used in textbooks are vastly different from the topics that students report as interesting.

A. Illustrative Examples

One might argue that simple and short programming exercises described in meaningful contexts are difficult to develop. This section offers examples of short programming exercises using student topics that had little to no coverage by the textbooks. In these examples, we provide the instructions and highlight the programming constructs involved. We hope that these examples can serve as motivation for others to develop coding exercises with relevant contexts.

1) Sports/Health (Time Conversion): This example enables students to practice arithmetic calculations in the context of a health application. It is designed to engage students from both a practical and technical perspective (which covers two of the three types of intrinsic motivation in computing [9]). This example reframes a traditional time calculation problem in the context of a trendy sports watch. Time calculation is usually done in the early part of a CS1 course.

Instructions: You are programming a Fitbit watch that measures the time a person spends doing sports activities. The Fitbit clocks the number of seconds engaged in an activity, and now you need to convert it into a format that is easy to read in terms of hours, minutes, and seconds.

- Determine the calculation for converting a specific number of seconds into the number of hours in those seconds.
- Next, consider the remaining seconds left, determine the calculation for converting them into the number of minutes in those seconds.
- Then, determine the number of seconds remaining.
- Display the converted time in an easy-to-read format.

2) Social Network (Friendship Analysis): This example enables students to practice arithmetic, conditionals, methods, and 2D arrays in the domain of social network analysis. It is designed to engage students from both a practical and technical perspective. This problem abstracts away the details in a social network and is presented as a simple 2D array exercise. 2D arrays are typically taught near the end of a CS1 course.

Instructions: You have a group of friends named Ann, Bob, Cam, Don, and Eva. However, they are not all friends with each other! Ann is a friend of Bob and Cam. Bob is friends with Ann, Cam, and Eva. Cam is friends with everyone. Don is friends with Cam and Eva. Lastly, we assume everyone is friends with themselves. Create an $N \times N$ table to represent their friendship (where N is the number of individuals involved). Specifically:

- In this table, write 1 in the cell of a given row and column if the names of that row and column are friends, and write 0 otherwise.
- Assume that friendship is mutual (so if X is a friend of Y then Y is also a friend of X).
- Write a method that takes a name, then traverses the corresponding column in the table, and counts how many friends that person has.

B. Threats to Validity

Due to a limited budget, we used versions of the textbooks that are at times older than the exact editions of the books used in the courses cited in the course syllabi gathered. Furthermore, we chose to focus on the core chapter content and excluded chapter exercises from our textbook analysis. Note that one text provides end-of-chapter exercises titled *Making a Difference* and claims to use "problems that really matter to individuals, communities, countries, and the world" (p. 200) [33]. Perhaps a more fair analysis of textbook topic coverage should include chapter exercises as well.

Coding reliability is inherently difficult. We came across a few issues specific to the textbook analysis process. Some texts do not provide code examples in an obvious format for counting purposes. As an example with clear formatting, [39] always displays example programs as a "listing". On the other hand, [31] (as well as several other texts) typically uses a figure to show code samples, but figures are also used for other illustrative purposes (such as tables and pictures). Moreover, code in figures from [31] are sometimes complete Java programs and sometimes partial code. In these cases, we counted all examples that show Java programs listed as figures.

A related problem is that some texts would present a Java program and then extract code snippets from it and explain each piece separately (e.g., [34]). Those cases only occurred in texts that did not clearly enumerate code examples. Here, we only counted complete Java programs. This approach yield fewer counts for texts that often showed partial code snippets. Another issue is that some examples span across two Java classes — one for the program logic and another for running the program. This happened often in the later chapter(s) on classes. In these cases, we counted them as two occurrences of the same topic.

The method we chose to compare intercoder reliability was calculated using the percent agreement of the coders. The main drawback of this method is that it does not account for chance agreement due to the coders simply guessing the same category. A better measure of intercoder reliability makes use of Cohen's kappa [42]. Unfortunately, since our coding process allowed categories to emerge, we could not compute this measure because it would require the coders to have a predetermined list of categories. Now that we have all the categories, a possible remedy is to conduct a posthoc coding analysis with all the textbook and focus group data and determine intercoder reliability using Cohen's kappa.

We note that the sample size in our focus groups is small and not representative of a general student population. We chose a focus group format because we wanted the opportunity to dive more deeply into topics and discussions as needed. We also note that these students come from a North American university and the topics that interest them may only represent the North American culture. As future work for gathering input from a larger sample and a broader student population, alternative data collection techniques such as surveys would be more suitable.

