# Developing and Classifying Benchmarking Questions for CS1

Temelkovski, Jonathon
University of Toronto

Shaikh, Maheen
University of Toronto

## ABSTRACT

Developing and validating benchmarking questions traditionally requires the difficult task of seeding question into examinations. We present a simpler method of developing potential benchmarking questions. Distributing an assessment of 12 questions, we analyzed and validated student responses across major themes in both CS1 and CS2 to determine the qualities a benchmarking question should have to be effective.

## CCS CONCEPTS

• **Social and professional topics** → *Computer science education*;

**ACM Reference Format:**
Temelkovski, Jonathon and Shaikh, Maheen. 2019. Developing and Classifying Benchmarking Questions for CS1. In *Proceedings of Computer Science Education*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.475/123_4

## 1 INTRODUCTION

Computing education is in its infancy stage of comprising a list of benchmarking questions. Because of the various languages involved in this discipline, standardizing questions across languages is very difficult, if not, impossible to do. Regardless, a call to more standardized assessments [13] set of benchmarking questions has been made.

With different pedagogy found across institutions and the influence instructors have over course content, a standard is required to ensure consistency across student CS1 knowledge [1, 2]. Defining a standard requires an instrument that can compare populations to generate data that can be argued. One potential instrument is benchmarking: gathering data from students responses to a validated set of questions and comparing it to established standard to determine how students perform relative to their peers.

Benchmarking questions serve as an instrument for gathering data on a range of students cognitive abilities. They further serve as a reliable source to analyze content material in terms of difficulty and how educators can combat misunderstandings. Establishing a benchmark requires the time consuming and difficult task of seeding questions into CS1 exams (see [3, 4, 5]). Identifying and validating benchmarking questions without the use of seeding, but instead relying on an objective analysis and validation process, would allow future researchers to explore a less time-consuming methodology to validate potential benchmarking questions. Furthermore, the proposed research provides validated questions that can be used to establish a future benchmark.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
*Computer Science Education, April, 2019*
© 2019 Copyright held by the owner/author(s).
ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

This paper will address major themes taught in first year Computer Science at the University of Toronto. Computer Science courses are split into two semesters in which students in CS2 need CS1 as a prerequisite. The 12 benchmarking questions proposed in this paper will be such that CS1 students will be able to solve. Major themes looked at include arrays, loops, objects, memory, variables, tracing code, conditionals and docstrings. The goal of the proposed research paper will be to define an objective process and use it to evaluate our research question: "To what extent can an objective process identify and validate potential benchmarking questions for CS1?"

## 2 RELATED WORK

### 2.1 Problem Interaction and Misconceptions

Analyzing students interacting with CS1 problems often exposes underlying misconceptions that manifest in aspiring computer scientists. Understanding the interaction between a student and a question can provide evidence when arguing the validity of a benchmarking question.

Tracing code, an important skill for a computer scientist to acquire, is a major area of difficulty for students in CS1 [11]. Students that use a line by line analysis approach, often rewarded partial credit for, are known to be at a disadvantage in their future courses when the code becomes increasingly complex [21]. Moreover, students that lack the ability to use abstraction often produce weaker code [15].

The misconception that the computer "knows what to do" is a consequence of many students using a mental model approach to evaluating programs [21]. R.D. Pea coined this misconception as the "superbug" [22]; the idea that there exists an all-powerful mind within the computer. While many students know this to be untrue, they lack the ability to determine what the computer can't do.

### 2.2 Important and Difficult CS1 Topics

A sound benchmark must address both the important and difficult topics in CS1 and provide good coverage of core topics. With the dispute over what topics in CS1 are inherently difficult, task of evaluating difficulty is redirected onto the researchers building the benchmark.

Goldman et al. failed to gain a strong consensus on what CS1 topics are difficult [18] and gained a fair consensus on important CS1 topics through multiple consultations with an expert panel. The vast amount of different programming languages was noted as a factor of the disagreement, with abstract concepts (e.g. memory model or recursion) being noted difficult, nonetheless. Attempting to categorize CS1 again and using the results from three Delphi processes, they compiled a more detailed list of important CS1 topics [24]. Discord among the expert panels produced skeptical results when attempting to categorize difficult topics.

Returns versus prints are a notably difficult distinction for CS1 students. Krushkova, Stoynova  Krushkov [] surveyed students to find the most difficult topics covered in computer science. They find

that the three biggest areas of concern are “basic concepts, subroutines, and arrays". Subroutines and procedures were once the now return and limited work has been done on these to understand the exact reason of what makes this a difficult concept.

