Designing and Validating Multiple Assessment Options with Equivalent Difficulty for Introductory Programming

Charlotte Pierce
Monash University
Melbourne, Australia
charlotte.pierce@monash.edu

Andrew Cain
Monash University
Melbourne, Australia
andrew.cain@monash.edu

Lashi Bandara
Deakin University
Geelong, Australia
lashi.bandara@deakin.edu.au

Nicolas Pallant
Deakin University
Geelong, Australia
s224119907@deakin.edu.au

Abstract

Introductory programming (i.e., CS1) is a key subject in any computing degree, as students who do not succeed are significantly more likely to discontinue their studies. As such, maintaining motivation and engagement is critical to increasing student's chance of success.

In this paper we explore the potential for increasing introductory programming student's motivation and engagement by offering multiple assessment options. These options are designed to cover the same concepts, and be of equivalent difficulty, but are themed differently. Our goal is to increase the likelihood that each student will find at least one option personally relatable and motivating. In order to achieve this, we also investigate methods for estimating the difficulty of a programming task based on an exemplar solution, so that we can ensure the assessment options are equivalent.

We found that students reacted positively to being given assessment options, and that there were no negative impacts to student's learning outcomes based on the assessment theme they chose. We also demonstrate the potential of several static code metrics in estimating the difficulty of introductory programming tasks.

CCS Concepts

• **Social and professional topics** → **CS1**; **Information technology education**; **Software engineering education**; Student assessment.

Keywords

CS1, Introductory Programming, CS Education, Engagement, Assessment Design

ACM Reference Format:
Charlotte Pierce, Andrew Cain, Lashi Bandara, and Nicolas Pallant. 2026. Designing and Validating Multiple Assessment Options with Equivalent Difficulty for Introductory Programming. In *28th Australasian Computing Education Conference (ACE 2026), February 09–13, 2026, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3786228.3786242

ACE 2026, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-2352-0/26/02
https://doi.org/10.1145/3786228.3786242

1 Introduction

Introductory programming, also referred to as CS1, is one of the most important subjects in a computing degree, as student's success here has a large impact on whether they continue their studies [7]. A key factor in student's success is the time they spend practicing their programming skills and attempting assessments, which is largely determined by their level of engagement and motivation [18]. As such, much research focuses on how to improve introductory programming education, particularly student engagement.

In this paper we focus on a specific intervention for improving student's satisfaction and engagement with introductory programming assessments: offering multiple assessment options. Each option covers the same concepts, and is designed to be of similar difficulty, but is themed differently. The intention is that by offering differently themed assessment tasks, we will increase the chance that each student will find an assessment option that they find relatable and motivating. This is consistent with literature noting that it is difficult to design a single assessment that will appeal to an entire cohort [5].

Our intervention is a form of flexible assessment, which is increasingly popular as a mechanism to improve student motivation and engagement across a variety of disciplines [2, 20], particularly to accommodate different learning needs [6, 23]. In this way, our specific approach of offering different but equivalent assessment options purely in the interest of variety is fairly unique.

The remainder of this paper is structured as follows. Section 2 discusses the current literature on choices in assessments and estimating programming task difficulty. In Section 3 we describe our teaching context, as well as our approach to designing and evaluating equivalent assessment tasks for an introductory programming unit. The results of our work are presented in Section 4, with a discussion of their implications and limitations in Section 5.

1.1 Research Questions

In this study we seek to explore the following research questions:

(1) How can multiple assessment options be designed for introductory programming without compromising overall assessment quality?
(2) What measures are appropriate for estimating the difficulty of introductory programming tasks?

(3) How do students respond to having different assessment task options?

2 Background

There are many studies exploring the development of assessments of equivalent difficulty. However, the majority of this work focuses on designing alternative assessments to accommodate student disability [6, 9, 23, 31]. Typically, this involves either allowing students to choose the format of submission [19] or the mode of assessment [3, 20]. This work, while valuable, is qualitatively different to ours, as we want to maintain the same assessment mode and format across our assessment options.