Lastly, while our results report on the disparity between the contexts used in textbook examples and the interests of young students, we note that the value of our findings rests with a future evaluation that assesses the impact of teaching material designed using interesting topics from our findings.

VI. Conclusions and Future Work

We found that textbook examples often focus on teaching the mechanics of programming constructs and are described in contexts that young students do not report as interesting. Specifically, we analyzed 910 programming examples from 12 Java textbooks and elicited 1936 interesting items from student participants. Among the textbook examples, about 42% of them belonged to topics that focused purely on programming mechanics without additional application context or motivation. Moreover, 25 out of the 53 textbook topics we discovered from the textbooks, or 47% of the textbook topics, did not overlap with the topics our student participants found interesting. From the student topics elicited, our analysis shows that 7 of the 24 topics, or nearly one-third of the topics, were not covered by any textbook examples.

In light of these results, we illustrated examples showing that even small programs can be reworded and embedded in application contexts that young students find interesting. Our data reveals that young students from our participant pool are active, interested in traveling, socially aware, and

creative. With this in mind, we recommend dedicating course design efforts towards developing motivating examples in CS1 courses because these courses play a crucial role in the recruitment and retention of the computer science program.

Although our present study focuses on teaching Java in CS1 courses, the results of interesting topics for young students are generalizable to other courses as well. We hope that by sharing our results that other educators may develop teaching material to engage young students in their courses as well.

Our immediate next steps include surveying a broader student population that is culturally diverse in order to obtain more representative data about student preferences. With more data, we can conduct subpopulation analyses that offer insights into the learning preferences of specific demographics as well (e.g., interests by gender or by visible minority groups). Our goal is to develop a repository of interesting programming exercises to share with other CS1 educators. Ultimately, we wish to assess the impact of these materials on student motivation and retention.

REFERENCES

[1] S. Furber, *Shut down or restart? The way forward for computing in UK schools*. London, UK: The Royal Society, 2012.

[2] B. Becker and K. Quille, "50 years of cs1 at sigcse: A review of the evolution of introductory programming education research," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 338—344, 2019.

[3] L. Rich, H. Perry, and M. Guzdial, "A cs1 course designed to address interests of women," in *Proceedings of the 35th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 190—194, 2004.

[4] R. Cohen, A. Fairley, D. Gerry, and G. Lima, "Accessibility in introductory computer science," in *Proceedings of the 36th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 17—21, 2005.

[5] A. Bhattacharyya, "On a measure of divergence between two statistical populations defined by their probability distributions," *Bulletin of the Calcutta Mathematical Society*, vol. 35, pp. 99–109, 1943.

[6] E. Deci and R. Ryan, *Intrinsic motivation and self-determination in human behavior*. New York: Plenum, 1985.

[7] E. Deci and R. Ryan, "The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior," *Psychological Inquiry*, vol. 11, no. 4, pp. 227—268, 2000.

[8] G. Taylor, T. Jungert, G. Mageau, K. Schattke, H. Dedic, S. Rosenfield, and R. Koestner, "A self-determination theory approach to predicting school achievement over time: the unique role of intrinsic motivation," *Contemporary Educational Psychology*, vol. 39, no. 4, pp. 342—-358, 2014.

[9] C. Werner and B. Girnat, "Towards self-motivated learning in computer science education," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pp. 26–32, 2020.

[10] A. Tharp, "Getting more oomph from programming exercises," in *Proceedings of the 12th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 91—95, 1981.

[11] I. Liss and T. McMillan, "An amazing exercise in recursion for cs1 and cs2," in *Proceedings of the 19th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 270—274, 1988.

[12] I. Liss and T. McMillan, "An example illustrating modularity, abstraction and information hiding using turbo pascal 4.0," in *Proceedings of the 20th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 93—97, 1989.

[13] R. Pattis, "A philosophy and example of cs-1 programming projects," in *Proceedings of the 21st ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 34—39, 1990.

[14] L. A. Carmony and R. L. Holliday, "An example from artificial intelligence for cs1," in *Proceedings of the 24th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 1—5, 1993.

[15] R. Pargas, J. Lundy, and J. Underwood, "Tournament play in cs1," in *Proceedings of the 28th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 214—218, 1997.