## 2.3 Analyzing Question Difficulty

Understanding why a question is difficult and how to mitigate non-concept difficulties from a question is required to generate valid benchmarking questions.

Petersen et al. determined the average CS1 question is composed of five to seven conceptions, and difficulty in one concept may result in poor overall performance on the question [9]. The number of discrete pieces of information needed to understand and answer the question, the task size, and conceptual level of topics covered were noted as influential factors on question difficulty [16]. The researchers asserted that creating questions that test atomic concepts are possible [17], paving the way for strong benchmarking questions.

An academic’s hypothesis on question difficulty positively correlates with student performance [16]. Simon et al. assert that computing academics have a fairly good idea of the difficulty of CS1 questions. Student difficulties arise when questions are poorly worded or students lack the understanding of how to answer the question [12]. The linguistic complexity of a question, coupled with external domain references, have been shown to negatively impact student performance [23]. Sheard et al. [19] synthesized that a question difficulty can be determined using five factors: external domain references, explicitness, linguistic complexity, conceptual complexity and code length.

## 2.4 Validating Instruments

An instruments used to gather data for benchmarking is difficult to validate. The inconsistencies found across courses and institutions makes arguing the validity of in course assessments increasingly difficult. Researchers developing benchmarking questions must carefully assess the psychological influence question formatting has on student performance.

Sheard et al.’s analysis of 11 introductory programming exams [13] exposed great variations in the number of aspects found across examinations. Of the 20 exam papers examined, fewer than 23% of questions reviewed were of medium difficulty, with 7 exams containing no ’difficult’ questions at all [10]. The inconsistencies found across examinations produce a lack of confidence in declaring exams as valid instruments.

At four institutions, Lister et al. administered trial assessments and discovered that students tend to guess on MCQ’s (approximately 12-30%) [8]. Moreover, an assessment of twelve MCQ’s took students on average 30-60 minutes to complete. Free-text questions, on the other hand, tend to have significantly worse student performance, due to students fixating on the code and missing the bigger picture [14]. An increase in student scores was recognized when they were presented with more detailed code.

The motivation of the student can impact their performance on a question. Intrinsic motivation positively impacts student performance, while extrinsic motivation seems to have little to no effect [20]. Ensuring that the students have intrinsic motivation when completing standardized assessments may produce more accurate results on a student’s cognitive ability, but this claim has not been explicitly verified.

## 2.5 Standardized Tests

Benchmarking questions serve as a valid instrument for gathering data on a student’s cognitive ability. Two fundamental CS1 assessments have shown that it is possible to correlate a student’s ability with assessments scores, but the task of doing so accurately is complex.

Tew et al. [6] developed an assessment to determine the extent that a language independent assessment can evaluate conceptual CS1 knowledge. Empirical studies and think-aloud interviews were used to build a set of validated pseudocode questions. An assessment composed of the questions, commonly known as the FSC1, was administered to 952 students across four institutions. A strong correlation between a student’s FSC1 scores and their final exam scores indicates the accurate measure of a student’s conceptual CS1 knowledge.

The success of the FSC1 inspired a replication and validation of such an assessment. Parker et al. developed the SCS1 [7] to determine to what extent the FSC1 was applicable at an international level. The SCS1 supported the findings proposed by the FSC1 but exposed the fragility of validated assessments. Adding new questions, additional question choices, or changing the order of questions invalidates the assessment. Questions with uninformative wording or that contained typos negatively impacts student performance.

Assessments traditionally use a common CS1 programming language, but the use of pseudocode provides relatively new insights to student performance. Bayman et al. [25] determined that common CS1 pedagogy informs students on “how to Java" rather than how to program, arguing a clear separation between the two is needed. A language composed of randomly generated syntax (called “Randomo") determined that the unintuitive syntax of many languages, especially ones such as try/catch, make programming more difficult for students [26].

## 2.6 Benchmarking

There is no widely accepted benchmark among the computer science education research community, but progress is being made in defining one.

Simon et al. [3, 5] explored establishing a CS1 benchmark through seeding questions into examinations in multiple countries. Comparing student scores to the average scores across all institutions (the benchmark), Simon et al. determined the relative performance of the student to their peers and under different pedagogy [3]. Moreover, the difficulty and size of an exam (by determining what percentage of the exam consisted of the benchmarking questions) was also measured using the benchmark [5].