At the time of writing we were unable to find any existing work similar to ours. However, there is some literature exploring methods for estimating the difficulty of a programming task [27]. This work is typically focused on creating an appropriate escalation of difficulty of assessments across a teaching period, or creating equivalent code snippets for tests or multiple choice quizzes.

The difficulty of a programming task could be estimated from the problem statement, an exemplar solution, or student behaviour and self-reported perceptions [13]. Most studies avoid using the problem statement, as doing so introduces confounding factors in terms of reading comprehension. Instead, the most common approach is to rely on trialling tasks with a set of subjects [1, 11, 18, 24, 28, 29]. This does not scale well, as it requires a continuous pool of testers who are at an appropriate level of skill, and ideally a fair way to compensate them. Some attempt to avoid this issue by using a real student cohort, and collecting data on student's behaviour, success, and perceptions [12, 18]. However, this approach introduces a lag whereby educators can only react to the measured difficulty of a task after it has already been given to students.

Whalley and Kasto [30] suggest that static software metrics, applied to an exemplar solution, are a useful measure of difficulty. This suggestion is followed by Dobias et al. [12], who notes that the majority of scalable difficulty metrics are based on source code statistics.

One commonly used metric is lines of code. It is simple and easy to calculate, but surprisingly indicative of program complexity at an introductory programming level [4, 27].

Another common metric is McCabe's cyclomatic complexity [22]. This is one of one of the most widely accepted metrics for static software [27], which measures the number of linearly independent execution paths through a program's source code.

Finally, the last most commonly used metric is Halstead's measure of difficulty [14, 15]. This quantifies a program in terms of the total and unique number of operators and operands within the source code.

Curtis et al. [10] showed that Halstead's measure of difficulty and McCabe's cyclomatic complexity together are correlated with the difficulty of understanding and modifying a piece of software. All three metrics have been used to estimate program difficulty of student's code [17, 18, 25, 28, 30].

3 Method

3.1 Teaching Context

In this paper we discuss assessment options that we implemented in a first year introduction to programming unit (i.e., CS1) in a large Australian university. This work received approval from the Deakin University Human Ethics Advisory Group (project ID: SEBE-2024-03). The unit had approximately 700 enrolments, comprising students in core computing degrees and those from other disciplines taking the unit as an elective. Although the unit was delivered in on-campus and online modes simultaneously, we focus only on the on-campus cohort.

3.2 Task Design

Across the teaching period students were asked to complete several assessment tasks. From these, we chose 6 where we offered differently themed options. These 6 tasks formed the second to seventh assessments in the unit, placed after an initial task requiring students to set up their development tools.

Each task had a specific conceptual focus, and specified the programming language students had to use. This is summarised in Table 1. Our intention with the chosen languages was to support student's practice of the selected concept with as minimal cognitive overhead as possible. For this reason we started with C# top-level statements, then moved to C++. Across all 6 tasks we used the following themes for the three options:

(1) Terminal:
Based on tasks typically seen in introductory programming courses, these programs only required terminal-based interactions with users. Examples include calculators, unit converters, and text menus. The SplashKit library[1] was used to abstract terminal input and output into simple `read_line` and `write_line` calls.
(2) Media:
Also terminal based, but themed towards the creation of a music player using the SplashKit library. Examples include loading and playing song files, creating playlists, and entering ratings.
(3) Game:
Simple games, also using the SplashKit library. Examples include basic movement, collisions, and score tracking.

Our goal with the three themes was to offer a variety of focuses and increase the chance that each student would find a topic that engaged them. We chose the SplashKit library as it supports the languages used in the course, the themes chosen for the assessment options, and was specifically designed for beginning programmers [26].

3.3 Estimating Task Difficulty

Before the teaching period, we attempted to estimate the difficulty of the new assessment tasks as they were developed to ensure they were equivalent. To do this we applied both qualitative and quantitative measures.