[16] A. Fisher and J. Margolis, "Unlocking the clubhouse: The carnegie mellon experience," *SIGCSE Bulletin*, vol. 34, no. 2, pp. 79—83, 2002.

[17] A. Forte and M. Guzdial, "Motivation and nonmajors in computer science: Identifying discrete audiences for introductory courses," *IEEE Transactions on Education*, vol. 48, no. 2, pp. 248—253, 2005.

[18] C. Alvarado and Z. Dodds, "Women in cs: An evaluation of three promising practices," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, (New York, NY, USA), pp. 57––61, 2010.

[19] C. Bennett, "Student-authored wiki textbook in cs1," *Journal of Computing Sciences in Colleges*, vol. 24, no. 6, pp. 50—56, 2009.

[20] L. Porter and B. Simon, "Retaining nearly one-third more majors with a trio of instructional best practices in cs1," in *Proceedings of the 44th ACM Technical Symposium on Computer science Education (SIGCSE)*, pp. 165—170, 2013.

[21] M. McGill, "Learning to program with personal robots: Influences on student motivation," *ACM Transactions on Computing Education*, vol. 12, no. 1, 2012.

[22] S. Sharmin, D. Zingaro, and C. Brett, "Weekly open-ended exercises and student motivation in cs1," in *Proceedings of the 20th Koli Calling International Conference on Computing Education Research*, pp. 1–10, 2020.

[23] M. Haungs, C. Clark, J. Clements, and D. Janzen, "Improving first-year success and retention through interest-based cs0 courses," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, (New York, NY, USA), pp. 589—594, 2012.

[24] J. Cohoon, "An introductory course format for promoting diversity and retention," in *Proceedings of the 38th ACM Technical Symposium on Computer Science Education (SIGCSE)*, pp. 395—399, 2007.

[25] B. Becker, "Pedagogies for teaching cs1 with java," tech. rep., University of Waterloo, 2001. Retrieved Jan 15, 2020 at https://cs.uwaterloo.ca/ bwbecker/papers/javaTextbooks/.

[26] J. McConnell and D. Burhans, "The evolution of cs1 textbooks," in *Proceedings of the IEEE Frontiers in Education Conference (FIE)*, 2002.

[27] K. McMaster, B. Rague, S. Sambasivam, and S. Wolthuis, "Coverage of cs1 programming concepts in c++ and java textbooks," in *Proceedings of the IEEE Frontiers in Education Conference (FIE)*, pp. 1—8, 2016.

[28] S. Mazumder, C. Latulipe, and M. Perez-Quinones, "Are variable, array and object diagrams in java textbooks explanative?," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pp. 425–431, 2020.

[29] B. Becker and T. Fitzpatrick, "What do cs1 syllabi reveal about our expectations of introductory programming students?," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, (New York, NY, USA), pp. 1011—1017, 2019.

[30] R. Bravaco and S. Simonson, *Java programming from the ground up*. McGraw Hill India, 2012.

[31] K. Bruce, A. Danyluk, , and T. Murtagh, *Java: An eventful approach*. Upper Saddle River, N.J: Pearson, 2005.

[32] J. Dean and R. Dean, *Introduction to programming with Java: A problem solving approach*. Boston: McGraw-Hill Science/Engineering/Math, 2007.

[33] P. J. Deitel and H. Deitel, *Java how to program, early objects*. New York: Pearson College Div., 11 ed., 2017.

[34] A. Downey and C. Mayfield, *Think Java: How to think Like a computer scientist*. Sebastopol, CA: O'Reilly Media, 2016.

[35] D. Eck, *Introduction to programming using JAVA*. San Francisco, Calif:Createspace Independent Pub., 2019.

[36] T. Gaddis, *Starting out with Java: From control structures through objects*. Boston: Pearson, 6 ed., 2015.

[37] C. Horstmann, *Big Java*. Hoboken, N.J: Wiley, 2 ed., 2005.

[38] J. Lewis and W. Loftus, *Java software solutions: Foundations of program design*. Boston: Pearson, 7 ed., 2011.

[39] Y. Liang, *Introduction to Java programming, comprehensive version*. Boston: Pearson, 10 ed., 2013.

[40] W. Savitch and K. Mock, *Absolute Java*. Boston: Pearson, 5 ed., 2012.

[41] R. Sedgewick and K. Wayne, *Computer science: An interdisciplinary approach*. Boston, MA: Addison-Wesley Professional, 2016.

[42] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, pp. 37—46, 1960.