The process used to determine benchmarking questions in [3, 4, 5] requires seeding questions into exams. Some instructors made minor adjustments to the questions to better integrate them into their courses. Altering the benchmarking questions invalidated them and produced skewed results that limited the number of questions available to establish the benchmark. For example, a change of indentation at one institution caused a question to be discarded from the benchmark in [3].

# 3 METHODOLOGY

A list of over thirty questions was initially compiled from past CS1 exams and weekly online homework submissions. Researchers examined each question under the following criterion:

- Questions should cover one or two topic areas [2]
- Questions do not rely heavily on non-CS concepts (e.g. cultural knowledge)
- Questions present clear, informative wording
- Question do not try to trick or catch the student (i.e. they are fair)
- Questions are not too complicated nor too simple (i.e. they are appropriate for an exam setting)

Questions that did not satisfy all five criteria or posed dispute amongst researchers were omitted. Questions that satisfy some criteria but not all were edited and sent back for review. Responses to questions from the online repository were analyzed and questions that suffered validity were omitted. Responses to past exam questions were not available to the researchers.

Careful analysis trimmed the list of questions down to twelve. Three of the remaining questions were proposed in various formats: MCQ, Parsons, or fill-in-the-blanks. The MCQ versions were used due to formatting restrictions of online medium. Consistency of wording, presentation and formatting was ensured across the final twelve questions. The benchmarking questions were presented in the Python, the programming used in CS1 and CS2 at the institution solicited. The questions can be found in Appendix I.

# 4 DATA

## 4.1 Collection

The assessment composed of the twelve benchmarking questions was distributed online at a large public research-intensive university in Canada. Students in CS1 and CS2 were solicited through their course forums during the later half of the winter term. Students in the computer science major would be enrolled in CS2 courses during this time, with the CS1 classes primarily filled with non-major or second-attempt students. The assessment is not affiliated with the courses advertised in and amount to no marks in the studentâĂŹs final grade.

We collected a total of fifty-four responses; ten from CS1 and forty-four from CS2. Students provided their consent to participate and were required to answer all twelve questions. An option to enter a lottery draw to win one of five $10 amazon gift cards for participating in our research was presented to the students upon completion of the assessment.

## 4.2 Themes

In this section we present and discuss certain themes in the benchmarking questions used in this study. Python was used to clean, organize, anonymize and report statistics (such as distractor usage) of the 54 responses. Script data was also used to construct graphs based on thematic characteristics.

Questions were analyzed individually and student responses were categorized into major CS1 themes. Synthesis across each theme was used to explore relationships across the assessment or a subset of the questions (see Table 1). Data was divided into CS1 and CS2 response sets and were considered exclusively and inclusively. A final analysis was applied across the assessment as a whole to draw contrast between CS1 and CS2 responses.

*4.2.1* ***Conditionals.*** Conditionals are fundamental for CS1 students to master early. Student responses indicate that students conceptually understand conditionals and can integrate their knowledge with other core CS1 concept areas.

Students across both sections demonstrated a strong understanding of constructing and evaluating conditionals. In question three, students were required to use an unnatural boolean negation to redirect the control flow of a simple algorithm. 83.6% of the students (60% CS1; 88.64% CS2) were able to correctly identify the negated conditional as correct. Moreover, another 10.9% of students choose a distractor that have valid conditionals but printed values instead of returning them. It is evident that students understand boolean equality (and negated equalities) robustly.

Tracing conditionals based on arbitrary variable values was explored by Simon et. al when exploring benchmarking questions (Selection B) [3]. We opted to remove the else if statement and focus directly on the one level of indirection. Students performed very well on this question, with only four students answering incorrectly, slightly better than correct response percentage presented in [3].

It is evident that students understand conditional statements and their effects on control flow. In questions that incorporated conditionals indirectly, only a small percentage of students would incorrectly evaluate the conditionals. Questions one, six, and eight had distractors that targeted incorrect conditional applications, and student choose responses that demonstrated correct conditional implementation (92.5%, 80.1%, and 96.2% of students, respectively).

*4.2.2* ***Arrays.*** Arrays are one of the fundamental topics for a students success in CS2. As it turns out, they also are one of the most overlooked concepts by students and when intertwined with other themes that often result to failure.

Overall we found that students in CS2 performed particularly well in contrast to CS1 students with average scores for CS2 ranging from 80-100% whereas CS2 scored 20-60%. One possibility for the poor results could be that arrays and lists are left until near the end of the semester to be taught and so students are still trying to grasp this idea.