The qualitative approach involved regular presentation of the options to a focus group of three expert educators, each of whom

[1] https://splashkit.io/

Task	Focus	Language
1	Sequence	C# top-level statements
2	Selection and iteration	C# top-level statements
3	Functions and procedures	C++
4	Structs	C++
5	Pointers and pass-by-reference	C++
6	Arrays	C++

Table 1: The focus of each assessment task, and the programming language students were required to use

have over a decade of experience teaching introductory STEM subjects. For the quantitative approach we applied three metrics to our exemplar solutions:

(1) Lines of code:
The total number of non-commented lines of code, after formatting it to meet language style guidelines. This is a naive metric, but does give a measure of program size [27], and was taken as a starting point.
(2) Cyclomatic complexity:
The cyclomatic complexity as defined by McCabe [22] was calculated for each program.
(3) Halstead difficulty:
The Halstead complexity metrics for software [14, 15] include a measure of program difficulty. To calculate this, we used the definitions of operators and operands from Halstead [15]: operators either alter the value of an operand, or alter the order in which the value of an operand is altered (i.e., operators 'do something'); operands are any variable or constant (i.e., data).

3.4 Validating Task Difficulty

To validate our estimates of task difficulty we randomly selected 10 passing student submissions from the terminal and game options for each of the 6 tasks. We then applied the same quantitative metrics as used on our sample solutions to the student work, and compared the results. We did not apply this validation to the media tasks as there were insufficient passing submissions for the theme.

3.5 Measuring Student Response

We used three strategies to measure student's response to the assessment options. First, we looked at the number of students who chose each task option across the 6 tasks, and then correlated their choice with their final grade for the unit.

We also asked students to complete an optional survey, adapted from Chu et al. [8], Hwang et al. [16], and Kılıç et al. [21], measuring self-reported satisfaction, mental load, and conceptual ability. The full set of questions is provided in Appendix A.

Finally, we collected staff observations over the teaching period.

4 Results

4.1 Task Difficulty

4.1.1 Difficulty Estimation. The number of lines of code and the cyclomatic complexity for the examplar solutions, shown in Figures 1 and 2, were roughly equivalent across the tasks. The only

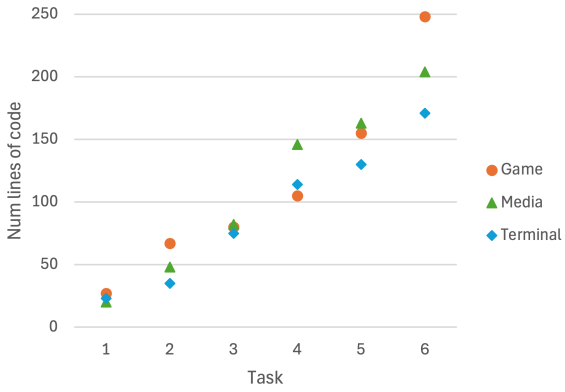

Figure 1: Number of lines of code in each exemplar solution

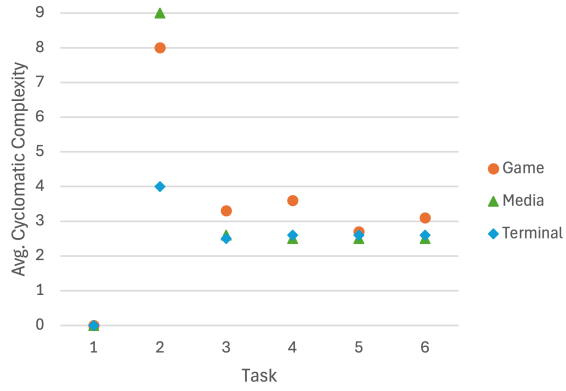

Figure 2: Average cyclomatic complexity for each exemplar solution

real outlier was the cyclomatic complexity for task 2, which was quite varied as a consequence of including both a mouse click and collision check in the game option, and validation checking the existence of the user's selected song file in the media option.

When comparing the Halstead difficulty of the tasks, shown in Figure 3, the game option becomes increasingly more difficult compared to the media and terminal options, which increase in difficulty over the 6 tasks but remain roughly equivalent to each other. This finding is consistent with the opinion of the expert panel, who noted several times that the game options appeared to be significantly more difficult. Unfortunately, their feedback was difficult to appropriately action given the timing of the project with the start of the teaching period.