Question seven was particularly performed poorly by CS1 students at 20% of students getting this question correct. Most CS1 students chose B as their first pick that proves that what is being misunderstood is variable referencing. This shows that students in CS1 understand the array construct, how the elements in the array could potentially be iterated over, but miss the concept of non-alias variables proving that arrays is not the difficulty of this question. Students in CS2 in contrast were better able to understand variable referencing and as a result perform better.

*4.2.3* ***Loops.*** Looping is one of the hardest concepts for CS1 students to grasp causing a need for benchmarking. Some of the things that make loops difficult are the edge/boundary cases, forgetting to increment in a while loop, and assuming that a question is correct without tracing it themselves.

Question two in specific was of keen interest to this research in which we had intended to test what truly makes loops hard. The loop in question two was missing the increment statement in the while loop and we wanted to test whether or not students were able

| Question | Arrays | Loops | Objects | Functions | Conditional | Tracing | Docstring | CS1 | CS2 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | X | X | | X | X | X | | 50% | 81.82% |
| 2 | | X | | | | | | 50% | 95.45% |
| 3 | | | | X | X | x | | 60% | 88.64% |
| 4 | | | | | X | x | | 90% | 93.18% |
| 5 | | | | X | X | x | | 50% | 81.82% |
| 6 | | | | X | | x | X | 60% | 70.45% |
| 7 | X | X | X | | | | | 20% | 84.09% |
| 8 | X | X | | | X | | X | 40% | 95.45% |
| 9 | | | | | X | | X | 70% | 90.91% |
| 10 | | | X | X | X | x | | 10% | 50% |
| 11 | X | X | | | X | x | | 60% | 100% |
| 12 | X | X | | | | X | | 60% | 84.09% |

**Table 1: Questions categorized as themes.**

to recognize this integral component of a successful loop. The distractors in this question, namely, (a) (b) and (c) will tell us if students are overlooking the increment assuming the question is correct and will get caught up in edge cases of loops.

Question two was performed quite well with 50% accuracy in CS1 and 95% accuracy in CS2. Finding what in loops specifically is difficult is essential for computer education research as loops are one of the foundational topics used in all programming languages. This structure of this question can be mimicked to out to nearly all other programming languages proving to be a great benchmarking question

*4.2.4* ***Functions.*** Functions encapsulate many different concepts that range in difficulty. Student responses provide evidence that students lack the ability to integrate different concepts together to correctly understand the semantics of functions [9]. Questions that involved functions had relatively low scores across both sections.

Variable bindings and scoping in function bodies seem to impose difficulty for some students. For question ten, 40% of students failed to correctly deduce the reassignment of variables lives only within the functions scope. As a side note, roughly 19% of students did not recognize that mystery() performed the standard three-variable swap, a common issue amongst students [11].

Student response data suggests that some students lack understanding of return statements and their effects on control flow. Fourteen students choose responses that involve a function returning twice, one for each conditional statement (see question 6 in appendix). We conjecture that the lack of else if use on the second conditional may have reinforced this misconception.

Six students choose a response that would consider print statements equivalent to return statements. The distractor used a conditional more familiar to the students, in comparison to the negated conditional in the correct response. Given five of the students are currently enrolled in CS2, misreading the question description may have contributed to the incorrect responses.

*4.2.5* ***Objects.*** Memory concepts, specifically pointers and allocation, are known to be difficult for students [25]. Expert panels argue a strong foundation in memory related concepts are fundamentally important for a computer scientist to acquire [Goldman]. Questions that involved memory concepts had the lowest scores across both sections.

Conceptual distinctions between variables that bind to immutable objects in comparison to mutable ones was poorly demonstrated by CS1 students. 80% of the students in CS1, in contrast to the 11% in CS2, believed that the iteration variable provides a reference to the integer's memory location in question seven. CS1 students failed to infer the immutability of integers, indicating a deeper misconception of object storage. The better performance by CS2 students is likely due to more familiarity working with such objects directly.

*4.2.6* ***Tracing.*** Students often skim over code wanting to get to a solution quickly without actually tracing it through. While tracing questions are often the questions students guess in, researchers [11] find that students struggle in tracing code.

The objective of question four was targeted at student's tracing ability in nested ifs. If statements are usually learned early on in CS1 and so we suspect students of both CS1 and CS2 to have a fair shot at this question. As expected, 90% of CS1 students got this question correct, and 93% of students in CS2. Students in both CS1 and CS2 had minimal use of distractors.

*4.2.7* ***Description Influence.*** Question descriptions impact how a student interacts with a problem, noted as one of the five influences on student question performance [33]. We attempted to provide question descriptions and docstrings that have consistent formatting across questions. Question descriptions are similar to the format used in the students coursework.

Students were able to translate docstring overviews of the algorithms into code segments well. Questions three, six, eight and nine required students to implement or analyze a block of code based off a short blurb in the question description. Only a small percentage of students chose responses that contradict or fail to conform to the docstring requirements. We suspect that these students skimmed over the questions rather than incorrectly understanding requirements due to the majority of students satisfying the description adequately.

## 4.3 Contrasting Sections

CS2 students scored higher than CS1 students on all questions in the assessment, seen in figure 1. Differences across sections has a mean of 37.9%.

Questions four and six had the most similar performance across sections, 3.18% and 10.45% differences respectively. Questions seven

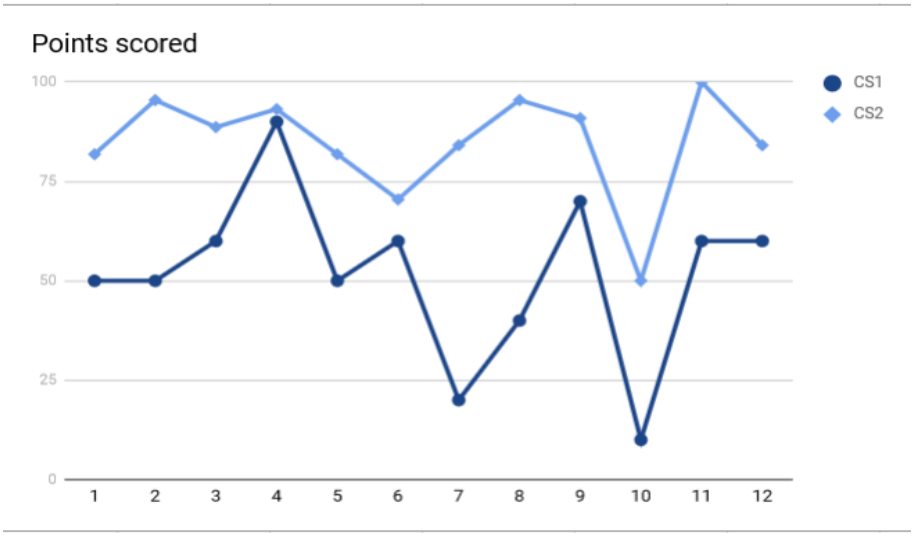


**Figure 1: Mean of student responses per question.**

and eight had the most drastic performance differences across sections, 64.09% and 55.45% respectively.

Evaluating the average scores across topic areas provides insights into what students find particularly difficult. Figure 2 presents the mean of student responses on questions grouped by a major theme. Questions involving objects had the lowest performance across both sections, with question involving conditionals the strongest across both.

No further analysis was completed using the generalized average across themes. Figure 3. simply provides an overview of the categorized student response data.

## 5 DISCUSSION

### 5.1 Trends

How students interact with the question descriptions were not directly measured using our assessment. We can, however, deduce that it is likely that students incorrectly answered questions due to misconceptions rather than unclear descriptions. Responses that would likely indicate such behaviour (e.g. question 3, distractor e) were often coupled with similar misconceptions later in the assessment (question 6, distractor a and d). Using different descriptions would invalidate our data and would require retesting [7]. We suspect that question descriptions and docstrings provide sufficient information for students to correctly answer the questions and do not pose a threat to a student's cognitive load.

Questions that targeted loops, conditionals, and arrays were performed well. However, when intertwined with other topics, such as variable referencing, often resulted in poor performances. One reason for this could be because students fail to trace code correctly. In question five specifically, the most common distractor chosen was (c), in which students look at the greater than inequality operator ('>=') and look for an answer with the closest relation. Without tracing through the whole question line by line, students appear to skim through the code and look fixate on what they deem as the "important" parts.

Using questions that target conditionals directly will likely obtain strong student performance comparable to student responses on question four and selection B in [3]. On the other hand, questions that incorporate conditionals expose student misconceptions when integrating concepts together and provide exposure to a student's performance using and evaluating conditionals in a non-isolated setting.