4.1.2 Difficulty Validation. Comparing the metrics on the examplar solutions to randomly selected student work revealed similar results. As shown in Figures 4 and 5, the number of lines of code and the cyclomatic complexity of the student work followed a similar pattern to the exemplars. It can also be seen that the spread of number of lines of code in the student work increased over the 6 tasks. This is expected, as the reduced scaffolding of instructions

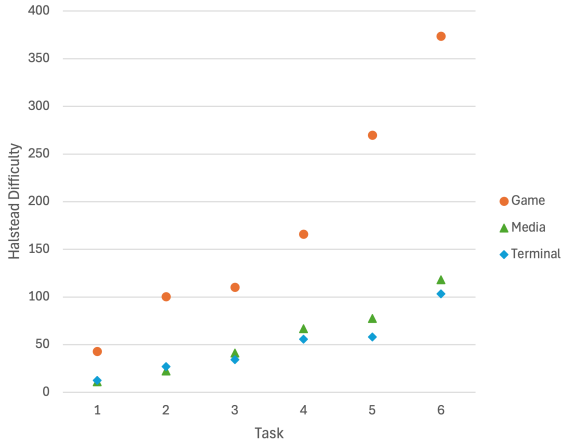

Figure 3: Halstead difficulty for each exemplar solution

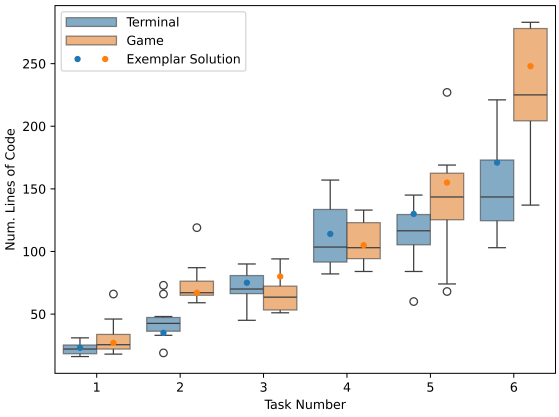

Figure 4: Spread of the number of lines of code of randomly selected student tasks, compared to the exemplar solutions

over time presented increasing opportunities for interpretation and variation in correct solutions.

When comparing the spread of Halstead difficulty of student work to the exemplar solutions, Figure 6 again shows a notable difference between the terminal and game options. This is paired with a much higher spread of difficulty in the student solutions for game tasks, implying significant room for interpretation and approach to the task instructions.

4.2 Student Response

4.2.1 Task Selection. Table 2 shows that the majority of students chose the terminal option for all 6 tasks. The next most common choice was to complete a mix of the task options. For those that chose a mix, it was most common for students to complete the terminal option for the first five tasks, then switch to the game option for task 6. A small minority of students completed all of the game tasks, and a very small number of students completed the sequence of media themed tasks.

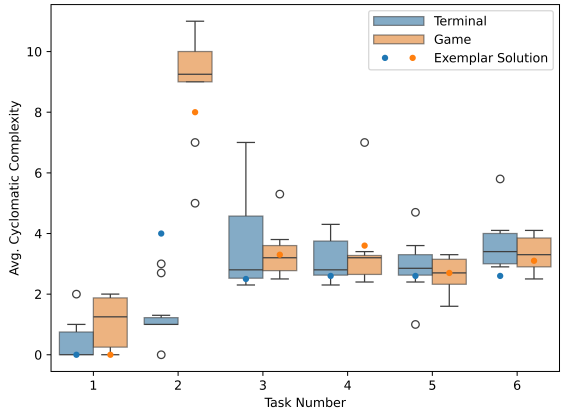

Figure 5: Spread of average cyclomatic complexity of randomly selected student tasks, compared to the exemplar solutions

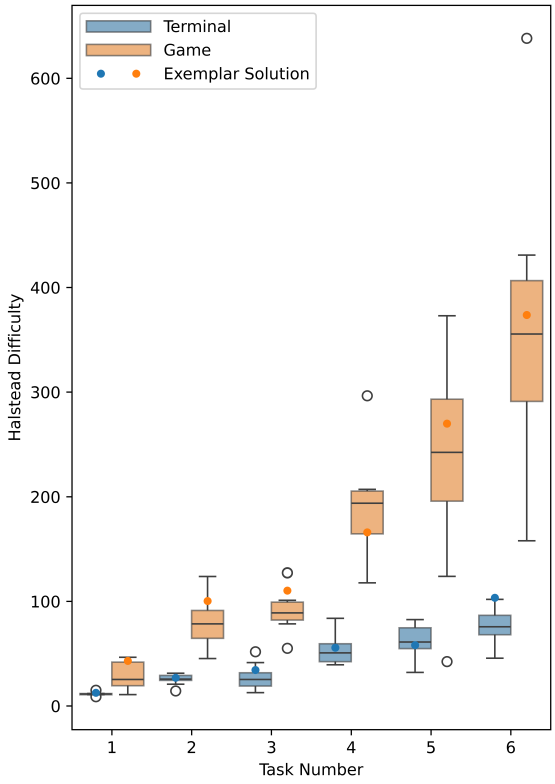

Figure 6: Spread of Halstead difficulty of randomly selected student tasks, compared to the exemplar solutions

There was no statistical significance between the task option a student selected and the final grade they achieved. However, as shown in Table 3, the students who chose the game themed sequence did tend to get a higher final grade.

Task Type	Percentage of Students
Terminal	73.67
Media	1.06
Game	5.32
Mix	19.95

Table 2: The percentage of students who chose each task option, rounded to 2 decimal places

	Terminal	Media	Game	Mix
Fail	16.97	0	0	10.67
Pass	50.54	100	40	46.67
Credit	15.88	0	25	25.33
Distinction	13.36	0	20	12
High distinction	3.25	0	15	5.33

Table 3: The percentage of students who received each possible final grade, split by task choice, rounded to 2 decimal places

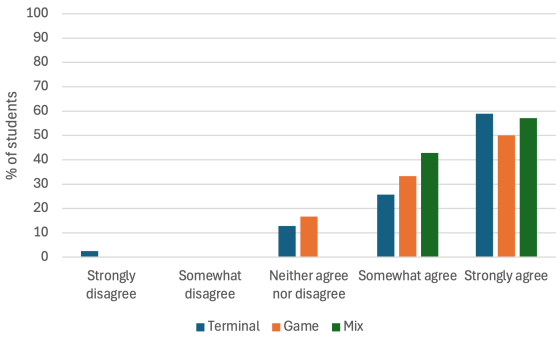

Figure 7: Typical distribution of responses for each survey question

4.2.2 Student Survey. Only a small subset of students opted to complete the survey. From those, only one chose the media sequence of tasks exclusively. As such, we did not include that student in our analysis. We also excluded responses from any student who did not finish the entire survey.

Of the remaining 52 respondents, 39 completed the terminal tasks, 6 completed the game tasks, and 7 completed a mix. 81% of all respondents reported finding their chosen tasks enjoyable, and the same proportion reported finding their chosen tasks challenging. None of the students who chose the game tasks responded negatively to either question.

For the remaining questions, the response distributions all looked similar to that shown in Figure 7, with some students responding negatively or neutrally but otherwise expressing a strongly positive sentiment. This indicates that students were generally satisfied with the assessment tasks, felt that the mental load and effort required was not onerous, and that their programming ability improved as a result of the work.

4.2.3 Teaching Staff Observations. Staff only reported two consistent observations over the teaching period. The first was that some students felt the game tasks were more difficult than the other options. It is not clear whether this came from a 'gut feeling', word of mouth, or students actually reading the task instructions.

The second observation was that students appeared to be strongly biased by the presentation order of the task options. In the teaching material the tasks were shown in sequence, with the terminal option first, media second, and game last. Several staff members reported that many students just completed the first option.