Functions present an often-overlooked cognitive complexity to a question. Benchmarking questions that incorporate functions must carefully consider the validity threats it imposes, specifically when functions are not the intended target topic area. Questions used in future benchmarks should mitigate the cognitive load and narrow in on a specific concept related to functions. (Return statements may be of particular importance.) Further research is required to more accurately depict the misconceptions students face.

### 5.2 Influence of Python

Restricting our benchmarking questions to Python likely tainted our data trends. The nature of Python applies a level of abstraction that may be difficult for students to break down, especially when dealing with concepts that requiring students to understand the inner-workings of library calls.

Slicing is a common practice for copying portions of a list in Python. Although exposed to such notation in their courses, misconceptions regarding object slicing is likely a factor in the poor performance on question 10. A portion of the 40% of students that inferred the result of the swap had a global effect may have used the slicing operator as their reasoning. Time constraints did not allow for the researchers to remove the slicing notation and retest the question to determine the extent the operator had on student performance.

We suspect that the abstracted syntax and the "everything is an object" ideology of Python may complicate a student's deductions on whether an object is immutable or not. Common CS1 languages (e.g. C) provide clear syntactic evidence to whether a variable is serving

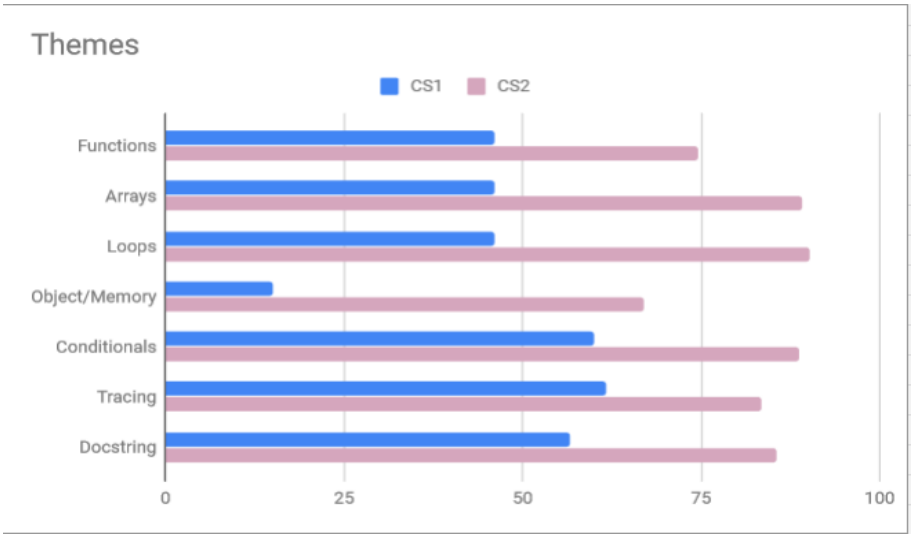


**Figure 2: Mean across major themes.**

as a pointer or not. Memory questions appear to require exclusive testing to ensure consistent performance.

### 5.3 CS1 and CS2

A sound CS1 benchmarking question should easily differentiate between pre and post CS1 knowledge.

Questions four is likely too easy for a benchmarking question due to the high student performance and low differentiation between sections. Question six had below average performance for CS2 students, potentially indicating a good potential benchmarking question due to targeting a topic area with misconceptions that coexist across sections. Low CS2 performance may be a result of skimming the question description and will need to be further tested.

Question seven and eight had significant performance differences across sections. Due to the familiarity that CS1 students have with such type of questions and formatting (derived from coursework), we deduce that they target concepts that are heavily reinforced in CS2. They provide a valid benchmark and accurately target common CS1 misconceptions (see Appendix II).

### 5.4 Qualities of Benchmarking Questions

Questions revolving around an atomic theme is essential to standardize benchmarking questions. While difficult in nature, limiting the number of interdependencies in a question provide a stronger analysis on whether the students conceptual grasps the intended topic area. It may be the case that a given student understands the targeted topic but due to a misconception in an intertwined topic, they produce an incorrect response.

The style of a benchmarking question can greatly affect students performance. A BRACElet [14] study finds that it is easier for students to select the purpose of a piece of code than to come up with the purpose on their own. This is the primary reason we opted for a multiple choice format in our benchmarking questions. Moreover, Parsons problems, short answer questions, debugging, and designing code questions are not only subjected to a variation of marking styles, but are also significantly more difficult to students.

We also found the limiting the cultural reference in a question enhances benchmarking questions as seen as well in [3]. Assuming students understand the scenario or specific vocabulary in a given question increases the chances for students to get this question incorrect. It is important to limit these assumptions for a more accurate result.

## 6 THREATS TO VALIDITY

One of the limitations of this research is that this was conducted through an online survey form due to time constraints. Because of this, students had the ability of testing code externally, and/or searching for the code online before submitting their responses. If done in a classroom setting, the distribution of marks may have differed. Due to the lack of grading incentive, we propose that the number students that may have abused the opportunity is minimal. A threat to validity is then subjected to this data distribution.

This assessment was conducted after the midterm evaluations of the second term, posing a bias on CS1 students. The distribution of CS1 students enrolled during this time are either non-computer science students, or computer science who have failed CS1 last semester. Since students in CS2 have completed over half the course, they are more likely in achieving higher marks and polluting our data. To mitigate this limitation, it would have been ideal if the assessment was distributed during the first term of the school year.

The small sample size, standing at 54 student submissions, subjects our analysis with biases that threatens the validity of our data.

## 7 FUTURE WORK

This report is a proposal of benchmarking questions that have been used on CS1 students sectioned into two courses through an online survey form. The validity of the conclusions made would be greatly enhanced if done on a larger number of students. For this, we call

on other educators. Appendix I includes the sample quiz that can be done on students, should professors choose from other universities.

Research on visualization of programming specifically in regards with parameter passing like returns will be valuable to Computing Education. Although past research suggests that subroutines and procedures were difficult in the past [], there was a strand of research done on why returns specifically are difficult to students of today's age.

## 8 CONCLUSION

Developing effective benchmarking questions requires rigorous testing and validating.To combat this, researchers often seed questions into CS1 examinations to analyze their performance, an often difficult task. Questions that fail to produce a desired benchmark are often discarded, likely due to the time constraints of retesting.

We have presented and analyzed 12 potential benchmarking questions under 54 student responses across CS1 and CS2. Each question was initially validated under a panel of researchers before being distributed as an online assessment. After collecting student response data, each question was validated by comparing student response distributions across individual questions and major themes.

Although the researchers were confined to a strict time constraint, reevaluating and retesting a question can be obtained through simple redistribution. This provides an effective and efficient way of validating potential benchmarking questions prior to their seeding into examinations.

The main contributions of this paper was to explore new ways to developing and validating benchmarking questions. Moreover, our assessment is composed of a set of validated potential benchmarking questions that can be used in future research.

## REFERENCES


[1] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles, "Identifying Important and Difficult Concepts in Introductory Computing Courses using a Delphi Process," *SIGCSE Bull.*, vol. 40, pp. 256âĂŞ260, Mar. 2008

[2] A. Luxton-Reilly, B. A. Becker, Y. Cao, R. McDermott, C. Mirolo, A. MĬLuhling, A. Petersen, K. Sanders, Simon, and J. Whalley, "Developing Assessments to Determine Mastery of Programming Fundamentals," in *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, ITiCSE-WGR '17, (New York, NY, USA), pp. 47âĂŞ69, ACM, 2017

[3] Simon, J. Sheard, D. D'Souza, P. Klemperer, L. Porter, J. Sorva, M. Stegeman, and D. Zingaro, "Benchmarking Introductory Programming Exams: Some Preliminary Results," in *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, (New York, NY, USA), pp. 103âĂŞ111, ACM, 2016

[4] J. Sheard, Simon, J. Dermoudy, D. D'Souza, M. Hu, and D. Parsons, "Benchmarking a set of exam questions for introductory programming," in *Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148*, ACE '14, (Darlinghurst, Australia, Aus-tralia), pp. 113âĂŞ121, Australian Computer Society, Inc., 2014

[5] Simon, J. Sheard, D. D'Souza, P. Klemperer, L. Porter, J. Sorva, M. Stegeman, and D. Zingaro, "Benchmarking Introductory Programming Exams: How and Why," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, (New York, NY, USA), pp. 154âĂŞ159, ACM, 2016.

[6] A. E. Tew and M. Guzdial, "The FCS1: A Language Independent Assessment of CS1 Knowledge," in *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Educa-tion*, SIGCSE '11, (New York, NY, USA), pp. 111âĂŞ116, ACM, 2011.

[7] M. C. Parker, M. Guzdial, and S. Engleman, "Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment," in *Proceedings of the 2016 ACM Conference on In- ternational Computing Education Research*, ICER 16, (New York, NY, USA), pp. 93âĂŞ101, ACM, 2016.