5 Discussion

From this initial exploratory study, we have learned a number of lessons in how to approach designing equivalent assessment options. Aside from the clear difference in difficulty of the game tasks, a significant misstep was having tasks build upon each other. Most of the tasks directly built on the previous task of the same theme, which exponentially increased the effort required to switch task themes the further into the sequence a student got. This cost of switching is likely the reason that most students who chose a mix of task options switched to the game option for task 6, as this was the only game task that did not directly build upon any of the previous ones.

It is unclear why there was a strong trend towards the terminal tasks. We may have introduced bias into student's choice in the way the tasks were presented. By structuring the options in sequence, students seemed liable to miss some options, or simply pick the first one. This likely biased selection towards the terminal option, as reported by staff. It is also possible that there was snowball affect here. That is, that once a certain number of students had chosen the terminal option, it increased the likelihood of other students doing the same so that they could compare progress with their peers and more directly and help each other.

Despite this, results indicate that offering differently themed assessment options was well received by the students. They also indicate that this assessment structure had no impact on student's ability to achieve the learning outcomes of the unit, given the lack of statistical significance between student's choice and their final result.

However, it is clear that ensuring all assessment options are equivalent in terms of difficulty and effort remains a challenge. The metrics we used to estimate task difficulty appear promising. All three showed similar results between the exemplar solutions and randomly selected student work. The Halstead difficulty in particular also aligned well with the expert panel's judgement. These results suggest that quantitative estimation of task difficulty is a valid approach when designing equivalent assessment options.

5.1 Limitations

The largest limitation of this study is the sample size, both in terms of how many students completed the opt-in survey, and the number of students who chose each theme of assessment option. A related limitation is the bias we likely introduced in the ordering of assessment options. As discussed above, by placing the terminal option first, students were probably more inclined to choose it. Similarly, the choice to have assessment tasks build on each other would

have impacted the probability of students choosing to switch task themes.

By making the survey opt-in, it is likely that the students who completed it were those who were more engaged with the unit. This may have biased results towards positive responses. However, we were careful in our messaging to emphasise that we welcomed all kinds of responses and feedback, which hopefully mitigated this bias to some degree.

Finally, it is important to note that none of our chosen complexity metrics capture context. This was a key factor in our decision to apply several metrics, so that they could be compared and interpreted as a group. The consistency between the metrics on the student and exemplar work, and the alignment of Halstead difficulty with the expert panel, indicates that these metrics are a valid approach.

5.2 Future Work

The clearest avenue for future work is to repeat this study after improving the relative difficulty of the game themed tasks, removing the introduced bias in task selection, and redesigning the tasks to be discrete instead of building upon each other. This would further reveal whether the approach has merit.

Another aspect to explore is what theming for task options would be most appropriate, and how many options should be offered. Our initial selection of themes was based off experience, and an intention to add variety. We decided on 3 total themes as we felt it would be a manageable balance between variety and the workload of designing the tasks. Neither decision was based on any empirical data. Future work in gathering evidence to inform these decision would be of high value.

Finally, further exploration of appropriate automated metrics for estimating programming task difficulty would provide an opportunity for creating practice materials to match student's capabilities. That is, to scaffold students through increasingly difficult exercises. This could be combined with methods for automatically generating practice exercises, to support students having a sufficient quantity of practice material that is specifically targeted to the right level of difficulty for them.

6 Summary

In this paper we demonstrated the potential of offering differently themed assessment options in increasing student's motivation and engagement in introductory programming. We also showed the validity in using a combination of lines of code, McCabe's cyclomatic complexity, and Halstead's difficulty measure in estimating task difficulty. Although some unintentional bias in the study design limits their generalisation, the results suggest that further work in this area would be of value.

A Student Survey Questions

Aside from the first question, which offered students 3 check-boxes (for Terminal, Media, and Game), responses were given on a 5-point Likert scale with the options: Strongly disagree, Somewhat disagree, Neither agree nor disagree, Somewhat agree, and Strongly agree.