[8] R Lister, ES Adams S Fitzgerald, W Fone, J Hamer, M Lindholm, R McCartney, JE Mostrom, K Sanders, O Seppala, B Simon, and L Thomas (2004). A multi-national study of reading and tracing skills in novice programmers. SIGCSE Bulletin, 36:4, 119-150

[9] Detienne, F, and Soloway, E. (1990) An empirically derived control structure for the process of program understanding. Int. J. of Man-Machine Studies, 33, pp. 323-342

[10] J Sheard, Simon, A Carbone, D Chinn, T Clear, M Corney, D D'Souza, J Fenwick, J Harland, M-J Laakso, and D Teague (2013). How difficult are exams? A framework for assessing the complexity of introductory programming exams. 15th Australasian Computing Education Conference (ACE 2013), 145-154.

[11] Simon (2011). Assignment and sequence: why some students can't recognise a simple swap. 11th Koli Calling International Conference on Computing Education Research, 21-30.

[12] Simon, J Sheard, A Carbone, D D'Souza, J Harland, and M-J Laakso (2012). Can computing academics assess the difficulty of programming examination questions? 11th Koli Calling International Conference on Computing Education Research, 160-163.

[13] Sheard, Judy, et al. âĂIJExploring Programming Assessment Instruments.âĂİ Proceedings of the Seventh International Workshop on Computing Education Research - ICER 11, 2011

[14] Simon and S Snowdon (2014). Multiple-choice vs free-text code-explaining examination questions. 14th Koli Calling International Conference on Computing Education Research, 91-97

[15] J Whalley, R Lister, E Thompson, T Clear, P Robbins, PKA Kumar, and C Prasad (2006). An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. Eighth Australasian Computing Education conference (ACE 2006), 243-251.

[16] Simon, Sheard, J., Carbone, A., D'Souza, D., Harland, J. and Laakso, M.-J. (2012): Can computing academics assess the difficulty of programming examination questions? In proceedings of the 11th Koli Calling International Conference on Computing Education Research, Finland

[17] Andrew Luxton-Reilly, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas MÃijhling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. 2017. Developing Assessments to Determine Mastery of Programming Fundamentals. In Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17). ACM, New York, NY, USA, 388-388

[18] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2008. Identifying important and difficult concepts in introductory computing courses using a delphi process. In Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08). ACM, New York, NY, USA, 256-260. DOI:

[19] Sheard, J., Simon, Carbone, A., Chinn, D., Clear, T., Corney, M., D'Souza, D., Fenwick, J., Harland, J., Laakso, M.-J. and Teague, D. (2013): How difficult are exams? A framework for assessing the complexity of introductory programming exams. In proceedings of the 15th Australasian Computing Education Conference (ACE 2013), Adelaide, Australia

[20] S. Bergin and R. Reilly. The influence of motivation and comfort-level on learning to program. In Proceedings of the 17th Annual Workshop og the Psychology of Programming Interest Group pages 293-304, University of Sussex, Brighton UK 29 June - 1 July, 2005. University of Sussex

[21] Kolikant, Y. B-D. and Mussai, M. 2008. "So my program doesn't run!" Definition, origins, and practical expressions of students' (mis)conceptions of correctness, Computer Science Education, 18, 2 (Jun. 2008), 135-151.

[22] Pea, R. D. 1986. Language-independent conceptual "bugs" in novice programming. Journal of Educational Computing Research, 2, 1 (1986), 25-36.

[23] Goldfinch, T., Carew, A. L., Gardner, A., Henderson, A., McCarthy, T. and Thomas, G. (2008): Cross-institutional comparison of mechanics examinations: A guide for the curious. Paper presented at the Australasian Association for Engineering Education conference (AAEE), Yeppoon.

[24] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey L. Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2010. Setting the Scope of Concept Inventories for Introductory Computing Subjects. Trans. Comput. Educ. 10, 2, Article 5 (June 2010), 29 pages.

[25] T. Jenkins. On the Difficulty of Learning to Program. In Proceedings for the 3rd Annual conference of the LTSN Centre for Information and Computer Sciences, Loughborough, UK August 27 - 29, 2002.

[26] Simon, Judy Sheard, Michael Morgan, Andrew Petersen, Amber Settle, and Jane Sinclair. 2018. Informing students about academic integrity in programming. In Proceedings of the 20th Australasian Computing Education Conference (ACE '18). ACM, New York, NY, USA, 113-122.