General questions:

- In the "Test Your Knowledge" activities, which option(s) did you complete?
- I found these tasks enjoyable (asked for each theme tried).
- I found that these tasks challenged me (asked for each theme tried).

Questions relating to student satisfaction:

- The assessment activities made me better understand how to identify and classify different programming elements.
- The assessment activities were not easy to complete, but it made it easy to understand what I was learning.
- The assessment activities were more challenging and interesting than in other units I am taking.
- I had new findings or knowledge about programming owing directly to the assessment activities.
- I have tried new ways or thinking styles to learn owing directly to the assessment activities.
- The guidance provided in the assessment activities is helpful to me in learning how to identify different programming elements.
- When completing the assessment activities, I learned how to observe the programming elements taught from a new perspective.

Questions relating to mental load and effort:

- The tasks in the assessment activities were difficult for me.
- I had to put a lot of effort into completing the tasks in the assessment activities.
- It was troublesome for me to complete the tasks in the assessment activities.
- I felt frustrated completing the tasks in the assessment activities.
- I did not have enough time to complete the tasks in the assessment activities.
- During the assessment activities, the content presentation caused me a lot of mental effort.
- The way the assessment activities were delivered were difficult to follow and understand.

Questions relating to conceptual ability:

- I can use loop structures (for, while, etc.) appropriately.
- I can use decision structures (if-else, switch-case) appropriately.
- I can determine the suitable datatype (string, int, char, float, etc.) for a variable.
- I can use mathematical (<, >,<=, >=, =, etc.) and logical operators (and, or, etc.) appropriately.
- I can use general methods [write(), read(), delay(), etc.].
- I can code programs in which decisions (if-else, switch-case) and loops (for, while) can be used together.
- I can determine which of the similar structures (if-switch, for-while) would be more appropriate to use.
- I know which variables I would use before I start coding.
- I can decide how to split complex programs into smaller pieces.
- I feel like I can program without relying on SplashKit.
- I found these tasks enjoyable.
- I found that these tasks challenged me.

References

[1] Andres Alvarez and Terry A Scott. 2010. Using student surveys in determining the difficulty of programming assignments. *Journal of Computing Sciences in Colleges* 26, 2 (2010), 157–163.

[2] Lumbini Barua and Barbara Lockee. 2025. Flexible Assessment in Higher Education: A Comprehensive Review of Strategies and Implications. *TechTrends* (2025), 1–9.

[3] Lumbini Barua and Barbara B Lockee. 2024. A review of strategies to incorporate flexibility in higher education course designs. *Discover Education* 3, 1 (2024), 127.

[4] Victor R Basili. 1980. Qualitative software complexity models: A summary. *Tutorial on models and methods for software management and engineering* (1980).

[5] Jessica D Bayliss and Sean Strout. 2006. Games as a" flavor" of CS1. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*. 500–504.

[6] Stephanie Cawthon and V Shyyan. 2022. Accessibility and accommodations on large-scale standardized assessments.

[7] A Christopher Strenta, Rogers Elliott, Russell Adair, Michael Matier, and Jannah Scott. 1994. Choosing and leaving science in highly selective institutions. *Research in higher education* 35 (1994), 513–547.

[8] Hui-Chun Chu, Gwo-Jen Hwang, Chin-Chung Tsai, and Judy CR Tseng. 2010. A two-tier test approach to developing location-aware mobile learning systems for natural science courses. *Computers & Education* 55, 4 (2010), 1618–1627.

[9] Deborah Craddock and Haydn Mathias. 2009. Assessment options in higher education. *Assessment & Evaluation in Higher Education* 34, 2 (2009), 127–140.

[10] Bill Curtis, Sylvia B. Sheppard, Phil Milliman, MA Borst, and Tom Love. 1979. Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. *IEEE Transactions on software engineering* 2 (1979), 96–104.

[11] Brian De Alwis, Gail C Murphy, and Shawn Minto. 2008. Creating a cognitive metric of programming task difficulty. In *Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*. 29–32.

[12] Vaclav Dobias, Vaclav Simandl, and Jiri Vanicek. [n. d.]. Number of Program Builds: Another Criterium for Assessing Difficulty of a Programming Task? 23, 3 ([n. d.]), 525–540.

[13] Tomáš Effenberger, Jaroslav Cechák, and Radek Pelánek. 2019. Difficulty and complexity of introductory programming problems. *Educational Data Mining in Computer Science Education (CSEDM)* (2019).

[14] Maurice H. Halstead. 1977. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc.

[15] Maurice H. Halstead. 1979. Advances in software science. In *Advances in Computers*. Vol. 18. Elsevier, 119–172.

[16] Gwo-Jen Hwang, Li-Hsueh Yang, and Sheng-Yuan Wang. 2013. A concept map-embedded educational computer game for improving students' learning performance in natural science courses. *Computers & Education* 69 (2013), 121–130.

[17] Petri Ihantola and Andrew Petersen. 2019. Code complexity in introductory programming courses. (2019).

[18] Petri Ihantola, Juha Sorva, and Arto Vihavainen. 2014. Automatically detectable indicators of programming assignment difficulty. In *Proceedings of the 15th Annual Conference on Information technology education*. 33–38.

[19] Brian Irwin and Stuart Hepplestone. 2012. Examining increased flexibility in assessment formats. *Assessment & Evaluation in Higher Education* 37, 7 (2012), 773–785.

[20] Giel Kessels, Kate Xu, Kim Dirkx, and Rob Martens. 2024. Flexible assessments as a tool to improve student motivation: An explorative study on student motivation for flexible assessments. In *Frontiers in Education*, Vol. 9. Frontiers Media SA, 1290977.

[21] Servet Kılıç, Seyfullah Gökoğlu, and Mücahit Öztürk. 2021. A valid and reliable scale for developing programming-oriented computational thinking. *Journal of Educational Computing Research* 59, 2 (2021), 257–286.

[22] Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering* 4 (1976), 308–320.

[23] Ceri Morris, Emmajane Milton, and Ross Goldstone. 2019. Case study: suggesting choice: inclusive assessment processes. *Higher Education Pedagogies* 4, 1 (2019), 435–447.

[24] Briana B Morrison, Brian Dorn, and Mark Guzdial. 2014. Measuring cognitive load in introductory CS: adaptation of an instrument. In *Proceedings of the tenth annual conference on International computing education research*. 131–138.

[25] Huy Nguyen, Michelle Lim, Steven Moore, Eric Nyberg, Majd Sakr, and John Stamper. 2021. Exploring metrics for the analysis of code submissions in an introductory data science course. In *LAK21: 11th International Learning Analytics and Knowledge Conference*. 632–638.

[26] Jake Renzella, Alex Cummaudo, Andrew Cain, John Grundy, and Jonathon Meyers. 2018. SplashKit: A development framework for motivating and engaging students in introductory programming. In *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. IEEE, 40–47.

[27] Emil Stankov, A Madevska Bogdanova, Bojan Ilijoski, and Mile Jovanov. 2018. A survey on software complexity metrics in the context of their application in educational environment. In *INTED2018 Proceedings*. IATED, 9395–9404.

[28] Emil Stankov, Mile Jovanov, and A Madevska Bogdanova. 2017. Improved approach for measuring complexity of code snippets for introductory programming tasks. In *ICERI2017 Proceedings*. IATED, 5892–5899.

[29] Ville Tirronen and Maria Tirronen. 2020. Estimating Programming Exercise Difficulty using Performance Factors Analysis. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–5.

[30] Jacqueline Whalley and Nadia Kasto. 2014. How difficult are novice code writing tasks? A software metrics approach. In *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*. 105–112.

[31] Adam E Wyse, Vincent J Dean, Steven G Viger, and Timothy R Vansickle. 2013. Considerations for equating alternate assessments: Two case studies of alternate assessments based on alternate achievement standards. *Applied Measurement in Education* 26, 1 (2013), 50–72.